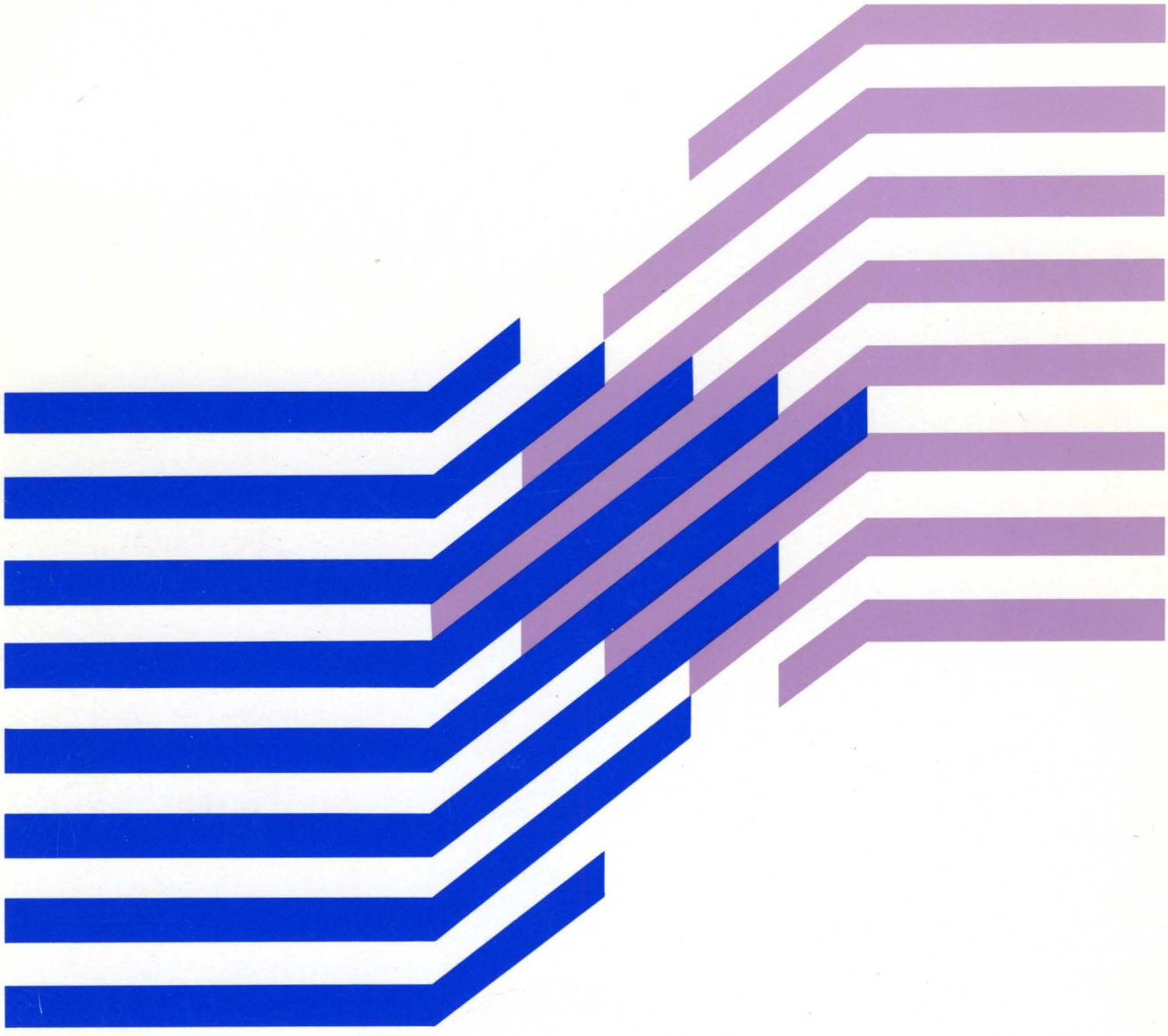
GC28-1857-5

MVS/ESA System Programming Library: Application Development Macro Reference

MVS/System Product:
JES2 Version 3
JES3 Version 3



GC28-1857-5

MVS/ESA System Programming Library: Application Development Macro Reference

MVS/System Product: JES2 Version 3 JES3 Version 3

Sixth Edition (June 1991)

This is a major revision of, and obsoletes, GC28-1857-4. See the Summary of Changes following About this Book for a summary of the changes made to this manual. Technical changes or additions to the text are indicated by a vertical bar to the left of the change.

This edition applies to Version 3 of MVS/System Product 5685-001 or 5685-002 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products or services do not imply that IBM intends to make these available in all countries in which IBM operates. References to IBM products in this document do not imply that functionally equivalent products may be used. The security certification of the trusted computing base that includes the products discussed herein covers certain IBM products. Please contact the manufacturer of any product you may consider to be functionally equivalent for information on that product's security classification. This statement does not expressly or implicitly waive any intellectual property right IBM may hold in any product mentioned herein.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 950, Poughkeepsie, NY 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Special Notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program which does not infringe any of IBM's intellectual property rights may be used instead of the IBM product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Programming Interfaces

This book is intended to help customers to do coding of macros that are available to authorized assembler language programs. It contains detailed information, such as the function, syntax, and parameters, needed to code the macros. This book primarily documents general-use programming interfaces and associated guidance information provided by MVS System Product Version 3.

General-use programming interfaces allow the customer to write programs that request or receive the services of MVS System Product Version 3.

However, this book also documents product-sensitive programming interface information.

Product-sensitive programming interfaces are provided to allow the customer installation to perform tasks such as tailoring, monitoring, modification, or diagnosis of this IBM product. Use of such, interfaces creates dependencies on the detailed design or implementation of the IBM product. Product-sensitive interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implantation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive programming interface information is explicitly identified where it occurs, either as an introductory statement to a chapter or section that is entirely product-sensitive programming interface information, or is marked as follows:

```
┌──────── PRODUCT-SENSITIVE PROGRAMMING INTERFACE ────────┐

                Description of the interface.

└──────── End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE ─┘
```

Trademarks

The following are trademarks of International Business Machines Corporation.

- BookMaster™
- ESA/370™
- Hiperbatch™
- Hiperspace™
- IBM™
- MVS/ESA™
- MVS/SP™
- MVS/XA™

About This Book

This book describes some of the authorized macros that the system provides. Authorized macros are available only to authorized programs — programs that reside in an APF-authorized library or that run in supervisor state with system key 0-7.

Some of the macros included in this book are not authorized, but are included because they are of greater interest to the system programmer than the general applications programmer. Macros are also included in this book if they have one or more authorized parameters — parameters that are available only to authorized programs.

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

Who Should Use this Book

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

The book assumes a knowledge of the computer, as described in *IBM ESA/370 Principles of Operation*, as well as an in-depth knowledge of assembler language programming. Assembler language programming is described in the following books:

- *Assembler H Version 2 Application Programming Guide*, SC26-4036
- *Assembler H Version 2 Application Programming: Language Reference*, GC26-4037

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

How This Book Is Organized

This book includes an introduction that describes information related to all macros. Most of the book, however, consists of descriptions of individual macros. The macro descriptions are presented in alphabetical order. Each description includes:

- A general description of the service that the macro performs.
- A table of syntax rules that you must follow when you code the macro.
- A list of the parameters you can specify and an explanation of each parameter.

How To Use This Book

This book is one of a set of books that describe developing applications in assembler language. This book is the macro reference book for people writing programs that run in supervisor state with system key 0-7 or reside on an APF-authorized library. Use this book to code the macros you need.

The following table shows the books that describe developing applications in assembler language and how this book fits with the others:

Book	Use this book to:
Application Development Guide, GC28-1821	Find out how to use system services provided by macros available to all assembler language programs. If you are relatively new to assembler language programming, this book is a good place to start.
Application Development Macro Reference, GC28-1822	Learn how to code macros that are available to all assembler language programs. This book is for all assembler language programmers.
SPL: Application Development Guide, GC28-1852	Find out how to use system services provided by macros that are available only to programs running in supervisor state with key 0-7 or that are APF-authorized programs. This book is for experienced assembler language programmers; it assumes, for example, that you are familiar with the information in *Application Development Guide*.
SPL: Application Development Macro Reference, GC28-1857	Learn how to code macros that are available only to programs running in supervisor state with key 0-7 or that are APF-authorized programs. This book is for experienced assembler language programmers.
SPL: Application Development — Extended Addressability, GC28-1854	Find out how to use access registers, cross memory services, data spaces, and hiperspaces™ to extend the storage available to programs. This book is for experienced assembler language programmers.
SPL: Application Development 31-Bit Addressing, GC28-1820	Find out how to code assembler language programs that run in 31-bit addressing mode. This book is for all assembler language programmers who need information about developing programs for 31-bit addressing mode.

Related Information

Where necessary, this book references information in other books, using shortened versions of the book title. The following table shows the complete titles and the order numbers:

Title	Order Number
A Structured Approach to Describing and Searching Problems	SC34-2129
The Considerations of Physical Security in a Computer Environment	G520-2700
Data Security Controls and Procedures - A Philosophy for DP Installations	G320-5649
ESA/370: Principles of Operation	SA22-7200
MVS/DFP Version 3 Release 2: General Information **Note:** For the titles and order numbers of referenced DFP books, see this GIM.	GC26-4552
MVS/ESA Diagnosis: Data Areas, Volumes 1 - 5	LY28-1043 to LY28-1047
MVS/ESA Diagnosis: System Reference	LY28-1011
MVS/ESA Diagnosis: Using Dumps and Traces	LY28-1843
MVS/ESA Interactive Problem Control System (IPCS) Command Reference	GC28-1834

Title	Order Number
MVS/ESA Interactive Problem Control System (IPCS) Planning and Customization	GC28-1832
MVS/ESA Interactive Problem Control System (IPCS) User's Guide	GC28-1833
MVS/ESA JCL Reference	GC28-1829
MVS/ESA JCL User's Guide	GC28-1830
MVS/ESA Message Library: System Messages Volume 1 and 2	GC28-1812, GC28-1813
MVS/ESA Message Library: System Codes	GC28-1815
MVS/ESA Operations: JES3 Commands	SC23-0074
MVS/ESA Operations: System Commands	GC28-1826
MVS/ESA Planning: Dump and Trace Services	GC28-1838
MVS/ESA Planning: Global Resource Serialization	GC28-1818
MVS/ESA Service Aids	GC28-1844
MVS/ESA System Programming Library: Application Development 31-Bit Addressing	GC28-1820
MVS/ESA System Programming Library: Initialization and Tuning	GC28-1828
MVS/ESA System Programming Library: System Modifications	GC28-1831
MVS/ESA System Programming Library: Installation Exits	GC28-1836
OS/VS Mass Storage System Extensions Messages	SH35-0041
OS/VS2 MVS RACF Command Language Reference	SC28-0733
Resource Access Control Facility (RACF) General Information Manual	GC28-0722
Resource Access Control Facility (RACF) Macros and Interfaces	SC28-1345
Security Assessment Questionnaire	GX20-2381
System Programming Library: RACF	SC28-1343
MVS System/370 ESA Vector Operations	SA22-7125

Notes:

1. All references to RACF in this publication indicate the program product Resource Access Control Facility.
2. All references to Assembler H in this publication indicate the program product Assembler H Version 2.
3. All references to RMF in this publication indicate the program product Resources Measurement Facility.

Summary of Changes

Summary of Changes
for GC28-1857-5
MVS System Product Version 3 Release 1.3

Changed Information: This revision contains maintenance changes, technical corrections, and services updates.

Summary of Changes
for GC28-1857-4
MVS/System Product Version 3 Release 1.3

New Information: Appendix A contains a list of the names of the macros intended for customer use. The macros identified in this appendix are provided to allow a customer installation to write programs that use the services of MVS. Only those macros identified in this appendix should be used to request or receive the services of MVS.

Changed Information: Numerous services updates have been made throughout the book.

Moved Information: The descriptions for the following macros have been moved to *Application Development Macro Reference*, GC28-1822:

- BLSABDPL
- BLSQMDEF
- BLSQMFLD
- BLSQSHDR
- BLSRDRPX
- BLSRESSY
- BLSRPRD

Summary of Changes
for GC28-1857-3
as updated September 18, 1990
by Technical Newsletter GN28-1437

Changed Information: This technical newsletter contains changes in support of APAR OY27049 as well as maintenance revisions.

Summary of Changes
for GC28-1857-3
as updated February 8, 1990
by Technical Newsletter GN28-1392

Changed Information: This technical newsletter contains maintenance revisions.

Summary of Changes
for GC28-1857-3
MVS/System Product Version 3 Release 1.3

New Information: This revision documents the following new macros:

- CMDAUTH verifies RACF authorization of commands.
- LLACOPY refreshes LLA directories.
- RACROUTE for RACF 1.9.
- COFSDONO causes the data lookaside facility (DLF) to delete a DLF object that is no longer needed.

Changed Information: This revision also documents changes in the following macros:

- TIMEUSED can now be used by unauthorized as well as authorized programs.
- MGCR now passes a user security token to the system.
- The HSPALET parameter on HSPSERV allows a program to take advantage of faster transfer of data between expanded storage and central storage.

- The SHARE parameter on DSPSERV creates a new type of hiperspace named a shared standard hiperspace.
- The DSPSERV LOAD and DSPSERV OUT services allow a program to load an area of a data space into central storage or page an area out from central storage.

Storage

This book uses the term *central storage* for the storage that has been called *real storage*. In the 3090 processor, storage consists of:

Central storage + expanded storage = processor storage

Virtual storage consists of pages contained in processor storage and auxiliary storage.

This revision also includes maintenance throughout the book.

Summary of Changes
for GC28-1857-2
MVS/System Product Version 3 Release 1.0e

New Information: This revision documents the following new macros:

- SCHEDXIT in support of APAR numbers 0Y19162, 0Y19163, 0Y19164, and 0Y19165.

Changed Information: This revision also documents changes in the following macros:

- DSPSERV and ALESERV support SCOPE=COMMON data spaces in response to APAR 0Y20855.
- The EXTEND service of DSPSERV allows a variable request for extension of data space or hiperspace storage. This supports APAR 0Y19885.
- A new service, STGTEST on the SYSEVENT macro, provides information about processor storage.
- A new parameter on the LOAD macro, ADRNAPF, allows a program to load an authorized module into an unauthorized library.
- New parameters on the SDUMP macro allow users to include specific data in a dump and to suppress duplicate SVC dump data.

This revision also includes maintenance throughout the book.

Summary of Changes
for GC28-1857-1
MVS/System Product Version 3 Release 1.0e

New Information: This revision documents the following new macros:

- HSPSERV
- DSPSERV for hiperspaces

Changed Information: This revision also documents changes in the following macros:

- ALESERV
- ETDEF
- DSPSERV for data spaces

Summary of Changes
for GC28-1857-0
MVS/System Product Version 3 Release 1.0

This book contains information previously presented in *MVS/Extended Architecture System Programming Library: System Macros and Facilities, Volume 2* (GC28-1857-4). The following summarizes the changes to that information.

New Information: For MVS/System Product Version 3, this revision describes the following new macros:

ALESERV	COFDEFIN	ETDEF
ASCRE	COFIDENT	ITTFMTB
ASDES	COFNOTIF	RESMGR
ASEXT	COFPURGE	SDUMPX
ATTACHX	COFREMOV	STORAGE
BLSRDRPX	COFRETRI	SYNCHX
BLSRPRD	CTRACE	SYSSTATE
CHANGEKEY	DSPSERV	TCBTOKEN
COFCREAT	ESTAEX	TIMEUSED

Changed Information: For MVS/System Product Version 3, this revision documents changes in the following macros:

ATSET	LOCASCB	SETFRR
ATTACH	PCLINK	SETLOCK
AXFRE	PGSER	SETRP
AXRES	POST	SPLEVEL
AXSET	RACDEF	SRBSTAT
BLSABDPL	RACHECK	SVCUPDTE
BLSQMFLD	RACINIT	SYMREC
CALLRTM	RACLIST	SYNCH
CPOOL	RACROUTE	TCTL
ESTAE	RACXTRT	VSMLIST
ETCON	RESUME	VSMLOC
ETCRE	RISGNL	WTO
FREEMAIN	SCHEDULE	WTOR
GETMAIN	SDUMP	

This revision also includes minor maintenance and editorial changes throughout.

Moved Information: The macros CBPZDIAG, CBPZLOG, CBPZPPDS, IOSDDT, and IOSDMLT are now described only in *System Modifications*.

The macros FRACHECK, RACHECK, RACROUTE, and RACSTAT have moved from part II of *MVS/Extended Architecture Supervisor Services and Macro Instructions (GC28-1154)* to this book.

The VRADATA macro, previously described in *MVS/Extended Architecture System Programming Library: System Macros and Facilities*, has moved to *MVS/ESA™ Application Development Macro Reference*.

Using the Macros

To request system services, programs use macros. The system restricts the use of most of the macros in this book to programs that are in supervisor state with system key 0-7 or that are from an APF-authorized library. A few of these macros are not restricted by the system but are included in this book because your installation might want to restrict the functions they perform. Some macros are totally restricted. Others are not totally restricted but contain one or more parameters that are restricted.

The programs that use the macros in this book must be assembler language programs. When you code a macro, the assembler processes it by using the macro definitions supplied by IBM and placed in the macro library when the system is generated.

The assembler expands the macro into executable machine instructions and/or data fields in the form of assembler language statements. The executable machine instructions typically consist of a branch around the data fields, instructions that load registers, and an instruction that gives control to the system. The instruction that gives control to the system can be a branch, a supervisor call, or a PC instruction. The macro expansion appears as part of the assembler output listing.

The data fields, which are derived from parameters of the macro, are used at execution time by the control program routine that performs the MVS service associated with the macro.

Selecting the Macro Level

MVS/System Product Version 3 (MVS/SP™ Version 3) supports all MVS/System Product Version 2 macros. Therefore, programs that issue macros and that run on a version 2 system should also run on a version 3 system.

There are certain version 3 macros that cannot execute on MVS/System Product Version 1. This means that programs that issue macros and that run on a version 3 system might not run on a version 1 system. A version 1 system cannot process all the macro parameters that work on a version 3 system. These macros are called downward incompatible. When you try to run a version 3 program on a version 1 system, the program might not execute as expected. The macros described in this book that are downward incompatible are:

- ATTACH
- ESTAE
- EVENTS
- FESTAE
- SCHEDULE SCOPE=GLOBAL
- SDUMP
- SETLOCK RELEASE TYPE=ALL
- CALLDISP
- WTOR

Callers executing in 31-bit addressing mode must use the version 2 expansion of the downward incompatible macros.

The SPLEVEL macro solves the problem associated with downward incompatible macros. The SPLEVEL macro allows you to use the version 3 macro library when you assemble programs and to select either the version 1 or version 3 expansion of the macro.

Existing programs that issue version 2 macros will execute properly on a version 3 system. If you change these programs to use new facilities of version 3, change the SPLEVEL macro to specify SET=3. Resetting the SPLEVEL to 3 ensures that your programs use the macro expansion that supports the new facilities.

Before issuing a downward-incompatible macro, a program can specify the macro level by invoking SPLEVEL and using the SET=*n* option.

If *n*=1, the assembler generates the MVS/System Product Version 1 Release 3 expansion of the macro code.

If *n*=2, the assembler generates the version 2 expansion of the macro code.

If *n*=3, the assembler generates the version 3 expansion of the macro code.

A program can also select the level of the macro at execution time, based on the system that is operating. The example in Figure 1 shows one method of selecting the macro level at execution time. The example uses the WTOR macro but would work for any downward incompatible macro. The example first tests the CVTOSEXT bit in CVTDCB and the CVTXAX bit in CVTOSLV0. Both are 1 when MVS/SP Version 3 is operating. If either is 0, then the example tests the CVTMVSE bit in byte CVTDCB of the communications vector table (CVT), which is a 1 when MVS System Product Version 2 is operating.

```
* DETERMINE WHICH SYSTEM IS EXECUTING
*
         TM    CVTDCB,CVTOSEXT
         BNO   SP2CHK
         TM    CVTOSLV0,CVTXAX
         BNO   SP2CHK
*
* INVOKE THE SP3 version OF THE MACRO
*
SP3      SPLEVEL SET=3
         WTOR  ...
         B     CONTINUE
*
SP2CHK   TM    CVTDCB,CVTMVSE
         BNO   SP1
*
* INVOKE THE SP2 version OF THE MACRO
*
SP2      SPLEVEL SET=2
         WTOR  ...
         B     CONTINUE
*
* INVOKE THE SP1 version OF THE MACRO
*
SP1      SPLEVEL SET=1
         WTOR  ...
*
CONTINUE SPLEVEL SET
```

Figure 1. Testing the Macro Level at Execution Time

Addressing Mode and the Macros

A program can execute in 24-bit addressing mode or 31-bit addressing mode. Regardless of the addressing mode that a program executes in, it can invoke most of the macros described in this book, including RACROUTE. However, the following macros require the program to be executing in 24-bit addressing mode and the parameters to be passed in 24-bit addressable storage:

- RACDEF
- RACHECK
- RACINIT
- RACLIST
- SPIE
- STAE

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

- Free storage above 16 megabytes using the FREEMAIN macro
- Allocate storage above 16 megabytes using the GETMAIN macro
- Perform cell pool services for cell pools located in storage above 16 megabytes using the CPOOL macro
- Perform page services for storage locations above 16 megabytes using the PGSER macro

If a program running in 31-bit addressing mode issues a macro, parameter addresses can be above or below 16 megabytes unless otherwise stated in the individual macro description. The macros that have restrictions on parameter addresses above 16 megabytes are:

ATTACH	PGFREEA
CALLRTM	PGSER
CPOOL	PURGEDQ
DEQ	RESERVE
DSGNL	RESUME
ENQ	SDUMP
EXTRACT	STATUS
GETMAIN	TCTL
PGFIX	VSMLIST
PGFIXA	VSMREGN
PGFREE	

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

ATTACH	MODESET
CALLDISP	SETRP
ESTAE	SYNCH
EVENTS	WTOR
FESTAE	

Address Space Control (ASC) Mode

A program can execute in either primary ASC mode or AR (access register) ASC mode. See *SPL: Application Development — Extended Addressability* for more detailed information.

Some macros can generate code that is appropriate for programs in either primary ASC mode or AR ASC mode. A global variable tells these macros which type of code to generate. The SYSSTATE macro allows you to test or to set this variable.

When you assemble a program, the initial value of this variable indicates primary ASC mode. If you do not change the variable, macros that test it will generate code appropriate for primary ASC mode. Thus, if your program receives control in primary ASC mode, you do not need to change the variable. If, however, your program receives control in AR ASC mode, you might have to issue SYSSTATE ASCENV=AR before issuing any macro that tests the variable. To ensure that your programs always generate code appropriate for their ASC mode, IBM recommends that:

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

Once a program has issued SYSSTATE, there is no need to reissue it unless the program switches ASC mode. Figure 4 on page 6 lists the macros that check the SYSTATE global variable.

Using X-Macros

Some MVS services support callers in both primary and AR ASC mode. When the caller is in AR mode, the macro service must generate larger parameter lists; the increased size of the list reflects the addition of ALETs to qualify addresses, as described under "ALET Qualification" on page 5. Some services offer two macros, one for callers in primary mode and one for callers in AR mode. The name of the macro for the AR mode caller is the same as the name of the macro for primary mode callers, except the AR mode macro name ends with an "X". This book refers to these macros as **X-macros**. The X-macros described in this book are:

- ATTACHX
- ESTAEX
- 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.

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 in this book 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 that you always use the ESTAEX macro. However, if your program is in primary mode, where the primary, secondary, and home address spaces are the same, you can use ESTAE.

Passing Parameters in AR Mode

Some macros that you can issue in AR mode include control parameters, user parameters, or both. Control parameters are parameters that control the operation of the macro service (You may also see control parameters referred to as system parameters). User parameters are parameters that the macro service passes to a routine that the macro service invokes on behalf of the caller. For example, the PARAM keyword on the ATTACH macro defines user parameters. The ATTACH macro service passes these parameters to the routine that it attaches. All other parameters on the ATTACH macro are control parameters that control the operation of the ATTACH macro service.

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

- All macros allow you to place parameters in the current primary address space.
- Some macros *require* you to place parameters into the current primary address space.
- Some macros allow you to place parameters in any address space.

Before issuing any macro described in this book, read the macro description to find out where the macro allows parameters to be located.

ALET Qualification

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

The only ALETs that MVS macros accept are:

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

MVS macros do not accept the following ALETs and you must not attempt to pass them to a macro:

- One (1), which signifies that the parameters reside in the caller's secondary address space
- An ALET that is on the caller's primary address space access list (PASN-AL)
- An ALET for a private entry on the PASN-AL or the DU-AL

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

User Parameters

The macro services shown in Figure 2 allow a caller in AR mode to pass information in the form of a parameter list (or parameter lists) to another routine. Figure 2 identifies the parameter that receives the ALET-qualified address(es) of the parameter list(s) and tells you where the target routine finds the ALET-qualified address(es).

Figure 2. Passing User Parameters in AR Mode

Macro	Parameter	Location of User Parameter List Address
ATTACH and ATTACHX	PARAM,VL=1	AR/GPR 1 contains the address of a list of addresses and ALETs. (See Figure 3 for the format of the list.)
ESTAEX	PARAM	SDWAPARM contains the address of an 8-byte area, which contains the address and ALET of the parameter list.

When a caller in AR mode passes ALET-qualified addresses to the called program through PARAM,VL=1 on the ATTACH/ATTACHX macro, the system builds a list formatted as shown in Figure 3. The addresses passed to the called program are at the beginning of the list, and their associated ALETs follow the addresses. The last address in the list has the high order bit on to indicate the size of the list. For example, Figure 3 shows the format of a list where an AR mode issuer of ATTACHX codes the PARAM parameter as follows:

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

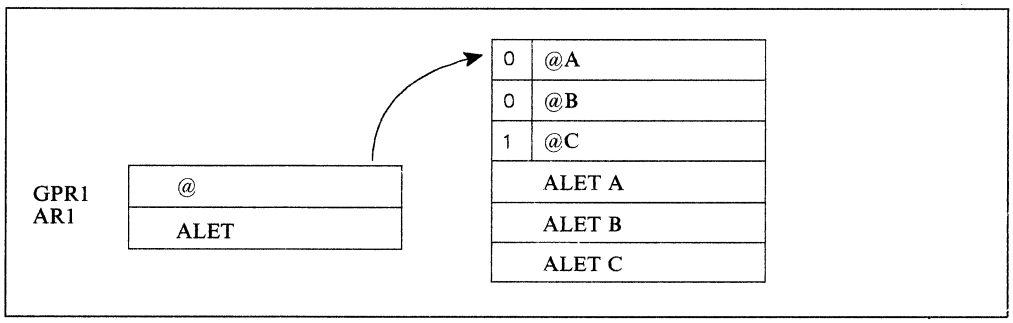


Figure 3. User Parameter List for Callers in AR Mode

Register Usage

After the caller issues a macro, the macro might use some registers as work registers or might change the contents of some registers. When control returns to the caller, each register will contain one of the following values or have the following status:

- The register content is unchanged and is the same as it was before the macro was issued.
- The register contains a value placed there by the macro for the caller's use. Examples of such values are return codes and tokens.
- The macro used the register as a work register. The register content is not the same as it was before the macro was issued and is not meaningful to the caller.

To retain the original contents of registers the macro uses or changes, the caller must save and restore those registers.

Macro Summary

Figure 4 on page 6 lists the macros described in this book. For each macro, the table indicates:

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

Notes:

1. Cross memory mode means that at least one of the following conditions is true:

   PASN≠SASN - The primary address space (PASN) and the secondary address space (SASN) are different.

   PASN≠HASN - The primary address space (PASN) and the home address space (HASN) are different.

   SASN≠HASN - The secondary address space (SASN) and the home address space (HASN) are different.

   For more information about functions that are available to programs in cross memory mode, see *MVS/XA SPL: Application Development Guide*.

2. A program running in primary ASC mode when PASN = SASN = HASN can issue any of the macros listed in the table. If you intend to use ATTACH, SDUMP, or SYNCH, and are in AR mode, IBM recommends that you use the corresponding X-macro (ATTACHX, SDUMPX, and SYNCHX) instead.

Before using any of the macros listed in Figure 4, read the individual macro description to see if any restrictions or limitations apply to your use of the macro.

Figure 4 (Page 1 of 5). Macro Summary

Macro	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
ALESERV	Yes	Yes	No	No
ASCRE	Yes	Yes	Yes	No
ASDES	Yes	Yes	Yes	No
ASEXT	Yes	Yes	No	No
ATSET	No	Yes	No	No
ATTACH	Yes (See note 1)	No	Yes	Yes
ATTACHX	Yes	No	Yes	Yes
AXEXT	No	Yes	No	No

Figure 4 (Page 2 of 5). Macro Summary

Macro	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
AXFRE	No	Yes	No	No
AXRES	No	Yes	No	No
AXSET	No	Yes	No	No
CALLDISP	No	Yes	No	No
CALLRTM	No	Yes	No	No
CHANGKEY	No	No	No	No
CIRB	No	No	No	No
CMDAUTH	No	No	No	No
COFCREAT	Yes	Yes	Yes	Yes
COFDEFIN	Yes	Yes	Yes	Yes
COFIDENT	Yes	Yes	Yes	Yes
COFNOTIF	Yes	Yes	Yes	Yes
COFPURGE	Yes	Yes	Yes	Yes
COFREMOV	Yes	Yes	Yes	Yes
COFRETRI	Yes	Yes	Yes	Yes
COFSDONO	No	No	Yes	Yes
CPOOL	No	Yes	No	No
CTRACE	No	No	Yes	Yes
DATOFF	Yes	No	No	No
DEQ	No	No	No	No
DOM	No	No	No	No
DSGNL	No	Yes	No	No
DSPSERV	Yes	Yes	Yes	Yes
DYNALLOC	No	No	No	No
ENQ	No	No	No	No
ESPIE	No	No	No	No
ESTAE (See note 2 on page 10)	No	No	Yes	Yes
ESTAEX	Yes	Yes	Yes	Yes
ETCON	No	Yes	No	No
ETCRE	No	Yes	No	No
ETDEF	Yes	Yes	No	No
ETDES	No	Yes	No	No
ETDIS	No	Yes	No	No
EVENTS	No	No	No	No
EXTRACT	No	No	No	No
FESTAE	No	No	No	Yes
FRACHECK	No	No	No	No
FREEMAIN	Yes (See note 3)	Yes	Yes	No
GETMAIN	Yes (See note 3)	Yes	Yes	No
GQSCAN	No	Yes	No	No

Figure 4 (Page 3 of 5). Macro Summary

Macro	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
GTRACE	No	No	No	No
HSPSERV	Yes	Yes	(See note 4)	No
IEFQMREQ	No	No	No	No
IOSINFO	No	No	No	No
IOSLOOK	No	No	No	No
ITTFMTB	No	No	No	No
LLACOPY	No	No	Yes	Yes
LOAD	No	No	No	No
LOCASCB	Yes	Yes	Yes	No
LXFRE	No	Yes	No	No
LXRES	No	Yes	No	No
MGCR	No	No	No	No
MODESET	No	Yes	No	No
NIL	No	No	No	No
NUCLKUP	No	No	No	No
OIL	No	No	No	No
OUTADD	No	No	No	No
OUTDEL	No	No	No	No
PCLINK	No	Yes	No	No
PGANY	No	No	No	No
PGFIX	No	Yes	No	No
PGFIXA	No	No	No	No
PGFREE	No	Yes	No	No
PGFREEA	No	No	No	No
PGSER	No	Yes	No	No
POST	No	Yes	No	No
PTRACE	No	Yes	No	No
PURGEDQ	No	No	No	No
QEDIT	No	No	No	No
RACDEF	No	No	No	No
RACHECK	No	No	No	No
RACINIT	No	No	No	No
RACLIST	No	No	No	No
RACROUTE	No	No	No	No
RACSTAT	No	No	No	No
RACXTRT	No	No	No	No
RESERVE	No	No	No	No
RESMGR	Yes	Yes	No	No
RESUME	No	Yes	No	No
RISGNL	No	Yes	No	No
SCHEDULE	Yes	Yes	Yes	Yes
SCHEDXIT	No	Yes	No	No

Figure 4 (Page 4 of 5). Macro Summary

Macro	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
SDUMP	Yes (See note 1)	Yes	Yes	Yes
SDUMPX	Yes	Yes	Yes	Yes
SETFRR	Yes	Yes	Yes	Yes
SETLOCK	Yes	Yes	Yes	Yes
SETRP	Yes	Yes	Yes	No
SPIE	No	No	No	No
SPLEVEL	Yes	Yes	No	No
SPOST	No	No	No	No
SRBSTAT	No	Yes	No	No
SRBTIMER	No	No	No	No
STAE	No	No	No	No
STATUS	No	No	No	No
STORAGE	Yes	Yes	No	No
SUSPEND	No	Yes	No	No
SVCUPDTE	No	No	No	No
SWAREQ	No	No	No	No
SYMREC	No	No	No	No
SYNCH	Yes (See note 1)	No	Yes	No
SYNCHX	Yes	No	Yes	No
SYSEVENT	No	No	No	No
SYSSTATE	Yes	Yes	No	No
TCBTOKEN	Yes	Yes	No	No
TCTL	No	No	No	No
TESTAUTH	No	No	No	No
TIMEUSED	Yes	Yes	No	No
T6EXIT	No	No	No	No
VSMLIST	No	Yes	No	No
VSMLOC	No	Yes	No	No
VSMREGN	No	Yes	No	No
WAIT	No	Yes	No	No
WTL	No	No	No	No
WTO	No	No	No	Yes

Figure 4 (Page 5 of 5). Macro Summary

Macro	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
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. The only programs that can use ESTAE are programs that are in primary mode with (PASN=SASN=HASN). IBM recommends that you always use ESTAEX instead of ESTAE.
3. IBM recommends that AR mode callers use the STORAGE macro instead of using GETMAIN or FREEMAIN.
4. If you use the HSPALET parameter, HSPSERV macro checks SYSSTATE.

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 in-line, 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 in-line parameter list again before the first task regains control. When the first task regains control, it invokes the macro. However, the in-line parameter list now has the wrong values.

A program that runs in a multiprogramming environment can avoid this error by using the list, modify, and execute forms of the macros. One technique is:

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. To change the parameter list, code the modify form of the macro. 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.

Coding the Macros

In this book, each macro description includes a syntax table near the beginning of the macro description. The table shows how to code the macro. The syntax table does not explain the meanings of the parameters; the meanings are explained in the parameter descriptions that follow the syntax table.

The syntax tables assume that the standard begin, end, and continue columns are used. Thus, column 1 is assumed as the begin column. To change the begin, end, and continue columns, use the ICTL instruction to establish the coding format you want to use. If you do not use ICTL, the assembler recognizes the standard columns. To code the ICTL instruction, see *Assembler H Version 2 Application Programming: Language Reference*.

Figure 5 shows a sample macro, TEST, and summarizes all the coding information that is available for it. The table is divided into three columns, A, B, and C.

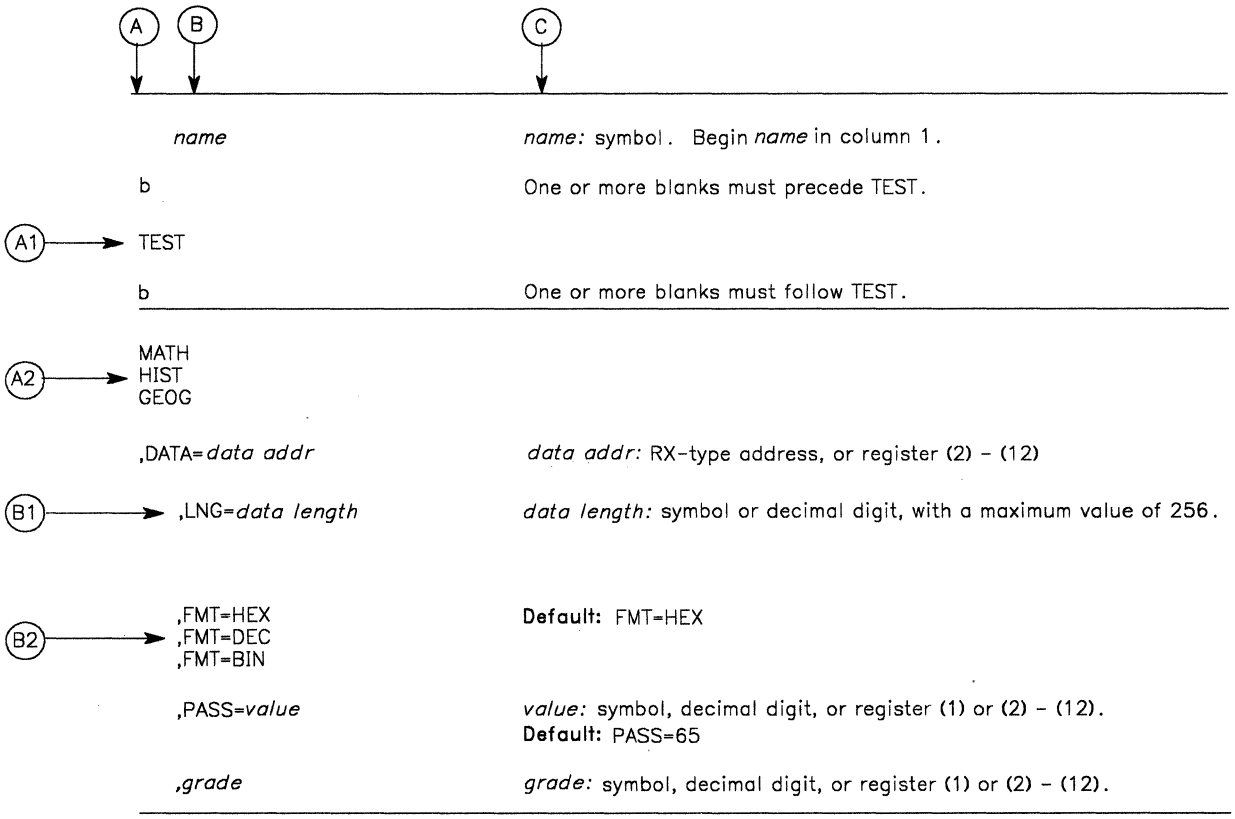


Figure 5. Sample Macro

- The first column, A , contains those parameters that are required for that macro. If a single line appears in that column, A1 , the parameter on that line is required and you must code it. If two or more lines appear together, A2 , you must code the parameter appearing on one and only one of the lines.
- The second column, B, contains those parameters that are optional for that macro. If a single line appears in that column, B1, the parameter on that line is optional. If two or more lines appear together, B2, the entire parameter is optional but, if you elect to make an entry, code one and only one of the lines.
- The third column, C, provides additional information about coding the macro.

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

symbol	any symbol valid in the assembler language. That is, an alphabetic character followed by 0-7 alphameric characters, with no special characters and no blanks.
decimal digit	any decimal digit up to 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.
RX-type address	any address that is valid in an RX-type instruction (for example, LA).
A-type address	any address that can be written in an A-type address constant.
default	a value that is used in default of a specified value; that is, the value the system assumes if the parameter is not coded.

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

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

Continuation Lines

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

1. Enter a continuation character (not blank, and not part of the parameter coding) in column 72 of the line.
2. 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 6 shows an example of each method.

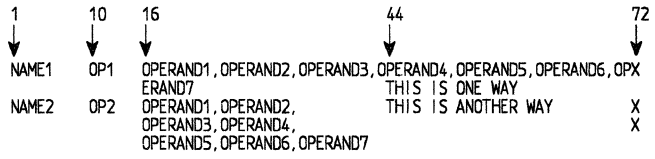


Figure 6. Continuation Coding

ALESERV — Control Entries in the Access List

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, or a hiperspace to which a program (or programs) has access. Access list entry tokens (ALETs) index the entries in the access list.

On the ALESERV macro, address spaces, data spaces, and hiperspaces are identified through STOKENs, an identifier similar to an address space identifier (ASID). *SPL: Application Development — Extended Addressability* 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. See that book for help in using ALESERV.

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)

  Note: You access data spaces and address spaces directly through ESA/370 instructions. You access hiperspaces through the HSPSERV macro.
- 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)

To add a hiperspace entry to an access list, the processor must have the move-page facility installed. If this feature is not on the processor, the system rejects the ALESERV ADD request with an return code X'70'.

The requirements for the caller are:

Authorization:	To request the following ALESERV services, the program must be supervisor state or PSW key 0 - 7: • Make ADD and DELETE requests for the PASN-AL • Use the CHKEAX = NO parameter • Make ADD and DELETE requests for SCOPE = ALL and SCOPE = COMMON data spaces and shared hiperspaces and expanded storage only (ESO) hiperspaces for the DU-AL Problem state programs with PSW key 8 - F can request all other ALESERV services.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary or access register (AR)
Serialization:	Enabled and unlocked for ADD, ADDPASN, and DELETE requests
Control parameters:	Control parameters can reside in any addressable area.

At exit, the ARs and general purpose registers (GPRs) 2 through 13 are preserved. GPR 15 contains the return code. In addition, for SEARCH and EXTRACT requests, GPR 0 contains the reason code for SEARCH and EXTRACT.

ALESERV is also described in *Application Development Macro Reference*, with the exception of the CHKEAX parameter.

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

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

	Valid parameters (Required parameters are underlined)
ADD	AL, STOKEN, ACCESS, ALET, CHKEAX, RELATED
ADDPASN	ALET, RELATED
DELETE	ALET, CHKEAX, RELATED
EXTRACT	ALET, STOKEN, RELATED
SEARCH	AL, ALET, STOKEN, RELATED
EXTRACTH	STOKEN, RELATED
,ACCESS=PUBLIC ,ACCESS=PRIVATE	**Default:** ACCESS=PUBLIC
,AL=WORKUNIT ,AL=PASN	**Default:** AL=WORKUNIT
,ALET=*alet-addr*	*alet-addr:* RX-type address or register (2) - (12).
,STOKEN=*stoken-addr*	*stoken-addr:* RX-type address.
,CHKEAX=YES ,CHKEAX=NO	**Default:** CHKEAX=YES.
,RELATED=*any-value*	*any-value*: Any valid macro parameter specification.

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) and, for address spaces, whether the entry is PUBLIC or PRIVATE (ACCESS parameter). The defaults are DU-AL and PUBLIC.

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

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.

Adding an entry for a hiperspace requires that the processor have the move-page facility installed. If a program issues ALESERV ADD for a hiperspace and the processor does not have the feature, the system rejects the ALESERV ADD request with a return code X'70'.

To ensure the integrity of hiperspaces, the system has certain rules for adding entries for hiperspaces to access lists. The following table summarizes the rules for problem state, PSW key 8 - F programs and supervisor state or PSW key 0 - 7 programs.

Do not use ALESERV ADD for a hiperspace unless you have the move-page facility installed.

Figure 7. Rules for Adding Entries for Hiperspaces to Access Lists

Function	Type of hiperspace	A problem state, key 8 - F program:	A supervisor state or key 0-7 program:
Add entries to the DU-AL	non-shared 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.
	shared standard and ESO	Cannot add entries.	Can add entries.
Add entries to the PASN-AL	Non-shared standard	Cannot add entries.	Can add entries if its PASN-AL is the same as the PASN-AL of the owner's home address space.
	Shared standard and ESO	Cannot add entries.	Can add entries for shared standard hiperspaces. Can add entries for ESO hiperspaces if no unauthorized program can run in the primary address space.

An access list entry for an ESO hiperspace should never 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, 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 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 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.

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

Figure 7 on page 16 describes the rules for adding entries for hiperspaces to the DU-AL and PASN-AL.

,ALET = *alet-addr*
specifies the four-byte ALET. For the ADD and ADDPASN request, ALET specifies the returned ALET for the access list entry that the system added.

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

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

For the SEARCH request, ALET (as input) specifies the point in the access list where the system is to begin the search. The following values are valid as beginning entries:

- Minus One (-1) - Start at the beginning of the DU-AL or PASN-AL.
- Valid ALET - Start the search with the next ALET in the access list.

As output from the SEARCH request, the ALET parameter specifies the searched-for ALET, if present. Otherwise, ALET is unchanged and register 15 contains a reason code that specifies that an ALET for that STOKEN is not on the access list.

,STOKEN = *stoken-addr*
specifies an eight-byte identifier of an address space, data space, 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.

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

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

Code	Meaning
0	ALESERV ADD has completed successfully.
8	The caller is not EAX-authorized to the specified space; the entry is not added. The ALET returned is invalid.
C	The current AL cannot be expanded. There are no free ALEs and the maximum size has been reached.
10	ALESERV could not obtain storage for an expanded access list.
18	The caller tried to add to the PASN-AL in problem state, PSW key 8 - F.
1C	The caller is locked.
20	The caller is disabled.
24	AR 1 contained an ALET of 1 on input or access register 1 contained an ALET for the PASN-AL.
38	The input STOKEN is invalid.
4C	The space represented by the input STOKEN is invalid for cross memory access.
50	Invalid ALESERV parameter list.
54	The caller tried to add a data space to an access list as a private entry.
5C	The caller was not authorized to add a data space or a hiperspace to an access list.
60	An unexpected error occurred. The request was not completed.
64	The caller tried to add an entry using CHKEAX = NO in problem state, PSW key 8 - F.
68	The caller attempted to add a hiperspace under conditions which are not allowed. See Figure 7 on page 16 for a summary of the rules for adding hiperspaces to an access list.
6C	The caller tried to add an entry for a SCOPE = COMMON data space to a DU-AL.
70	The caller tried to add an entry for a hiperspace and did not have the move-page facility installed.

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

Code	Meaning
0	ALESERV ADDPASN has completed successfully.
C	The current AL cannot be expanded. There are no free ALEs and the maximum size has been reached.
10	ALESERV could not obtain storage for an expanded access list.
1C	The caller is locked.
20	The caller is disabled.
24	AR 1 contained an ALET of 1 on input or access register 1 contained an ALET for a PASN-AL.
50	The ALESERV parameter list is invalid.
60	An unexpected error occurred; the request was not completed.

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

Code	Meaning
0	ALESERV DELETE has completed successfully.
8	The caller is not EAX-authorized to the address space specified by the ALET. The entry is not deleted.
14	The input ALET corresponds to an invalid access list entry.
1C	The caller is locked.
20	The caller is disabled.
24	AR 1 contained an ALET of 1 on input or an ALET for the caller's PASN-AL.
28	The caller specified an invalid ALET.
2C	The caller attempted to delete ALET 0, 1, or 2

Code	Meaning
30	A problem state, PSW key 8 - F caller attempted to delete an entry from the PASN-AL.
60	An unexpected error occurred. The request was not completed.
64	The caller tried to delete an entry using CHKEAX=NO in problem state, PSW key 8 - F.

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

Code	Meaning
0	ALESERV EXTRACT has completed successfully. Register 0 contains one of the following reason codes: 00 - The access list entry is a public entry. 04 - The access list entry is a private entry.
14	The input ALET corresponds to an invalid ALE.
24	AR 1 contained an ALET of 1 on input or contains an ALET for the caller's PASN-AL.
28	The caller specified an invalid ALET.
3C	An ALET value of 1 was specified for the ALESERV EXTRACT request.
40	The space associated with the input ALET is invalid for cross memory access.
44	The ALE associated with the input ALET represents addressing capability to a deleted or terminated space.
50	The ALESERV parameter list is invalid.
58	The ALET the caller specified represents an invalid capability.
60	An unexpected error occurred. The request was not completed.

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

Code	Meaning
0	ALESERV SEARCH has completed successfully. Register 0 contains one of the following reason codes: 00 - The access list entry is a public entry. 04 - The access list entry is a private entry.
24	AR 1 contained an ALET of 1 on input or an ALET for the caller's PASN-AL.
28	The caller specified an ALET that is not valid on the specified access list.
34	The caller specified a STOKEN not represented on the specified access list.
48	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.
60	An unexpected error occurred. The request was not completed.

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

Code	Meaning
0	ALESERV EXTRACTH has completed successfully.
24	AR 1 contained an ALET of 1 on input or contains an ALET associated with the caller's PASN-AL.
60	An unexpected error occurred. The request was not completed.

Example of Adding an Entry to a DU-AL

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

```
         ALESERV ADD,STOKEN=DSPCSTKN,ALET=DSPCALET
 *
DSPCSTKN DS   CL8                     DATA SPACE STOKEN
DSPCALET DS   F                       DATA SPACE ALET
```

ALESERV (List Form)

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

The list form is written as follows:

```
   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
```

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.

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

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

	Valid parameters (Required parameters are underlined)
ADD	AL, STOKEN, ACCESS, ALET, CHKEAX, MF, RELATED
ADDPASN	ALET, MF, RELATED
DELETE	ALET, MF, CHKEAX, RELATED
EXTRACT	ALET, STOKEN, MF, RELATED
SEARCH	AL, ALET, STOKEN, RELATED, MF
EXTRACTH	STOKEN, MF, RELATED
,ACCESS=PUBLIC ,ACCESS=PRIVATE	**Default:** ACCESS=PUBLIC
,AL=WORKUNIT ,AL=PASN	**Default:** AL=WORKUNIT
,ALET=*alet-addr*	*alet-addr:* RX-type address or register (2) - (12).
,STOKEN=*stoken-addr*	*stoken-addr:* RX-type address.
,CHKEAX=YES ,CHKEAX=NO	**Default:** CHKEAX=YES.
,RELATED=*any-value*	*any-value*: Any valid macro parameter specification.
,MF=(E,*cntl-addr)*	*cntl-addr*: RX-type address or register (2)-(12).

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

,MF=(E,*cntl addr*)
specifies the execute form, which uses a remote parameter list. *cntl addr* specifies the address of the remote parameter list, generated by the list form of the macro.

ASCRE — Create Address Spaces

The ASCRE macro creates an address space. The address space is full-function; that is, it starts after the system is initialized and has all of the system services. The caller of the ASCRE macro can establish cross memory linkages between the creating address space and the new address space.

Use either the ASNAME or STPARM parameter to name the new address space and specify the first program that will execute in it.

Use the INIT parameter to specify an address space initialization routine to perform such actions as loading modules and building control blocks.

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

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

The requirements for the caller are:

Authorization:	Supervisor state
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary or AR
Serialization:	Enabled and unlocked
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).

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.

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

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	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 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	Used as a work register by the macro
1	Contains a 0 if the return code is 4; otherwise, used as a work register by the macro.
2 - 13	Unchanged
14 - 15	Used as work registers by the macro

See *SPL: Application Development Guide — Extended Addressability* for information on initializing address spaces, which gives an example of creating an address space, including coding the ASCRE macro.

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

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

ASNAME=*as-name* STPARM=*start-parm-addr*	*as-name:* One to seven characters, enclosed in apostrophies. **Note:** Code either ASNAME or STPARM. *start-parm-addr:* RX-type address or register (2) - (12).
,INIT=*init-rtn-addr or init-rtn-name*	*init-rtn-addr:* RX-type address or register (2) - (12). *init-rtn-name*: One to eight characters, enclosed in apostrophes.
,ODA=*output-data-addr*	*output-data-addr:* RX-type address or register (2) - (12).
,TRMEXIT=*rtn name*	*rtn name:* 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 options, 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:** You must also specify LXLIST.
,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.

The parameters are explained as follows:

ASNAME = *as-name*
specifies the address space name (which is 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 contain one to seven characters, enclosed by apostrophes. The first character must be alphabetic or national; other characters can be alphabetic, national, or numeric.

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.

STPARM = *start-parm-addr*
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". It 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.

You must specify either STPARM or ASNAME.

,INIT = *init-rtn-name* **or** *init-rtn-addr*
specifies the address of an eight-character string containing the name of the address space initialization routine. *init-rtn-name* is a string of up to eight alphanumeric characters, enclosed 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 can perform functions such as loading modules, must reside in either the LPA (PLPA, MLPA, fixed LPA) or in SYS1.LINKLIB. 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, it 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 created address space

  If you use the ASDES macro to terminate the created address space, you can obtain the STOKEN from this field.
- Four bytes for the address of the ASCB of the created address space
- Four bytes for the address of the two contiguous ECBs (EAERIMWT and EAEASWT).

  The creator of the address space and the created 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 = *rtn name*
specifies the address of the termination routine — a routine that gets control when the created address space terminates. The routine receives control in 31-bit addressing mode as an asynchronous exit in the creator's address space under the creator's TCB. If you specify UTOKEN, on entry to the routine, register 1 contains the address of a copy of the token specified by the UTOKEN parameter. If the ASDES macro terminates the address space, the termination routine does not receive control.

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 contains the address of a standard 18-word save area.
- 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 created 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 created 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:

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

,AXLIST = *ax-list-addr*
specifies the address of a list of halfwords containing the AX values that ASCRE is to set for the created 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 created 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 created 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 created address space issuing the TKLIST parameter on the ETCON macro.

When you specify TKLIST, you must also specify LXLIST.

,LXLIST = *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 describes 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 created address space issuing the LXLIST parameter on the ETCON macro.

When you specify TKLIST, you must also specify LXLIST.

,RELATED = *value*

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

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

Return code	Reason code	Meaning
0	0	The address space has been created.
0	4	The address space creation has been scheduled.
4	4	The address space has been created synchronously; there was an error accessing the ODA.
4	8	The address space creation is scheduled; there was an error accessing ODA.
8	4	The caller is not in supervisor state.
8	8	The caller is not enabled.
8	12	The caller is not in task mode.
8	16	The caller is not unlocked.
8	20	GRP 0 has an invalid function code on input.
8	24	ASCRE could not establish recovery.
12	4	ASCRE cannot reference the parameter list.
12	8	The parameter list has an invalid version number.
12	12	The reserved field in the parameter list is not 0.
16	4	ASCRE cannot reference the INIT parameter.
16	8	The initialization routine is not specified or is specified incorrectly.
20	4	ASCRE cannot reference the STPARM or ASNAME parameter.
20	8	Neither STPARM or ASNAME was specified.
20	12	The STPARM length is not 1-124.
24	4	The reserved attribute bit is set.
24	8	Both HIPRI and NONURG are specified.
28	4	ASCRE cannot reference the UTOKEN.
28	8	UTOKEN is specified without TRMEXIT.
32	4	ASCRE cannot reference the ASPARM parameter.
32	8	The ASPARM length is not 0-254.
36	4	ASCRE cannot reference AXLIST.
36	8	The AXLIST length is not 1-32 elements.
40	4	ASCRE cannot reference LXLIST.
40	8	The LXLIST length is not 1-32 elements.
44	4	ASCRE cannot reference the TKLIST parameter.
44	8	The TKLIST length is not same as LXLIST length.

Return code	Reason code	Meaning
48	8	The DISPLAY A procedure name is invalid.
52	4	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 re-evaluate your installation-defined storage thresholds.
52	8, 12, 16	Record the return and reason codes and inform your technical support personnel.
56	16	The caller specified an invalid address space.
60, 64, 68, 72	Any	Record the return and reason codes and inform your technical support personnel.

ASCRE (List Form)

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

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

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

ASNAME=*as-name* STPARM=*start-parm-addr*	*as-name:* One to seven characters, enclosed in **Note:** Code either ASNAME or STPARM. *start-parm-addr:* RX-type address. **Note:** Code either ASNAME or STPARM.
,INIT=*init-rtn-addr or init-rtn-name*	*init-rtn-addr:* A-type address. *init-rtn-name*: One to eight characters, enclosed in apostrophes.
,ODA=*output-data-addr*	*output-data-addr:* A-type address.
,TRMEXIT=*rtn-name*	*rtn-name:* 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.
,ATTR=*attribute-list*	*attribute-list:* List of options, separated by commas.
,AXLIST=*ax-list-addr*	*ax-list-addr:* A-type address.
,TKLIST=*token-list-addr*	*token-list-addr:* A-type address. **Note:** You must also specify LXLIST.
,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	

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.

The execute form of the macro is written as follows:

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

ASNAME=*as-name* STPARM=*start-parm-addr*	*as-name:* One to seven characters, enclosed in apostrophes. **Note:** Code either ASNAME or STPARM. *start-parm-addr:* RX-type address or register (2) - (12).
,INIT=*init-rtn-addr or init-rtn-name*	*init-rtn-addr:* RX-type address or register (2) - (12). *init-rtn-name*: One to eight characters, enclosed in apostrophes.
,ODA=*output-data-addr*	*output-data-addr:* RX-type address or register (2) - (12).
,TRMEXIT=*rtn-name*	*rtn-name:* 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 options, 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:** You must also specify LXLIST.
,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)

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.

ASDES — Terminate an Address Space

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

SPL: Application Development — Extended Addressability describes how to create and terminate address spaces.

Requirements for the caller of ASDES are:

Authorization:	Supervisor state
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary or AR
Serialization:	Enabled and unlocked
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).

Additionally, 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.

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

The syntax of the ASDES macro is as follows:

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.

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 codes and reason codes (in decimal form) are in the following table:

Return code	Reason code	Meaning
0	0	Address space is terminated.
8	4	Caller is not in supervisor state.
8	8	Caller is not enabled.
8	12	Caller is not in task mode.
8	16	Caller is not unlocked.
8	20	GPR 0 had invalid function code.
8	24	ASDES could not establish recovery.
12	4	ASDES could not reference the STOKEN parameter.
12	8	STOKEN does not map to a valid address space. Address space might have already terminated.
16	4	The address space was not created by ASCRE.

ASEXT — Extract Address Space Parameters

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 the ASPARM parameter on the ASCRE macro was specified.

The requirements for the caller are:

Authorization:	Supervisor state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = 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
Serialization:	Enabled and unlocked
Control parameters:	The control parameter must be in the primary address space.

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

The syntax of the ASEXT macro is as follows:

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

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

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.

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	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 macro
15	Return code

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

Register	Contents
0	Used as a work register by the macro
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 macro

The return codes and reason codes for AXEXT are as follows:

Return code	Reason code	Meaning
0	0	The ASEXT service has completed successfully.
8	4	The caller is not in supervisor state.
8	8	The caller is not enabled.
8	12	The caller is not in task mode.
8	16	The caller is not unlocked.
8	20	GPR 0 on input has an invalid function code.
8	24	AXEXT is unable to establish recovery.
12	4	GPR 1 has an invalid extract code on input.
16	4	An unexpected error occurred while ASEXT was in progress.

ATSET — Set Authorization Table

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.

To enter ATSET, register 13 must point to a standard register save area addressable in primary mode.

These are the requirements for the caller:

Authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
Control parameters:	Must be addressable in the caller's primary address space

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

This is the standard form of the ATSET macro:

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 ,PT=YES	**Default:** PT=NO
,SSAR=NO ,SSAR=YES	**Default:** SSAR=NO
,RELATED=*value*	*value:* Any valid macro keyword specification.

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.

When control returns, register 15 contains this return code:

Hexadecimal Code	Meaning
0	The selected authorization table entry has been set

ATTACH and ATTACHX— Create a New Task

The ATTACH macro creates a new task. EP, EPLOC or DE indicate the entry point of the new task. The entry point name must be a member name or an alias in a directory of a partitioned data set, or it must have been specified in an IDENTIFY macro. When the specified entry point cannot be located, the new subtask is abnormally terminated.

For information about selecting a macro for an MVS/SP version, other than the current version, see " Selecting the Macro Level" on page 1.

If your program is in access register (AR) mode, use the ATTACHX macro. ATTACH and ATTACHX have the same parameters. However:

- The STAI parameter is not valid for callers in AR mode.
- For callers in AR mode issuing ATTACHX, addresses in the caller's parameter list (on the PARAM parameter) can be in address spaces other than the primary.

This chapter includes information about the ATTACH and ATTACHX macros:

- The syntax of the ATTACH macro, and ATTACH parameters
- The standard form of the ATTACHX macro and callers in AR mode
- The list form of the ATTACH and ATTACHX macros
- The execute form of the ATTACH and ATTACHX macros

These are the requirements for the caller:

Authorization:	Supervisor state or Problem state
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary or AR
Serialization:	Enabled and unlocked
Control parameters:	Parameter lists, and any data pointed to by the parameter lists, must reside in the caller's primary address space. For callers in AR mode, the parameter list address is qualified by an ALET of 0. For callers in AR address space control (ASC) mode, the user's parameter list (PARAM parameter) 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).

On entry to the attached routine, the high order bit, bit 0, of GPR 14 is set to indicate the addressing mode of the issuer of the ATTACH macro. When bit 0 is 0, the issuer is executing in 24-bit addressing mode. When bit 0 is 1, the issuer is executing in 31-bit addressing mode.

The address of the task control block for the new task is returned in GPR 1. 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 new task is posted; an exit routine to be given control, when the new task is terminated; and a parameter list the address of which is passed in GPR 1 to the new task. The subtask is automatically removed from the system upon completion of its execution, unless the ECB or ETXR parameters are coded. When the ECB parameter is specified in the ATTACH macro, the ECB must be in storage. You can wait, using the WAIT macro. The control program can post it on behalf of the terminating task.

The ATTACH macro can specify that ownership of virtual subpools is to be assigned to the new task, or that the subpools are to be shared by the originating task and the new task.

When the issuer is executing in 31-bit, all input parameters to the ATTACH macro can reside in storage above 16 megabytes addressing mode. DCB is an exception.

For a description of the ATTACH, see also *Application Development Macro Reference*. The SM, SVAREA, KEY, DISP, TID, NSHSPV, NSHSPL, and RSAPF parameters are restricted to supervisor state or PSW key 0-7 programs.

This is the standard form of the ATTACH macro:

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* Symbol. *entry name addr:* A-type address, or register (2) - (12). *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* ,PARAM=*addr*,VL=1	*addr:* A-type address, or register (2) - (12). **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* ,GSPL=*subpool list addr*	*subpool nmbr:* Symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,SHSPV=*subpool nmbr* ,SHSPL=*subpool list addr*	*subpool nmbr:* Symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,SZERO=YES ,SZERO=NO	**Default:** SZERO=YES
,TASKLIB=*dcb addr*	*dcb addr:* A-type address, or register (2) - (12).
,STAI=*(exit addr)* ,STAI=*(exit addr,parm addr)* ,ESTAI=*(exit addr)* ,ESTAI=*(exit addr,parm addr)*	*exit addr:* A-type address, or register (2) - (12). *parm addr:* A-type address, or register (2) - (12). **Note**: AR mode callers must not use STAI.
,PURGE=QUIESCE ,PURGE=NONE ,PURGE=HALT	**Note:** PURGE may be specified only if STAI or ESTAI is specified. **Default for STAI:** PURGE=QUIESCE **Default for ESTAI:** PURGE=NONE
,ASYNCH=NO ,ASYNCH=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 ,TERM=YES	**Note:** TERM may be specified only if ESTAI is specified. **Default:** TERM=NO
,SM=PROB ,SM=SUPV	**Default:** SM=PROB
,SVAREA=YES	**Default:** SVAREA=YES

,SVAREA=NO	
,KEY=PROP ,KEY=ZERO	**Default:** KEY=PROP
,DISP=YES ,DISP=NO	**Default:** DISP=YES
,TID=*task id*	*task id:* Decimal digits 0-255, or register (2) - (12). **Default:** TID=0
,NSHSPV=*subpool nmbr* ,NSHSPL=*subpool list addr*	*subpool nmbr:* Symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,RSAPF=NO ,RSAPF=YES	**Default:** RSAPF=NO
,ALCOPY=YES ,ALCOPY=NO	**Default**: ALCOPY=NO
,RELATED=*value*	*value:* Any valid macro keyword specification.

These are the parameters:

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 60-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.

Notes:

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. The BLDL and the ATTACH must be issued from the same task; otherwise, the system terminates the program with an abend code of 106 and a return code of 15. **Do not issue an ATTACH or a DETACH between issuances of BLDL and ATTACH**.

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.

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

Register	Contents
0	Used as a work register by the system.
1	Address of the user parameter list if specified on either the PARAM or MF=E parameters; otherwise unchanged.
2 - 12	Used as work registers by the system.
13	Address of a standard save area.
14	Return address. Bit 0 is 0 if the subtask routine gets control in 24-bit addressing mode; bit 0 is 1 if the subtask routine gets control in 31-bit addressing mode.
15	Entry point address of the subtask routine.

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

Register	Contents
0	Used as a work register by the system.
1	Zero if you specified an user parameter list on either the PARAM or MF=E parameters; otherwise unchanged.
2 - 12	Used as work registers by the system.
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 (255 or less) to be subtracted from the current limit priority of the originating task. The result is the limit priority of the new task. When this parameter is omitted, the current limit priority of the originating task is assigned as the limit priority of the new task.

,DPMOD=*disp prior nmbr*
specifies the signed number (255 or less) to be algebraically added to the current dispatching priority of the originating task. The result is assigned as the dispatching priority of the new task, unless it is greater than the limit priority of the new task. When the result is greater, the limit priority is assigned as the dispatching priority.

When a register is designated, a negative number must be in two's complement form in the register. When this parameter is omitted, the dispatching priority assigned is smaller than the new task's limit priority or the originating task's dispatching priority.

,PARAM=*addr*
,PARAM=*addr*,**VL=1**
specifies the address(es) to be passed to the attached program. Each address is expanded inline to a fullword on a fullword boundary, in the order designated. When the program is given control, Register 1 contains the address of the first word.

VL=1 should be designated when 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 new task. The system uses this to indicate the termination of the new task. The ECB must be in storage. 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 the system 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 new task 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.

These are the contents of the general purpose registers, GPRs, when the exit routine is given control:

Register	Contents
0	Used as a work register by the system
1	Address of the task control block for terminated task
2-12	Used as work registers by the system
13	Address of a save area provided by the system
14	Return address
15	Address of the exit routine

This is the contents of ARs when the exit routine receives control:

Register	Contents
0	Used as a work register by the system
1	Zero
2-12	Used as work registers by the system
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 new task. 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 new task 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
,SZERO = NO
specifies whether subpool 0 is to be shared (YES) or not to be shared (NO) with the subtask.

,TASKLIB = *dcb addr*

specifies the address of the DCB for the library to be used as the attached task's library. Otherwise, the task library is propagated from the originating task. Searching LINKLIB indicates the end of the search. When the DCB address specifies LINKLIB, no other library is searched.

Note: The DCB must be opened before the ATTACH macro is executed and must reside in storage below 16 megabytes.

,STAI = *exit addr*
,STAI = *exit addr,parm addr*
,ESTAI = *exit addr*
,ESTAI = *exit addr,parm addr*

specifies whether or not a STAI or ESTAI SCB is to be created. STAI/ESTAI SCBs queued to the originating task are propagated to the new task.

The *exit addr* specifies the address of the STAI or ESTAI exit routine. This routine receives control when the subtask abnormally terminates. The exit routine must be in virtual storage at the time of abnormal termination. The *parm addr* is the address of a parameter list to be used by the STAI or ESTAI exit routine.

ATTACH processing passes control to the ESTAI exit routine in the addressing mode of the caller of the ATTACH service routine. The ESTAI exit routine can execute in either 24-bit or 31-bit addressing mode. A STAI exit routine can execute only in 24-bit addressing mode. When a caller, in 31-bit addressing mode or in AR mode, specifies the STAI parameter on the ATTACH macro, the caller is abended with an X'52A' completion code.

,PURGE = QUIESCE
,PURGE = NONE
,PURGE = HALT

specifies the action to be taken with regard to I/O operations when the subtask abnormally terminates. NONE indicates that no action is specified. HALT indicates halting of I/O operations. QUIESCE indicates quiescing of I/O operations.

,ASYNCH = NO
,ASYNCH = YES

specifies whether or not asynchronous exits are to be allowed when a subtask abnormally terminates.

ASYNCH = YES must be coded when:

- Supervisor services, that require asynchronous interruptions to complete their normal processing, are to be requested by the ESTAI exit 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 ESTAI exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: An ABEND recursion develops, when ASYNCH = YES is specified, and ABEND is scheduled because of an error in asynchronous exit handling.

,TERM = NO
,TERM = YES

specifies whether or not the exit routine associated with the ESTAI request is scheduled in these situations:

- CANCEL
- Forced LOGOFF
- Job step timer expirations
- Wait time limit for job step exceeded
- ABEND condition. Incomplete task detached when STAE option is not specified on DETACH
- Attaching task abnormally terminates

,SM = PROB
,SM = SUPV

PROB specifies that the system is to run in problem program mode. SUPV specifies that the system is to run in supervisor mode when executing the attached task.

,SVAREA = YES
,SVAREA = NO

specifies whether or not a save area is needed for the attaching task. YES specifies that the ATTACH routine obtains a 72-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 4K-byte block. NO specifies that no save area is needed.

,KEY = PROP
,KEY = ZERO

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 copied from the TCB for the task using ATTACH.

,DISP = YES
,DISP = NO

YES specifies that the subtask is dispatchable. NO specifies that the subtask is nondispatchable.

Note: When DISP = NO is specified, before the ATTACH processing can be completed for the new task, the attaching task must use the STATUS macro to make the task dispatchable.

,TID = *task id*

specifies the task identifier to be placed in the TCB field of the attached task.

,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 = YES
,RSAPF = NO

specifies that the attached subtask comes from an unauthorized library. When it comes from an APF-authorized library and is link-edited with the APF-authorized attribute, the step begins execution with APF authorization.

RSAPF = YES when these conditions are met:

- The caller is running in supervisor state, system key (0-7), or both
- The caller is running non-APF authorized
- The task is attached in the problem program state and with a non-system key.

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

,ALCOPY = NO
,ALCOPY = YES

specifies the EAX value for the new task and determines the contents of its access list. ALCOPY = NO gives the new task an EAX of zero and a null access list. ALCOPY = YES gives the new task:

- The same EAX as the caller
- A copy of the caller's DU-AL

The default is ALCOPY = NO.

,RELATED = (*value*)
specifies information used to self-document 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.

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

Hexadecimal Code	Meaning
00	Successful completion.
04	ATTACH is issued in a STAE exit. Processing not completed.
08	Insufficient storage available for control block for STAI/ESTAI request. Processing not completed.
0C	Invalid exit routine address or invalid parameter list address specified with STAI parameter. Processing not completed.
14	Authorized task specifying JSTCB = YES is not a job step task. Processing not completed.
18	Attempt to create a new subtask results in the job step tasks and non-job step tasks in becoming subtasks of the current task. Processing not completed.

Notes:

1. Upon return, for any return code, register 1 is set to zero. The 00 is an exception.
2. After control is returned to the originating task, the program manager, processing for ATTACH, is performed under the new subtask. It is possible for the originating task to obtain return code 00, and still not have the subtask successfully created. For example, when the entry name cannot be found by the program manager. In such cases, the new subtask is abnormally terminated.

Example 1
Operation: Attach program SYSPROGM, runs with protection key 0 and in supervisor mode. Subpool 0 is not to be shared, and the new task is not to have a save area.

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

Example 2
Operation: Attach the program name addressed in register 7. The new task is to run in problem program mode, a save area is to be provided, subpool 0 is not to be shared, a task library DCB is provided, and the new task is to be nondispatchable.

```
ATTACH EPLOC=(7),SM=PROB,SVAREA=YES,SZERO=NO,    X
  DISP=NO,TASKLIB=(8)
```

Example 3
Operation: 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 4
Operation: 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 5

Operation: 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 dispatchable unit access list DU-AL.

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

ATTACHX — Create a New Task

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

The caller in AR mode must issue the SYSSTATE ASCENV=AR to tell ATTACHX to generate code and addresses appropriate for callers in AR mode. Parameters for the ATTACHX macro are the same as those for the ATTACH macro.

These are the requirements for the caller:

Authorization:	Supervisor state or Problem state
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary or AR
Serialization:	Enabled and unlocked
Control parameters:	Parameter lists, and any data pointed to by the parameter lists, must reside in the caller's primary address space. For callers in AR mode, the parameter list address is qualified by an ALET of 0. For callers in AR address space control (ASC) mode, the user's parameter list (PARAM parameter) 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).

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

System parameter lists, and data pointed to by those parameter lists, must reside in the caller's primary address space. Addresses on the user's parameter list (PARAM parameter) can be in any address space.

This is the standard form of the ATTACHX macro:

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* A-type address, or register (2) - (12). *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* ,PARAM=*addr*,VL=1	*addr:* A-type address, or register (2) - (12). **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* ,GSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,SHSPV=*subpool nmbr* ,SHSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).

,SZERO=YES ,SZERO=NO	**Default:** SZERO=YES
,TASKLIB=*dcb addr*	*dcb addr:* A-type address, or register (2) - (12).
,STAI=(*exit addr*) ,STAI=(*exit addr,parm addr*) ,ESTAI=(*exit addr*) ,ESTAI=(*exit addr,parm addr*)	*exit addr*: A-type address, or register (2) - (12) *parm addr*: A-type address, or register (2) - (12) **Note**: AR mode callers must not use STAI.
,PURGE=QUIESCE ,PURGE=NONE ,PURGE=HALT	**Note:** Specify PURGE only if you specify ESTAI. **Default for ESTAI:** PURGE=NONE
,ASYNCH=NO ,ASYNCH=YES	**Note:** Specify SYNCH only if you specify ESTAI. **Default for ESTAI:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Note:** TERM may be specified only if ESTAI is specified. **Default:** TERM=NO
,SM=PROB ,SM=SUPV	**Default:** SM=PROB
,SVAREA=YES ,SVAREA=NO	**Default:** SVAREA=YES
,KEY=PROP ,KEY=ZERO	**Default:** KEY=PROP
,DISP=YES ,DISP=NO	**Default:** DISP=YES
,TID=*task id*	*task id:* decimal digits 0-255, or register (2) - (12). **Default:** TID=0
,NSHSPV=*subpool nmbr* ,NSHSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,RSAPF=NO ,RSAPF=YES	**Default:** RSAPF=NO
,ALCOPY=YES ,ALCOPY=NO	**Default**: ALCOPY=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under ATTACH. The parameters must be ALET-qualified when a program in AR mode passes a parameter list to the attached task through the PARAM parameter. These are the parameters:

PARAM=*addr*
PARAM=*addr*,**VL=1**

specifies address(es) the caller passes 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.

For programs in AR mode, the addresses passed to the system are in the first half of the parameter list and their associated ALETs are in the last half of the list.

To pass a variable number of parameters, designate VL=1. It tells the system to set the high-order bit of the last address to 1. The 1 in the high-order bit identifies the last address parameter, but not the last entry in the list. For more information about passing user parameters, see " User Parameters" on page 5.

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

Register	Contents
0	Used as a work register by the system.
1	Zero if you specified a user parameter list on either the PARAM or MF=E parameters; otherwise unchanged.
2 - 12	Used as work registers by the system.
13 - 15	Zeroes.

Example

Operation: 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 dispatchable unit access list DU-AL
- Executing in AR ASC Mode.

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

ATTACH and ATTACHX (List Form)

Both, control and user parameter lists are used in the ATTACH and ATTACHX macros. The control parameter list is constructed with the list form of ATTACH or ATTACHX. The user parameter list, is constructed with the list form of the CALL macro. Refer to this parameter list in the execute form of ATTACH or ATTACHX.

The parameter lists for callers in AR mode, for the same number of addresses, are twice the size of the parameter lists for callers in primary mode. The system qualifies each address on the parameter list with an ALET. ALET identifies the address space.

This is the list form of the ATTACH or ATTACHX macro:

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* A-type address. *list entry addr:* A-type address.
,DCB=*dcb addr*	*dcb addr:* A-type address.
,LPMOD=*limit prior nmbr*	*limit prior nmbr:* symbol or decimal digit.
,DPMOD=*disp prior nmbr*	*disp prior nmbr:* symbol or decimal digit.
,ECB=*ecb addr*	*ecb addr:* A-type address.
,ETXR=*exit rtn addr*	*exit rtn addr:* A-type address.
,GSPV=*subpool nmbr* ,GSPL=*subpool list addr*	*subpool nmbr:* symbol or decimal digit. *subpool list addr:* A-type address.
,SHSPV=*subpool nmbr* ,SHSPL=*subpool list addr*	*subpool nmbr:* symbol or decimal digit. *subpool list addr:* A-type address.
,SZERO=YES ,SZERO=NO	**Default:** SZERO=YES
,TASKLIB=*dcb addr*	*dcb addr:* A-type address.
,STAI=*(exit addr)* ,STAI=*(exit addr,parm addr)* ,ESTAI=*(exit addr)* ,ESTAI=*(exit addr,parm addr)*	*exit addr:* A-type address. *parm addr:* A-type address. **Note:** STAI is valid only for callers in primary mode.
,PURGE=QUIESCE ,PURGE=NONE ,PURGE=HALT	**Note:** PURGE may be specified only if STAI or ESTAI is specified. **Default for STAI:** PURGE=QUIESCE **Default for ESTAI:** PURGE=NONE
,ASYNCH=NO ,ASYNCH=YES	**Note:** ASYNCH can be specified only when STAI or ESTAI is specified. **Default for STAI:** ASYNCH=NO **Default for ESTAI:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Note:** TERM can be specified only when ESTAI is specified. **Default:** TERM=NO
,SM=PROB ,SM=SUPV	**Default:** SM=PROB

,SVAREA = YES ,SVAREA = NO	**Default:** SVAREA = YES
,KEY = PROP ,KEY = ZERO	**Default:** KEY = PROP
,DISP = YES ,DISP = NO	**Default:** DISP = YES
,TID = *task id*	*task id:* decimal digits 0-255. **Default:** TID = 0
,NSHSPV = *subpool nmbr* ,NSHSPL = *subpool list addr*	*subpool nmbr:* symbol, decimal digit. *subpool list addr:* A-type address.
,RSAPF = NO ,RSAPF = YES	**Default:** RSAPF = NO
,ALCOPY = YES ,ALCOPY = NO	**Default:** ALCOPY = NO
,RELATED = *value*	*value:* any valid macro keyword specification.
,SF = L	

The parameters are explained under the standard form of the ATTACH or ATTACHX macro. This is the exception:

,SF = L
specifies the list form of the ATTACH or ATTACHX macro.

Note: When the RSAPF parameter is not specified on the list form of ATTACH or ATTACHX, the default is RSAPF = NO. When RSAPF = YES is specified on the list form or on a previous execute form using the same SF = list, RSAPF = NO is ignored for any subsequent execute forms of ATTACH or ATTACHX.

Once RSAPF is specified, it is in effect for all users of that list.

ATTACH and ATTACHX (Execute Form)

Two parameter lists are used in ATTACH and ATTACHX: a control parameter list and an optional user parameter list. Either or both of these parameter lists can be remote and can be referred to and modified by the execute form of ATTACH or ATTACHX. When 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.

This is the execute form of the ATTACH or ATTACHX macro:

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* RX-type address, or register (2) - (12). *list entry addr:* RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr:* RX-type address, or register (2) - (12).
,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* ,PARAM=*addr*,VL=1	*addr:* RX-type address, or register (2) - (12). **Note:** *addr* is one or more addresses, separated by commas. For example, PARAM=*addr,addr,addr*
,ECB=*ecb addr*	*ecb addr:* RX-type address, or register (2) - (12).
,ETXR=*exit rtn addr*	*exit rtn addr:* RX-type address, or register (2) - (12).
,GSPV=*subpool nmbr* ,GSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* RX-type address, or register (2) - (12).
,SHSPV=*subpool nmbr* ,SHSPL=*subpool list addr*	*subpool nmbr*: symbol, decimal digit, or register (2) - (12). *subpool list addr:* RX-type address, or register (2) - (12).
,SZERO=YES ,SZERO=NO	
,TASKLIB=*dcb addr*	*dcb addr:* RX-type address, or register (2) - (12).
,STAI=*(exit addr)* ,STAI=*(exit addr,parm addr)* ,ESTAI=*(exit addr)* ,ESTAI=*(exit addr,parm addr)*	*exit addr:* RX-type address, or register (2) - (12). *parm addr*: RX-type address, or register (2) - (12). **Note:** AR mode callers must not use STAI.
,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 ,TERM=YES	**Note:** TERM may be specified only when ESTAI is specified.
,SM=PROB ,SM=SUPV	**Default:** SM=PROB

,SVAREA=YES ,SVAREA=NO	**Default:** SVAREA=YES
,KEY=PROP ,KEY=ZERO	**Default:** KEY=PROP
,DISP=YES ,DISP=NO	**Default:** DISP=YES
,TID=*task id*	*task id:* decimal digits 0-255, or register (2) - (12). **Default:** TID=0
,NSHSPV=*subpool nmbr* ,NSHSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* RX-type address, or register (2) - (12).
,RSAPF=NO ,RSAPF=YES	**Default:** RSAPF=NO
,ALCOPY=YES ,ALCOPY=NO	**Default:** ALCOPY=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*prob addr*) ,SF=(E,*ctrl addr*) ,MF=(E,*prob addr*),SF=(E,*ctrl addr*)	*prob addr:* RX-type address, or register (1) or (2) - (12). *ctrl addr:* RX-type address, or register (2) - (12) or (15).

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. When no parameter list is provided, user or control parameters are provided in parameter lists expanded inline.

Notes:

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: *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.
5. When RSAPF=YES is specified on the list form or on a previous execute form of the ATTACH or ATTACHX macro using the same SF=list, RSAPF=NO is ignored for any subsequent execute forms of the macro.

AXEXT — Extract Authorization Index

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

These are the requirements for the caller:

Authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
Control parameters:	Control parameters must be in the primary address space.

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

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

This is the standard form of the AXEXT macro:

name	*name:* symbol. Begin *name* in column 1.
ḃ	One or more blanks must precede AXEXT.
AXEXT	
ḃ	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.

These are 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.

When control returns, register 15 contains this return code:

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

AXFRE — Free Authorization Index

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.

These are the requirements for the caller:

Authorization:	Supervisor state or PKM = 0 to 7
Dispatchable unit mode:	SRB or task
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
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. When the macro is issued, the list of AX values passed to the AXFRE macro must also be addressable in primary mode.

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

This is the standard form of the AXFRE macro:

name	*name:* symbol. Begin *name* in column 1.
ḃ	One ore more blanks must precede AXFRE.
AXFRE	
ḃ	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.

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.

When control returns, register 15 contains one of the these return codes:

Hexadecimal Code	Meaning
0	The specified authorization index or indexes are successfully freed.
4	The specified authorization index or indexes are not successfully freed. One or more of the indexes are unavailable for use.

AXRES — Reserve Authorization Index

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.

These are the requirements for the caller:

Authorization:	Supervisor state or PKM = 0 to 7
Dispatchable unit mode:	SRB or task
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
Control parameters:	For callers in primary mode, control parameters must be in the primary address space.

The parameter list passed to the AXRES macro must be addressable in primary mode when the macro expansion is executed. Register 13 must point to a standard register save area addressable in primary mode.

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

This is the standard form of the AXRES macro:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede AXRES.
AXRES	
b	One or more blanks must follow AXRES.
AXLIST=*list addr*	*list addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

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 halfwords 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.

When control returns, register 15 contains this return code:

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

AXSET — Set Authorization Index

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 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.

These are the requirements for the caller:

Authorization:	Supervisor state or PKM = 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
Control parameters:	Primary

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

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

This is the standard form of the AXSET macro:

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.

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.

When control returns, register 15 contains this return code:

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

CALLDISP — Pass Control to Another Ready Task

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 redispatched, control is returned to the next sequential instruction.

These are the requirements for the caller:

- When BRANCH = NO

Authorization:	None
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
Control parameters:	None

- When BRANCH = YES

Authorization:	Supervisor state or PKM = 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Any
Serialization:	Enabled and unlocked
Control parameters:	None

When control returns to the caller:

- The cross memory mode is unchanged.
- When FIXED=NO is specified, registers 14-1 are destroyed; otherwise registers are unchanged.
- No locks are held.
- Control returns enabled.
- PCLINK status is saved and restored.

This is the standard form of the CALLDISP macro:

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

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

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 *SPL: Application Development 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 *SPL: Application Development Guide* for an explanation of the CALLDISP function used with SUSPEND/RESUME processing.

Example 1
Operation: Pass control to another ready task.

```
CALLDISP
```

Example 2
Operation: A non-page-fixed task with an enabled, unlocked task FRR gives control to another ready task. When the task regains control, the contents of registers 14, 15, 0, and 1 have been destroyed.

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

CALLRTM — Call Recovery Termination Manager

The CALLRTM macro schedules abnormal termination for a task or address space. The system selects the appropriate recovery or termination process according to the status of the system and the requests of its invokers.

See also: "Invoking the Recovery Termination Manager" in *SPL: Application Development Guide.*

These are the requirements for the caller:

Authorization:	Key 0, Supervisor state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any cross memory mode
Amode:	Any
ASC mode:	Home, primary, or secondary. When TYPE=ABTERM is specified in a mode other than home mode, ASID must be specified.
Serialization:	• When TYPE=ABTERM and TCB=0, or equals the address of the current task, the caller must be disabled for I/O and external interrupts. • When TYPE=ABTERM and TCB does not equal 0 or the address of the current task, the caller must hold the local lock. • When TYPE=MEMTERM, the user has no disablement or locking requirements.
Control parameters:	Must reside in the currently addressable space. When TYPE=ABTERM and TCB does not equal 0 or the address of the current task, the dump options must reside in fixed or DREF storage.
Savearea	• When TYPE=ABTERM is specified and the ASID parameter is omitted, there are no savearea requirements • When TYPE=ABTERM is specified with ASID, or when TYPE=MEMTERM is specified, the invoker must provide the address of an 18-word savearea in register 13.

When the caller is executing in 31-bit addressing mode, all input parameters, except the TCB, can reside in storage above 16 megabytes.

This is the standard form of the CALLRTM macro:

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

TYPE=ABTERM TYPE=MEMTERM	
,COMPCOD=*comp code*	comp code: symbol, decimal digit, or register (2) - (12).
,REASON=*code*	*code:* a symbol, decimal or hexadecimal number, or register (2) - (12).
,ASID=*asid*	*asid:* decimal digits 0-32,765 or register (2) - (15).
,TCB=*tcb addr*	*tcb addr:* 0, or register (2) - (12). **Note:** This parameter can only be specified with TYPE=ABTERM.
,DUMP=YES ,DUMP=NO	**Default:** DUMP=YES **Note:** This parameter can only be specified with TYPE=ABTERM.

,STEP=NO ,STEP=YES	**Default:** STEP=NO **Note:** This parameter can only be specified with TYPE=ABTERM.
,DUMPOPT=*parm list addr* ,DUMPOPX=*parm list addr*	*parm list addr:* register (3)- (15). *parm list addr:* register (3)-(15).

These are the parameters:

TYPE=ABTERM
TYPE=MEMTERM
specifies whether the services of the recovery termination manager are directed towards task termination (ABTERM) or address space termination (MEMTERM). For MEMTERM, all recovery processing is skipped in the address space.

TYPE=ABTERM is supported in home mode when ASID is specified.

To specify TYPE=ABTERM and TCB=0, the user must be disabled for I/O and external interrupts. To specify TYPE=ABTERM but not TCB=0, the user must be holding the local lock.

In a cross memory environment, when ASID is not specified, the TCB must reside in the home address space. When ASID is specified, the TCB must be in the same address space as the ASCB.

,COMPCOD=*compcode*
specifies the system completion code associated with the abnormal termination. This parameter can be specified as a hexadecimal code (x'80A'.), a decimal code (2058), or a register containing a hexadecimal code. In all cases, the result is hexadecimal.

,REASON=*code*
specifies additional 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 uses the SDWACRC field of the SDWA to pass the reason code to the recovery routine.

,ASID=*asid*
specifies the address space identifier of the address space to be terminated (for MEMTERM) or the address space identifier of the address space containing the TCB of the task to be terminated (for ABTERM). When this parameter is omitted or when zero is specified, the current address space is assumed. When this parameter is specified, an 18-word work area must be supplied and its address must be passed in register 13.

Note: The contents of register 2 is destroyed when this parameter is used.

,TCB=*tcb addr*
specifies the TCB address of the task to be terminated. In a cross memory environment, when ASID is not specified, the TCB must reside in the home address space. When ASID is specified, the TCB must be in the same address space as the ASCB.

Note: The TCB resides in storage below 16 megabytes.

,DUMP=YES
,DUMP=NO
specifies whether a dump is (YES) or is not (NO) to be taken. When the DUMPOPT or DUMPOPX parameter is not specified, the contents of the dump are defined by the //SYSABEND, //SYSMDUMP, or //SYSUDUMP DD statement and the system or user-defined defaults.

,STEP=NO
,STEP=YES
specifies whether the job step is (YES) or is not (NO) to be abnormally terminated.

,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.

The system dump options, 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, 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 DUMPOPT or DUMPOPX parameter is ignored.

Note: The contents of register 3 is destroyed when this parameter is used.

Register 15 contains one of these return codes for TYPE = ABTERM:

Hexadecimal Code	Meaning
0	The ABTERM request was processed successfully.
4	The task has already been scheduled for termination by a previous ABTERM request.
8	An asynchronous unit of work has been scheduled to terminate the task.
18	The ASID value is invalid.
1C	The TCB passed to RTM is invalid.
28	An SRB could not be obtained.

Register 15 contains one of these return codes for TYPE = MEMTERM:

Hexadecimal Code	Meaning
0	The MEMTERM request was processed successfully.
18	The ASID value is invalid.
2C	The address space cannot be terminated by the MEMTERM request.

Example 1

Operation: Terminate the current address space with a completion code of 123.

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

Example 2

Operation: 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)
```

CHANGKEY — Change Virtual Storage Protection Key

The CHANGKEY macro changes the protection key and fetch protection status of one or more pages of virtual storage. The CHANGKEY function is available only for use by programs that execute in supervisor state and key zero. Callers can be enabled or disabled.

CHANGKEY is valid for virtual storage that is obtained by a GETMAIN or a STORAGE macro. The storage must be obtained in page multiples from subpools that are available to programs in problem program state. Callers must provide an 18-word save area and place the address of the save area in register 13. If the caller is disabled, the save area must be in fixed storage.

The CHANGKEY macro is written as follows:

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

R,BA=*page addr*,EA=*page addr* L,LISTAD=*list addr*	*page addr:* A-type address or register (1) - (12). **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).
,BRANCH=YES	Required.

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.

Notes:

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 addressing mode of the issuer of the macro.

LISTAD specifies the address of the first double-word 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.

,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 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.

Note: The requestor must have addressability to the CVT.

Upon completion of the CHANGKEY macro, register 15 contains a zero return code. If a caller requested that the key be changed to key 0, the caller is abended with a code X'08F'.

Example 1

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

```
CHANGKEY R,BA=(REG5),EA=(REG5),KEY=8,BRANCH=YES
```

Example 2

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

```
       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
```

CIRB — Create Interruption Request Block

The CIRB macro causes the exit effector routine to create an interruption request block (IRB). Other parameters of this macro specify the building of a register save area and/or a work area to contain interruption queue elements. For information about asynchronous exit routines, see *SPL: Application Development Guide*.

These are the requirements for the caller:

- When BRANCH = NO

Authorization:	None
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	Enabled and unlocked
Control parameters:	Must be in primary address space

- When BRANCH = YES

Authorization:	Supervisor state and PSW key = 0
Dispatchable unit mode:	Task or IRB
Cross memory mode:	PASN = HASN
Amode:	Any
ASC mode:	Primary
Serialization:	LOCAL lock
Control parameters:	Must be in primary address space

For BRANCH=YES:

- The caller must pass a TCB address 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.

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

This is the standard form of the CIRB macro:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CIRB.
CIRB	
b	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 ,KEY=SUPR	**Default:** KEY=PP
,MODE=PP ,MODE=SUPR	**Default:** MODE=PP
,SVAREA=NO ,SVAREA=YES	**Default:** SVAREA=NO
,RETIQE=YES ,RETIQE=NO	**Default:** RETIQE=YES
,STAB=(DYN)	
,WKAREA=*workarea size*	*workarea size:* Decimal digit, or register (2) - (12). **Default:** zero
,BRANCH=NO ,BRANCH=YES	**Default:** BRANCH=NO
,RETRN=NO ,RETRN=YES	**Default:** RETRN=NO **Note:** This parameter has meaning only when RETIQE=NO is specified.
,AMODE=CALLER ,AMODE=DEFINED	**Default:** AMODE=CALLER

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.

,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 by EXIT.

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 double words.

,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) returned to the available queue 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.

Example 1
Operation: 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
Operation: 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
```

CMDAUTH — Command Authorization Service

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

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

The requirements for the caller are:

Authorization:	One of the following: • Supervisor state and key 0 • APF authorized
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

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

Code	Meaning
0	Command issuer is authorized to issue the command.
4	No authorization decision was made.
8	Command issuer is not authorized to issue the command.

When control is returned from CMDAUTH, register 0 contains a return code from the security product that is installed on the system. If the security product is RACF, see the description of the return codes listed with "RACROUTE REQUEST=AUTH — Check RACF Authorization (for RACF Release 1.9)" on page 477 for further 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	Return code from the security product.
1	Address of error messages if MSGRTN=YES is specified; otherwise, used as a work register by the macro.
2 - 13	Unchanged
14	Used as a work register by the macro
15	Return code from CMDAUTH.

CMDAUTH (List Form)

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

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

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

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.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CMDAUTH.
CMDAUTH	
b	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 ,CBLKTYPE=SSCM	**Note:** See usage note (following) for usage information.
,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 ,MSGSUPP=NO	**Default:** NO
,MSGRTN=YES ,MSGRTN=NO	**Default:** NO
,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.

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

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

CBLKTYPE = SSCM

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

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.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

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

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,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.

,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.

Example

Operation: 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)
```

COFCREAT — Create a VLF Object

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 *SPL: Application Development Guide*.

When you issue COFCREAT to create a VLF object, you must provide the UTOKEN for a currently-identified end user, as well as the major name, minor name, and object parts list for the object to be placed in VLF storage. The object parts list describes the ALET, location, and size of each source area that is to be made part of the VLF object. To issue COFCREAT, your program must be running under a task with the same home ASID as the issuer of the COFIDENT macro that identified the user.

To ensure that VLF does not create an object if the permanent source data changes between the time you obtain the object from permanent storage and the time you create the object, VLF requires that you issue COFRETRI to try to retrieve the object before you issue COFCREAT.

Thus, 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.

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. VLF guarantees that no manipulations with allocation can allow the user to alter the data sets associated with a DDNAME used to identify a VLF user. In such a case, 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. Perform secure I/O to read the object from DASD. Performing secure I/O, which protects the data from malicious tasks, has the following requirements:
   a. The DCB used for I/O must not be in key 8 storage.
   b. 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 rules 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.

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 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.

Environment:

To invoke COFCREAT, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

Restrictions:

- If you do not specify REPLACE, you must issue the COFRETRI macro before you issue COFCREAT on behalf of the user.
- If you specify REPLACE, you must ensure that the source object cannot change until VLF has processed the COFCREAT macro for the object.

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

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

MAJOR=*major* CINDEX=*cindex*	*major*: Rx-type address or register (2) - (12). *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 ,REPLACE=NO	Specify REPLACE only with MAJOR. Default: REPLACE=NO
,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).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).

The parameters of the standard form 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 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*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary,

,RSNCODE = *rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary.

Return Codes and Reason Codes

The hexadecimal return codes from COFCREAT are as follows:

Return code	Meaning
00	The CREATE function completed without error.
02	No VLF object was created. See reason codes for details.
04	The requested major name is not in the user's search order.
0A	The parameter list cannot be accessed.
0C	The class to which the user is identified is not currently defined.
10	A user token was specified but the user is not currently identified to VLF.
12	The DDNAME is not the same as the DDNAME specified on the COFIDENT macro that returned this user token.
14	VLF incurred a program check when it tried to access the object parts list.
18	An input parameter contained an invalid value.
1C	There was not enough storage available to create this object.
28	VLF is not active.
2C	There was an unexpected error in VLF.

The hexadecimal reason codes from COFCREAT are as follows:

Reason code	Return code	Meaning
00	00	The VLF object has been created.
02	02	No VLF object was created because the create request specified an ineligible major name.
04	02	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.
nn5000	14	VLF was unable to access OBJPRTL(nn), where *nn* is a hexadecimal number indicating the part in which the access failure occurred.
00	18	The class to which the user is identified is a PDS class, but CINDEX was not specified.
02	18	OBJPLSZ was larger than the maximum allowable size, or the number of parts in the object parts list was greater than 16.
04	18	REPLACE was specified, but the class to which the user is identified is a PDS class.
0A	18	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).
0B	18	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).
0C	18	The object parts list 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).
0D	18	A part in the object parts list 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).
nnnn	2C	The reason code associated with return code X'2C' (44 decimal) is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

COFCREAT (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string Default: 0D

The parameters of the list form are explained as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFCREAT (Execute Form)

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

name	*name*: symbol. Begin name in column 1.
b	One or more blanks must precede COFCREAT
COFCREAT	
b	One or more blanks must follow COFCREAT
MAJOR=*major* ,CINDEX=*cindex*	*major*: Rx-type address or register (2) - (12). *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 ,REPLACE=NO	Specify REPLACE only with MAJOR. 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,*mfctrl*)	*mfctrl*: Rx-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFCREAT macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFDEFIN — Define a VLF Class

COFDEFIN creates a class of VLF objects. Before issuing COFDEFIN, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *SPL: Application Development 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.

To obtain the attributes of the class, the system uses the input parameters of the macro and the description of the class in the active COFVLFxx parmlib member. The maximum amount of virtual storage available for the class can be controlled by the MAXVIRT keyword on the CLASS statement in COFVLFxx. When the MAXVIRT keyword is not used, the default is 4096 pages.

Environment:

To invoke COFDEFIN, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

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

name	*name*: symbol. Begin name in column 1.
b	One or more blanks must precede COFDEFIN
COFDEFIN	
b	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 ,TRIM=OFF	**Default**: ON
,AUTHRET=NO ,AUTHRET=YES	**Default**: NO
,RETCODE=*retcod*	*retcod*: RX-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: RX-type address or register (2) - (12).

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*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary,

,RSNCODE = *rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary,

Return Codes and Reason Codes

The hexadecimal return codes from COFDEFIN are as follows:

Return code	Meaning
00	The class is defined to VLF.
02	A define request for this class is already in progress or the class is already defined.
04	The define request failed. The class state is not valid.
08	A purge request for the same class has overridden the define request. The class is not defined at this time.
0C	There was no description for the class in the active COFVLFxx parmlib member.
10	One or more parameters are not valid. See the reason code.
18	The parameter list ALET is not valid.
28	VLF is not active.
2C	There was an unexpected error in VLF.

The hexadecimal reason codes from COFDEFIN are as follows:

Reason code	Return code	Meaning
00	00	The define request was successful.
04	02	A define request for the same class is currently in progress.
08	02	The class is already defined. You must issue COFPURGE for the class before you can redefine the class.
0C	02	The class is already defined. You must issue COFPURGE for the class before you can redefine the class. 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.
00	08	A purge request for the same class was issued before the define request completed.
04	08	The class was being purged when you issued COFDEFIN.
04	10	The value for MAJLEN is not within the allowed range.
08	10	The value for MINLEN is not within the allowed range.
0C	10	The values for both MAJLEN and MINLEN are not within the allowed range.
nnnn	2C	The reason code associated with return code X'2C' (44 decimal) is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

COFDEFIN (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string Default: 0D

The parameters of the list form are as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFDEFIN (Execute Form)

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

name	*name*: symbol. Begin name in column 1.
b	One or more blanks must precede COFDEFIN
COFDEFIN	
b	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 ,TRIM=OFF	**Default**: ON
,AUTHRET=YES ,AUTHRET=NO	**Default**: NO
,RETCODE=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: RX-type address or register (2) - (12).
,MF=(E,*mfctrl*)	*mfctrl*: RX-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFDEFIN macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFIDENT — Identify a VLF User

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 *SPL: Application Development 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.

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.

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:

To invoke COFIDENT, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

Restrictions:

- 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 SCOPE=HOME is specified or defaulted, 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 SCOPE=SYSTEM is specified, 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 mask 0-7, can retrieve objects from the class.

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

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

DDNAME=*ddname* MAJNLST=*majnlst*	*ddname*: Rx-type address or register (2) - (12). *majnlst*: Rx-type address or register (2) - (12).
,CLASS=*class*	*class*: Rx-type address or register (2) - (12).
,SCOPE=HOME ,SCOPE=SYSTEM	**Default**: HOME
,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).

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 seven-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. Subsequent requests to create or retrieve VLF objects on behalf of this user must supply this token to VLF.

,RETCODE = *retcod*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE = *rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary.

Return Codes and Reason Codes

The hexadecimal return codes from COFIDENT are as follows:

Return code	Meaning
00	Successful completion. The user has been identified to VLF with the specified major name search order.
02	The user is already identified to VLF for this class.
04	The identify request cannot be completed. Another identify request from the same home ASID is currently in progress for the same class and DDNAME.
08	No major names in the search order contain objects that are eligible objects for VLF.
0C	The class has not been defined to VLF.
10	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.
14	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.
18	There was an error in the parameter list.
1C	An error was detected during processing of the DDNAME for a PDS class. See the reason codes.
28	VLF is not active.
2C	There was an unexpected error in VLF.

The hexadecimal reason codes from COFIDENT are as follows:

Reason code	Return code	Meaning
00	00	Successful completion. The user has been identified to VLF with the specified major name search order.
08	02	The user is already identified to VLF for this class. The user token from the previous IDENTIFY has been returned in the UTOKEN field.
08	18	The number of major names in a search order is not in the range 1-256.
04	1C	The DDNAME was not open.
08	1C	The DDNAME was not allocated.
12	1C	The DDNAME concatenation was changed without deallocating the DDNAME. VLF no longer accepts user identification requests that specify the DDNAME.
0C	18	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).
nnnn	2C	The reason code associated with return code X'2C' (44 decimal) is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

COFIDENT (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string. Default: 0D

The parameters of the list form are explained as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFIDENT (Execute Form)

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

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

DDNAME=*ddname* MAJNLST=*majnlst*	*ddname*: Rx-type address or register (2) - (12). *majnlst*: Rx-type address or register (2) - (12).
,CLASS=*class*	*class*: Rx-type address or register (2) - (12).
,SCOPE=HOME ,SCOPE=SYSTEM	**Default**: HOME
,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,*mfctrl*)	*mfctrl*: Rx-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFIDENT macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFNOTIF — Notify VLF

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 *SPL: Application Development 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:

To invoke COFNOTIF, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

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

---

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

FUNC=DELMAJOR
FUNC=DELMINOR
FUNC=ADDMINOR
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 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*: 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).

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 the name of 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*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE = *rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary.

Return Codes and Reason Codes

The hexadecimal return codes from COFNOTIF are as follows:

Return code	Meaning
00	Successful completion. VLF now reflects the indicated changes.
02	No changes to VLF storage occurred.
18	Parameter list error. See the reason codes.
1C	An error occurred while accessing an input major name. The reason code indicates the position of the major name in the input major name list.
20	An error occurred while accessing an input minor name. The reason code indicates the position of the minor name in the input minor name list.
28	VLF is not active.
2C	There was an unexpected error in VLF.

The hexadecimal reason codes from COFNOTIF are as follows:

Reason code	Return code	Meaning
00	00	VLF now reflects the indicated changes.
08	02	No changes to VLF storage were necessary.
0C	02	The specified class was not defined to VLF. This code is only returned for an input class that does not have a major name to PDS correspondence.
10	02	The specified class is not defined in the active COFVLFxx parmlib member.
00	18	The parameter list ALET is either a SASN ALET or is not on the caller's dispatchable unit access list (DU-AL).
08	18	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).
0C	18	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).
nnnn	1C	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.
nnnn	20	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.
nnnn	2C	The reason code associated with return code X'2C' (44 decimal) is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

COFNOTIF (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string Default: 0D

The parameters of the list form are explained as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFNOTIF (Execute Form)

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

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

FUNC=DELMAJOR FUNC=DELMINOR FUNC=ADDMINOR 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 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,*mfctrl*)	*mfctrl*: Rx-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFNOTIF macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFPURGE — Purge a VLF Class

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 *SPL: Application Development 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 reidentify the users of the class.

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:

To invoke COFPURGE, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

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

name	*name*: symbol. Begin name in column 1.
ḃ	One or more blanks must precede COFPURGE
COFPURGE	
ḃ	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).

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*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE = *rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary.

Return Codes and Reason Codes

The hexadecimal return codes from COFPURGE are as follows:

Return code	Meaning
00	Successful completion. The class is no longer described to VLF.
02	The specified class was not described in the active COFVLFxx parmlib member.
28	VLF is not active.

The hexadecimal reason codes from COFPURGE are as follows:

Reason code	Return code	Meaning
00	00	The purge was successful.
04	02	The specified class was not described in the active COFVLFxx parmlib member.

COFPURGE (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string. Default: 0D

The parameters of the list form are as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFPURGE (Execute Form)

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

name	*name*: symbol. Begin name in column 1.
b	One or more blanks must precede COFPURGE
COFPURGE	
b	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,*mfctrl*)	*mfctrl*: RX-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFPURGE macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFREMOV — Remove a VLF User

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 *SPL: Application Development 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:

To invoke COFREMOV, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

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

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).

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*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary.

Return Codes and Reason Codes

The hexadecimal return codes from COFREMOV are as follows:

Return code	Meaning
0	Successful completion. The record of the identified user corresponding to the input UTOKEN has been removed. Subsequent requests to access VLF objects with this UTOKEN will fail.
02	An unknown user token was specified.
18	The ALET of the input parameter is not valid.
28	VLF is not active.
2C	There was an unexpected error in VLF.

The hexadecimal reason codes from COFREMOV are as follows:

Reason code	Return code	Meaning
0	0	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.
10	02	An unknown user token was specified.
nnnn	2C	The reason code associated with return code X'2C' (44 decimal) is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

COFREMOV (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string. Default: 0D

The parameters of the list form are explained as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFREMOV (Execute Form)

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

name	*name*: symbol. Begin name in column 1.
b	One or more blanks must precede COFREMOV
COFREMOV	
b	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*	*retcod*: Rx-type address or register (2) - (12).
,MF=(E,*mfctrl*)	*mfctrl*: Rx-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFREMOV macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFRETRI — Retrieve a VLF Object

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 *SPL: Application Development Guide*.

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:

- It returns the user token you must supply on COFRETRI.
- It establishes the major name search order for this user.
- It 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).

Environment:

To invoke COFRETRI, a program can be in cross memory mode and must be:

- In enabled, unlocked task mode with no EUT FRRs in effect.
- In supervisor state or with PSW key mask 0-7. If the COFDEFIN macro that defined the class specified (or defaulted to) AUTHRET=NO, the program can be in problem state.
- In either 24-bit or 31-bit addressing mode, but all addresses passed must be full 31-bit addresses.
- In primary address space control (ASC) mode or access register ASC mode.

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

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

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 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 is performed to locate a member.

,RETCODE = *retcod*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE = *rsncod*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 0. If you specify a storage location, it must be on a fullword boundary.

Return Codes and Reason Codes

The hexadecimal return codes from COFRETRI are as follows:

Return code	Meaning
00	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).
02	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). Issue a BLDL 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. First, use the alternate method to acquire the object for the user. Second, issue a COFCREAT macro to create the VLF object.
04	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). Increase the size of the target area, then issue COFRETRI again.
06	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). 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	The object does not exist in VLF.
0A	The parameter list cannot be accessed.
0C	The class to which the user is identified is not currently defined.
0E	The user has insufficient authorization. To retrieve an object for the class, the caller must be a task running in supervisor state or with PSW key mask 0-7.
10	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. In either case, you must issue the COFIDENT macro; you must re-identify the user to VLF before you can retrieve objects for the user.
14	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.
18	An input parameter contains an invalid value.
28	VLF is not active.
2C	There was an unexpected error.

The hexadecimal reason codes from COFRETRI are as follows:

Reason code	Return code	Meaning
00	08	VLF could not find a matching object to retrieve.
04	08	A retrieve was attempted for a major name that has changed or been deleted.
nn4000	14	VLF could not access TLIST(nn), where *nn* is a hexadecimal number indicating the part in which the access failure occurred.
02	18	TLSIZE is greater than the maximum allowable size, or the number of target areas is greater than 16.
0B	18	The object specified on MINOR 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).
0C	18	TLIST 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).
0D	18	A target area in the target list 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).
nnnn	2C	The reason code associated with return code X'2C' (44 decimal) is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

COFRETRI (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *mfattr*: 1- to 60-character input string. Default: 0D

The parameters of the list form are explained as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFRETRI (Execute Form)

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

name	*name*: symbol. Begin name in column 1.
b	One or more blanks must precede COFRETRI
COFRETRI	
b	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,*mfctrl*)	*mfctrl*: Rx-type address or register (2) - (12).

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

,MF=(E,*mfctrl*)
Specifies the execute form of the COFRETRI macro. This form generates the code to store the parameters into the parameter list and execute the function of creating a new class of VLF objects. *mfctrl* specifies the location of the parameter list.

COFSDONO — Delete DLF (Data Lookaside Facility) Object

Use the COFSDONO macro to cause DLF to delete a DLF object that is no longer needed.

The requirements for the caller are:

Authorization:	Supervisor state or with PSW key mask 0-7.
Dispatchable unit mode:	Task mode.
Cross memory mode:	PASN=HASN
Amode:	31-bit addressing.
ASC mode:	Primary ASC mode.
Serialization:	Enabled.
Control parameters:	

Note: Use of hiperbatch requires expanded storage and a processor that has the move-page facility installed.

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

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

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 one to 44-character data set name, left-justified. Pad the 64-character field on the right with blanks (X"40'").

,RETCODE=*ret addr*
an optional parameter that specifies the location where the system is to place the return code. The system copies the return code into the location from register 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsn addr*
an optional parameter that specifies the location where the system is to place the reason code. The system copies the reason code into the location from register 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.

When control is returned, register 15 contains a hexadecimal return code and register 0 contains a hexadecimal reason code, as follows:

Return Code	Reason Code	Meaning
0	0	Success. The DLF object has been deleted.
2	0	The object did not exist in DLF.
28	0	DLF is not active.
2C	(*nnnn*)	Unexpected error in DLF. The reason code associated with return code X'2C' is for IBM only. It should be recorded and supplied to the appropriate IBM support personnel.

COFSDONO (List Form)

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

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

MF=(L,*mfctrl*) MF=(L,*mfctrl*,*mfattr*)	*mfctrl*: symbol. *attr*: 1- to 60-character input string. Default: 0D

The parameters of the list form are explained as follows:

MF=(L,*mfctrl*)
MF=(L,*mfctrl*,*mfattr*)

Specifies the list form of the macro, which defines an area for a parameter list. This parameter is required for the list form. If you specify the list form, the system ignores any other parameters.

mfctrl specifies the location of the parameter list. The system generates the parameter list area at the present value of the location counter and equates *mfctrl* to this location counter value. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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

COFSDONO Macro (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede COFSDONO.
COFSDONO	
b	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=(E,*ctrl addr*)	*ctrl addr:* RX-type address or register (2) - (12).

Parameters for the execute form of COFSDONO are described in the standard form of the macro with the following exceptions:

,MF=(E,*mfctrl*)
Specifies the execute form of the COFSDONO macro. This form generates the code to store the parameters into the parameter list and execute the function of deleting a DLF object. *mfctrl* specifies the location of the parameter list.

CPOOL — Perform Cell Pool Services

The CPOOL macro creates a cell pool, obtains a cell from the pool, returns a cell to the cell pool, deletes a previously built cell pool, or places the starting and ending addresses of the cell pool extents in a buffer.

Requirements for the BUILD, GET, DELETE, and FREE requests are:

Authorization:	For the BUILD request, use LINKAGE=BRANCH only if the caller is in supervisor state and key 0. To use the TCB or KEY parameters or create a cell pool in a subpool greater than 127, the caller must be supervisor state, or key 0-7, or APF-authorized. For all GET, FREE, and DELETE requests, the caller can be problem state or supervisor state.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN not = HASN is supported.
Amode:	Any
ASC mode:	Primary Secondary, if LINKAGE=BRANCH
Serialization:	For GET, UNCOND requests, the caller must not be disabled when the specified cell pool is in a disabled reference (DREF) subpool. Otherwise, there is no requirement.
Control parameters:	Except for TCB, parameters can reside in storage above 16 megabytes if the caller is in 31-bit addressing mode.

Requirements for the LIST request are:

Authorization:	Use VERIFY only if the caller is supervisor state.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN not = HASN is supported.
Amode:	Any
ASC mode:	Primary or secondary
Serialization:	No requirement
Control parameters:	Parameters can reside in storage above 16 megabytes if the caller is in 31-bit addressing mode.

On entry to this macro, users who specify the BUILD, DELETE, LIST, or REGS=SAVE parameters must pass the address of a 72-byte save area in register 13.

The CPOOL macro is also described in *Application Development Macro Reference* with the exception of the KEY, TCB, LINKAGE, and VERIFY parameters.

The CPOOL macro is written as follows:

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

	Valid parameters (Required parameters are underlined)
BUILD	PCELLCT,SCELLCT,CSIZE,SP,LOC,CPID,KEY,TCB,HDR,LINKAGE
GET	UNCOND,COND,CPID,CELL,REGS,LINKAGE
FREE	CPID,CELL,REGS
DELETE	CPID,LINKAGE
LIST	CPID,WORKAREA,VERIFY
,UNCOND ,U ,COND ,C	**Default:** UNCOND

,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
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=ANY ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES
,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).
,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 quotes, RX-type address, or register (0), (2) - (12). **Default:** 'CPOOL CELL POOL'.
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM **Note:** Do not specify LINKAGE with FREE or LIST requests or the GET request with the COND parameter.
,REGS=SAVE ,REGS=USE	**Default:** REGS=SAVE
,WORKAREA=*(workarea,length)*	*workarea*: symbol, RX-type address, or register (0), (2) - (12). *length*: symbol or decimal number.
,VERIFY=NO ,VERIFY=YES	**Default:** VERIFY=NO.

The parameters are explained as follows:

BUILD
GET
FREE
DELETE
LIST

specifies the cell pool service to be performed.

BUILD creates a cell pool in a specified subpool by allocating storage and chaining the cells together.

GET attempts to obtain a cell from the previously built cell pool. This request can be conditional or unconditional as described under the UNCOND/COND keyword.

FREE returns a cell to the cell pool.

DELETE deletes a previously built cell pool and frees storage for the initial extent, all secondary extents, and all pool control blocks.

LIST places the beginning and ending addresses of the extents of a cell pool in a work area provided by the caller.

,UNCOND
,U
,COND
,C

when used with GET specifies whether the request for a cell is conditional or unconditional.

If you specify COND or C and no more free cells are available in the cell pool, the system returns to the caller without a cell. The system places a zero in the field specified by the CELL parameter.

If you specify UNCOND or U and no more free cells are available in the cell pool, the system obtains more storage for the cell pool. CPOOL then obtains a new cell for the caller. An unconditional CPOOL GET request fails only if enough storage is not available to extend the cell pool.

,PCELLCT = *primary cell count*

specifies the number of cells expected to be needed in the initial extent of the cell pool.

,SCELLCT = *secondary cell count*

specifies the number of cells expected to be in each secondary or non-initial 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.

,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.

,LOC = BELOW
,LOC = (BELOW,ANY)
,LOC = ANY
,LOC = (ANY,ANY)
,LOC = RES
,LOC = (RES,ANY)

specifies the location of virtual storage and central storage for the cell pool. This is helpful for users with 24-bit dependencies. The location of central storage specified in this parameter is the location of the storage after it is fixed, either by definition or by PGFIX, PGFIXA, or PGSER. When you specify the LOC parameter, the location of central storage is guaranteed only when the area is fixed.

LOC = BELOW indicates that virtual and central storage are to be allocated below 16 megabytes.

LOC = (BELOW,ANY) indicates that virtual storage is to be allocated below 16 megabytes and central storage can be anywhere.

LOC = ANY and LOC = (ANY,ANY) indicate that both virtual and central storage can be located anywhere.

LOC = RES indicates that the location of virtual and central storage depends on the location of the issuer of the macro. If the issuer resides below 16 megabytes, virtual and central storage are allocated below 16 megabytes; if the issuer resides above 16 megabytes, virtual and central storage can be located anywhere.

LOC = (RES,ANY) indicates that the location of virtual storage depends on the location of the issuer of the macro. If the issuer resides below 16 megabytes, virtual storage is allocated below 16 megabytes; if the issuer resides above 16 megabytes, virtual storage is allocated anywhere. Central storage can be located anywhere.

Note: Callers executing in 24-bit addressing mode could perform services for cell pools located in storage above 16 megabytes by specifying LOC = ANY or LOC = (ANY,ANY).

,CPID = *pool id*
specifies the address or register to contain the cell pool identifier that is returned to the caller after the pool is created using CPOOL BUILD. The issuer must specify CPID on subsequent GET, FREE, DELETE, and LIST requests.

,CELL = *cell addr*
specifies the address or register where the cell address is returned to the user by a GET or a FREE request.

,KEY = *key number*
specifies the key in which storage is to be obtained. If a register is specified, the key is taken from bits 28-31. This parameter is valid for subpools 227, 228, 229, 230, 231, and 241.

,TCB = *tcb addr*
specifies the TCB address for task related storage requests. 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 area storage and the caller does not specify TCB, the default is the TCB address in PSATOLD.

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 indicates that the linkage is through a PC instruction, LINKAGE = BRANCH indicates branch entry. For BUILD and DELETE this processing is between the caller and CPOOL processing; for GET UNCOND, the linkage is within CPOOL processing.

,REGS = SAVE
,REGS = USE
indicates whether or not registers 2-12 are to be saved for a GET or FREE request. If REGS = SAVE is specified, the registers are saved in a 72-byte user-supplied save area pointed to by register 13. If REGS = USE is specified, the registers are not saved.

,WORKAREA = *(workarea,length)*
specifies the address of a **pointer** to the work area (**not** the address of the work area) and also specifies the length of that area. The length must be at least 1024 bytes. The system places the beginning and ending addresses of the extents of 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, in order to complete a CPOOL LIST request, your program may have to issue CPOOL LIST more than once. If CPOOL LIST uses up all the space in the work area, but still has more information to return, it indicates (with a return code) that there are more address pairs. Your program can then reissue CPOOL LIST to get more information, and keep reissuing CPOOL LIST until all of the information is returned.

CPOOL LIST must be able to tell the difference between the beginning of a request (that is, the first time your program issues CPOOL LIST to get some information about a cell pool) and the continuation of a request (that is, when your program issues CPOOL LIST to get more information). Your program tells CPOOL LIST that it is beginning a new request by setting the first bit of word 0 in the work area to 1.

Until your program has obtained all the information about a cell pool that it needs from CPOOL LIST, it should not change the setting of that bit, nor should it issue a GET, FREE, or DELETE request for that cell pool. (If your program does issue a GET or FREE request before it has obtained all of the information it needs from CPOOL LIST, it must begin a new CPOOL LIST request; that is, set the first bit of word 0 to 1 and start all

over again. If your program deletes the cell pool, it can no longer issue the CPOOL LIST for that cell pool.)

CPOOL LIST uses the second through fourth words (words 1-3) in the work area to return information to your program:

- Word 1 has a return code:
  - 0 - indicates the request completed successfully.
  - 1 - indicates the system filled the work area, but has more information to give.
  - 2 - indicates that your program passed one or more invalid parameters.
  - 3 - indicates that the system found an invalid or inaccessible cell pool. In this case, the work area contains whatever starting address/ending address pairs were in it before the the system found the invalid or inaccessible cell pool.
- 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. Use VERIFY only if your program is in supervisor state.

Note: If GET U,LINKAGE=SYSTEM,REGS=USE is specified, the caller's secondary ASID will not be preserved. In all other cases the secondary ASID is unchanged.

Contents of the Registers on Return from CPOOL

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.

The contents of the registers on return from CPOOL BUILD are:

Register	Comment
0	Contains the cell pool id.
1	Used as a work register by the macro.
2-13	Unchanged
14-15	Used as work registers by the macro.

The contents of the registers on return from CPOOL GET are:

Register	Comment
0	Contains the cell pool id.
1	Address of the obtained cell for either an UNCOND request or a successful COND request. It contains a zero for unsuccessful COND requests.
2-4	Unchanged, if REGS=SAVE is specified; otherwise used as work registers by the macro.
5-12	Unchanged, if REGS=SAVE or COND REGS=USE is specified; otherwise used as work registers by the macro.
13	Unchanged
14-15	Used as work registers by the macro.

The contents of the registers on return from CPOOL FREE are:

Register	Comment
0	Contains the cell pool id.
1	Used as a work register by the macro.
2-4	Unchanged, if REGS=SAVE is specified; otherwise, used as work registers by the macro.
5-13	Unchanged
14-15	Used as work registers by the macro.

The contents of the registers on return from CPOOL DELETE are:

Register	Comment
0	Contains the cell pool id.
1	Used as a work register by the macro.
2-13	Unchanged
14-15	Used as work registers by the macro.

The contents of the registers on return from CPOOL LIST are:

Register	Comment
0 and 1	Used as work registers by the macro.
2-13	Unchanged
14 and 15	Used as work registers by the macro.

Example 1

Operation: 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

Operation: 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 3

Operation: Free a cell specifying the pool ID in register 2 and the cell address in register 3.

```
CPOOL FREE,CPID=(2),CELL=(3)
```

Example 4

Operation: Delete a cell pool, specifying the pool ID in register 2. Use a PC instruction for linkage.

```
CPOOL DELETE,CPID=(2),LINKAGE=SYSTEM
```

Example 5

Operation: 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
```

CPOOL (List Form)

The list form of the CPOOL macro builds a non-executable parameter list that can be referred to by the execute form of the CPOOL macro.

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

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

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
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=ANY ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES
,CPID=*pool id*	*pool id:* A-type address.
,KEY=*key number*	*key number:* decimal numbers 0 - 15.
,TCB=*tcb addr*	*tcb addr:* A-type address or register. **Default:** TCB address in PSATOLD.
,HDR=*hdr*	*hdr:* character string enclosed in single quotes, A-type address.
,MF=L	

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)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CPOOL.
CPOOL	
b	One or more blanks must follow CPOOL.
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
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=ANY ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES
,CPID=*pool id*	*pool id:* RX-type address or register (0), (2) - (12).
,KEY=*key number*	key number: decimal numbers 0 - 15 or register (0), (2) - (12).
,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 quotes, RX-type address, or register (0), (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM
,MF=(E,*ctrl prog*)	*ctrl prog:* RX-type address or register (0) - (12).

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

,MF=(E,*ctrl prog*)
specifies the execute form of the CPOOL macro.

CTRACE — Connect a User Application to Component Trace

The CTRACE macro connects a user application to component trace (DEFINE parameter). Once the application is connected to component trace:

- An MVS operator can use the MVS commands TRACE CT and REPLY to activate and deactivate tracing for the application. Also, the operator can use DISPLAY TRACE to obtain the status of the application. For information about how to use the TRACE CT, REPLY, and DISPLAY TRACE commands, see *System Commands*.
- The interactive problem control system (IPCS) can format and display the trace information through the CTRACE subcommand. For a description of the CTRACE subcommand, see *IPCS Command Reference*.

Before the application terminates, it should use the CTRACE macro to disconnect itself from component trace (DELETE parameter). This action prevents the system from reporting inaccurate status information on the DISPLAY TRACE command display.

Planning: Dumps and Trace Services describes how to take advantage of the services of component trace.

The caller must ensure that register 13 points to a standard 72-byte save area. Other requirements for the caller are:

Authorization:	Supervisor state and key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
Amode:	31-bit
ASC mode:	Primary
Serialization:	Enabled and unlocked

Registers 0, 1, 14, and 15 are not preserved.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CTRACE.
CTRACE	
b	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). Required on DEFINE.
,ASIDS=NO ,ASIDS=YES	**Default:** ASIDS=NO.
,BUFFER=NO ,BUFFER=YES	**Default:** BUFFER=NO.
,JOBS=NO ,JOBS=YES	**Default:** JOBS=NO.
,MINOPS=*options* ,MINOPS=NONE	*options*: RX-type address. **Default**: MINOPS=NONE.
,FMTTAB=*fmtabs*	*fmtabs*: RX-type address or register (2) - (12).

,FMTTAB=NONE	**Default**: FMTTAB=NONE.
,RC=*rc*	*rc*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).
,MF=(S)	

The parameters are explained as follows:

DEFINE
connects the application to component trace. NAME and STARTNAM are required parameters on the DEFINE request; ASIDS, BUFFER, JOBS, MINOPS, FMTTAB, RSNCODE, RC, and MF are optional parameters.

DELETE
disconnects the application from component trace. NAME is a required parameter on the DELETE request; RSNCODE, RC, and MF are optional parameters.

,NAME=*name*
specifies the external name of the application to be connected or disconnected. The name must begin with an alphabetic or national character and contain up to eight alphanumeric or national characters. (The first three letters must not be *SYS* because these are reserved for IBM use.) NAME is required for both DEFINE and DELETE.

The operator uses this name on the COMP parameter on the TRACE CT command to start and stop the tracing of the application.

,STARTNAM=*sname*
specifies the name of the application start/stop routine that the system invokes when the operator issues the TRACE CT command. The routine, or an alias, must reside in SYS1.LINKLIB or SYS1.LPALIB. STARTNAM is required on the DEFINE request.

The start/stop routine should perform any functions required to activate or deactivate tracing for the application. Such functions might include obtaining storage for the application's trace buffers or determining the events that the application will trace. For information on writing this start/stop routine, see *SPL: Application Development Guide*.

,ASIDS=YES
,ASIDS=NO
allows the operator to restrict the address spaces that the component traces by ASIDs (ASIDS=YES). On the ASID parameter on the REPLY command, in response to the TRACE CT command, the operator can specify up to 16 ASIDs that the system will trace for the application. If you specify ASIDS=NO (or use the default), the operator cannot request tracing by ASIDs.

,BUFFER=YES
,BUFFER=NO
allows the operator to specify the size of a trace buffer area (BUFFER=YES) on the TRACE CT command. If you code BUFFER=NO (or use the default), the operator cannot specify the size of the buffer area on the TRACE CT command

,JOBS=YES
,JOBS=NO
allows the operator to restrict the jobs that the application traces by job names (JOBS=YES). On the JOBS parameter on the REPLY command, in response to the TRACE CT command, the operator can specify up to 16 jobs that the system will trace for the application. If you code JOBS=NO on CTRACE (or use the default), the operator cannot specify the jobs that the application is to trace.

,MINOPS = *options*
,MINOPS = NONE
specifies a list of options that are in effect while the application is connected to component trace. These options cannot be turned off by the TRACE CT command; specify those options that you do not want an operator to be able to turn off. The character string for the options list must not exceed 255 bytes. The default is MINOPS = NONE.

,FMTTAB = *fmtabs*
,FMTTAB = NONE
specifies the name of the load module in SYS1.MIGLIB that contains the IPCS format table for the application. Use the ITTFMTB macro, described in this book, to create this format table.

The default (FMTTAB = NONE) specifies that IPCS is not to format the trace.

,RC = *rc*
identifies the location where the system is to place the return code from the CTRACE macro. The system copies the return code into the location from register 15.

,RSNCODE = *rsncode*
identifies the location where the system is to place the reason code from the CTRACE macro. The system copies the reason code into the location from register 0.

,MF = S
specifies the standard form, which places parameters into an inline parameter list and invokes the CTRACE macro service.

When control returns from CTRACE, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
0	CTRACE was successful.
4	CTRACE was unsuccessful. • For the DEFINE request, the application was already defined to component trace. • For the DELETE request, the application is not connected to component trace.
8	CTRACE was unsuccessful; look for the following reason codes in register 0: xxxx06xx - Insufficient storage for a DEFINE operation. xxxx07xx - CTRACE could not establish a recovery environment.

CTRACE (List Form)

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

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

,MF=(L,*cntl*) ,MF=(L,*cntl*,*attr*)	*cntl*: symbol. *attr*: 1- to 60-character input string. Default: 0D

The parameters are explained as follows:

,MF=(L,*cntl*)
,MF=(L,*cntl*,*attr*)

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)

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

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CTRACE.
CTRACE	
b	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). Required on DEFINE.
,ASIDS=NO ,ASIDS=YES	**Default:** ASIDS=NO.
,BUFFER=NO ,BUFFER=YES	**Default:** BUFFER=NO.
,JOBS=NO ,JOBS=YES	**Default:** JOBS=NO.
,MINOPS=*options* ,MINOPS=NONE	*options*: RX-type address. **Default**: MINOPS=NONE.
,FMTTAB=*fmtabs* ,FMTTAB=NONE	*fmtabs*: RX-type address or register (2) - (12). **Default**: FMTTAB=NONE.
,RC=*rc*	*rc*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).
,MF=(E,*cntl*) ,MF=(E,*cntl*,COMPLETE)	*cntl*: RX-type address or register (2) - (12).

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

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

DATOFF — DAT-OFF Linkage

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.

The DATOFF macro is written as follows:

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

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	DAT-OFF compare double and swap
INDMVCL0	General DAT-OFF move character long
INDMVCLK	General DAT-OFF move character long in user key
INDXC0	General DAT-OFF exclusive OR character
INDUSR1	User-written
INDUSR2	User-written
INDUSR3	User-written
INDUSR4	User-written

For each of the 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 must be 31-bit addresses.

INDCDS

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.

INDMVCL0

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

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

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 four DAT-OFF indexes that users can define. These indexes are INDUSR1, INDUSR2, INDUSR3, and INDUSR4. 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 off.
- The DAT-OFF function must have the attributes AMODE=31 and RMODE=ANY.
- 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.

Note: See *SPL: Application Development 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.

DEQ — Release a Serially Reusable Resource

DEQ removes control of one or more serially reusable resources from the active task. Register 15 is set to 0 if the request is satisfied. An unconditional request to release a resource from a task that is not in control of the resource or a request that contains invalid parameters results in abnormal termination of the task.

Note: When global resource serialization is active, it searches the SYSTEM inclusion resource name list and the SYSTEMS exclusion resource name list for every resource specified with a scope of SYSTEM or SYSTEMS. A resource whose name appears on one of these resource name lists might have its scope changed from the scope that appears on the macro. (See *Planning: Global Resource Serialization* for additional information about global resource serialization.)

The description of the entire DEQ macro follows. The DEQ macro also appears in *Application Development Macro Reference* with the exception of the RMC, GENERIC, TCB, and UCB parameters. These parameters are restricted in use to programs that run in supervisor state, key 0-7, or with APF authorization, and are, therefore, described only here.

Except for TCB and UCB, all input parameters to this macro can reside in storage above 16 megabytes for callers executing in 31-bit addressing mode.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede DEQ.
DEQ	
b	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*	*rname length:* symbol, decimal digit, or register (2) - (12). **Note:** *rname length* must coded if a register is specified for *rname addr*.
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
)	
,RET=HAVE ,RET=NONE	
,RMC=NONE ,RMC=STEP	**Default:** RMC=NONE
,GENERIC=NO ,GENERIC=YES	**Default:** GENERIC=NO **Note:** If GENERIC=YES is specified, you must also specify RET=HAVE above.
,TCB=*tcb addr*	*tcb addr:* A-type address, or register (2) - (12). **Note:** TCB cannot be specified with RMC above.
,UCB=*ucb addr*	*ucb addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows.

(
specifies the beginning of the resource description.

qname addr
specifies the address in virtual storage of an 8-character name. The *qname* must be the same name specified for the resource in an ENQ macro.

,*rname addr*
specifies the address in virtual storage of the name used in conjunction with qname and scope to represent the resource acquired by a previous ENQ macro. The name can be qualified and must be from 1 to 255 bytes long. The rname must be the same name specified for the resource in an ENQ macro.

,
,*rname length*
specifies the length of the *rname* described above. The length must have the same value as specified in the previous ENQ macro. If this parameter is omitted, the assembled length of the *rname* is used. You can specify a value between 1 and 255 to override the assembled length, or you may specify a value of 0. If 0 is specified, the length of the *rname* must be contained in the first byte at the *rname addr* specified above.

,
,STEP
,SYSTEM
,SYSTEMS
specifies the scope of the resource. You must specify the same STEP, SYSTEM, or SYSTEMS option as you used in the ENQ macro requesting the resource.

)
specifies the end of the resource description.

Note: Multiple resources can be specified with the DEQ macro. You can repeat *qname addr*, *rname addr*, *rname length*, and the scope until there is a maximum of 255 characters including the parentheses.

,RET = HAVE
,RET = NONE
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 ENQ was executed with ECB. 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 is abnormally terminated 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
,GENERIC = NO
,GENERIC = YES
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 terminate the must complete function (STEP). The NONE or STEP subparameter must agree with the subparameter specified in the SMC parameter of the corresponding ENQ macro.

GENERIC specifies whether or not (YES or NO) all resources with the specified *qname* are to be released. In order for the resource to be released, the task must have control of or be in ECB wait for the resource. (ECB was specified on the original ENQ.) If the task is waiting for a resource, but is not in an ECB wait, the task remains queued and waiting.

The following return codes are associated with a GENERIC DEQ:

Hexadecimal Code	Meaning
0	One or more resources which the task had control of or was in ECB wait for have been released.
4	One or more resources were unconditionally requested by the task, but the task was not assigned control. The task is not removed from the wait condition. However, other resources with the same *qname* might have been released.
8	No resources were found for the specified *qname*.

,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) is abnormally terminated if the RET parameter is omitted and an attempt is made to DEQ 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 TCB DEQ request releases the resources with a zero return code.

Note: The TCB resides in storage below 16 megabytes.

,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. The UCB parameter is optional.

Note: The UCB resides in storage below 16 megabytes.

,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.

Return codes are provided by the control program 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 8.

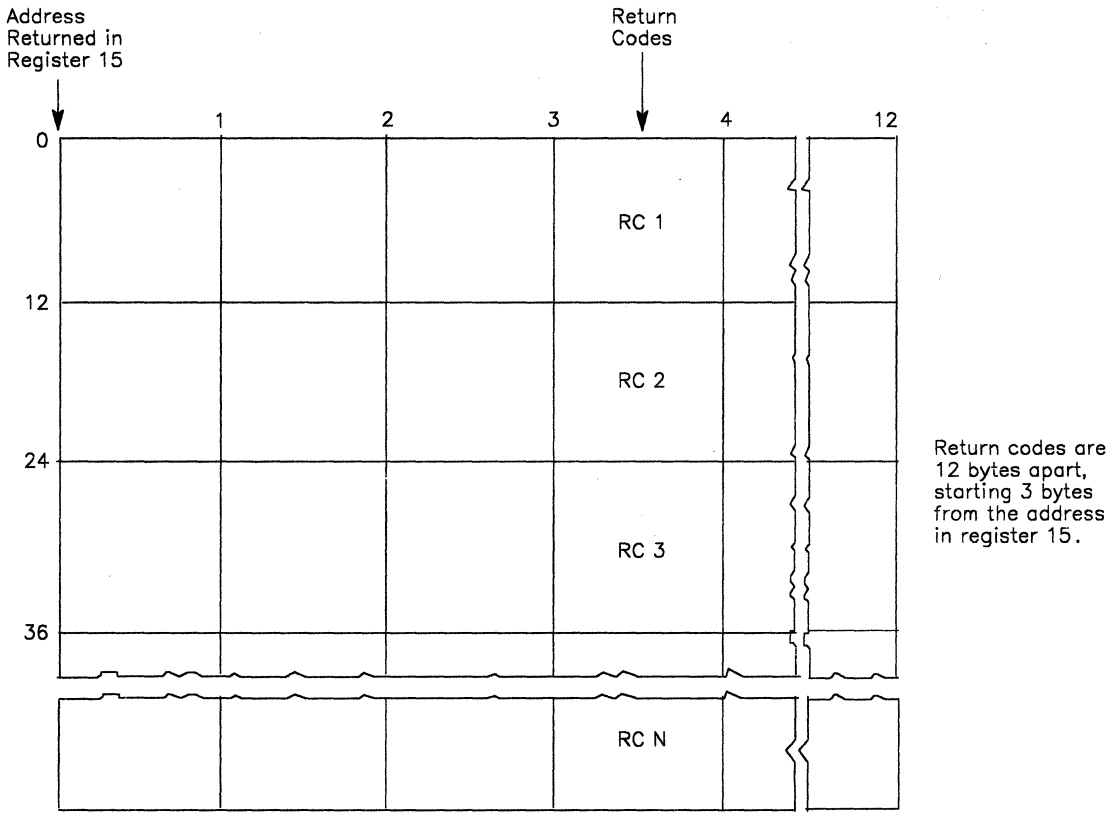


Figure 8. Return Code Area Used by DEQ

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 are shown below.

Hexadecimal Code	Meaning
0	The resource has been released.
4	The resource has been requested for the task, but the task has not been assigned control. The task is not removed from the wait condition. (This return code could result if DEQ is issued within an exit routine which was given control because of an interruption.)
8	Control of the resource has not been requested by the active task, or the resource has already been released.

Example 1
Operation: Unconditionally release control of the resource in Example 1 of ENQ, and reset the "must-complete" state.

```
DEQ (MAJOR1,MINOR1,8,STEP),RMC=STEP
```

Example 2
Operation: Conditionally release control of the resource in Example 2 of ENQ.

```
DEQ (MAJOR2,MINOR2,4,SYSTEM),TCB=(R2),RET=HAVE
```

Example 3
Operation: Unconditionally release control of the resource (device) in Example 1 of RESERVE.

```
DEQ (MAJOR3,MINOR3,,SYSTEMS),UCB=(R3)
```

Example 4
Operation: 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:

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

(	
qname addr	*qname addr:* A-type address.
, ,*rname addr*	*rname addr:* A-type address.
, ,*rname length*	*rname length:* symbol or decimal digit.
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
)	
,RET=HAVE ,RET=NONE	**Default:** RET=NONE
,RMC=NONE ,RMC=STEP	**Default:** RMC=NONE
,GENERIC=NO ,GENERIC=YES	**Default:** GENERIC=NO **Note:** If GENERIC=YES is specified, you must also specify RET=HAVE above.
,TCB=0	**Note:** TCB cannot be specified with RMC above, and must be specified on the list form if used on the execute form.
,UCB=*ucb addr*	*ucb addr:* A-type address.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the DEQ macro.

DEQ (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the DEQ macro. The parameter list can be generated by the list form of either the DEQ or the ENQ macro.

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

name	*name:* symbol. Begin *name* in column 1.
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*	*rname addr:* RX-type address, or register (2) - (12).
, ,*rname length*	*rname length:* symbol, decimal digits, or register (2) - (12).
, ,STEP ,SYSTEM ,SYSTEMS	
)	**Note:** *See note opposite (above.*
,RET=HAVE ,RET=NONE	
,RMC=NONE ,RMC=STEP ,GENERIC=NO ,GENERIC=YES	**Note:** If GENERIC=YES is specified, you must also specify RET=HAVE above.
,TCB=*tcb addr*	*tcb addr:* RX-type address, or register (2) - (12). **Note:** TCB cannot be specified with RMC above, 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).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) - (12).

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

,MF=(E,*ctrl addr*)
specifies the execute form of the DEQ macro using a remote control program parameter list.

DOM — Delete Operator Message

The DOM macro is used to delete an operator message or group of messages from the display screen of the operator's console. It can also prevent messages from ever appearing on any operator's console. When a program no longer requires that a message be displayed, it can issue the DOM macro to delete the message.

Depending on the timing of the DOM relative to the WTO(R), the message may or may not be displayed. If the message is being displayed, it is removed when space is required for other messages. If the message is not yet displayed, it is removed before it gets displayed.

When a WTO or WTOR macro is issued, the system assigns an identification number to the message and returns this number (32 bits right-justified) to the issuing program in register 1. 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 general register 1.

The DOM macro is written as follows:

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

MSG=*addr*	*addr:* register (1) - (12). or an address.
MSGLIST=*list addr*	*list addr:* symbol, RX-type address, or register (1) - (12).
TOKEN=*addr*	*addr.* register (1) - (12), or an address.
,COUNT=*addr*	*addr:* register (2) - (12), or an address.
,SYSID=*addr*	*addr:* register (2) - (12), or an address.
,SCOPE=SYSTEM ,SCOPE=SYSTEMS	

The parameters are explained as follows:

MSG=*addr*
The field or register that contains the 32-bit identification number of a message to be deleted.

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.

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(R) via the TOKEN parameter. WTO(R) returns control to the application with a message id in register 1. To delete the message by the TOKEN method, ignore the message id returned by WTO(R) 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.

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=
specifies a field or register containing the one-byte count of 4-byte message ids associated with this request. The count must be from 1 to 60. If COUNT is specified, the issuer must not set the high order bit on in the last entry of the DOM parameter list. If COUNT is not specified, the message ids are treated as 3-byte ids. If an address is used, the address points to a 1-byte field that contains the count. COUNT is invalid with SYSID and TOKEN

SYSID= *addr*
specifies a field or register containing the 1-byte id of the system on which the message was issued. If no message ids are specified, (that is, MSG or MSGLIST is not specified) all messages issued from the specified system are deleted. If message ids are specified, (that is, MSG or MSGLIST has been specified), messages indicated by the MSG or MSGLIST parameter issued from the specified system are deleted.

SYSID is invalid with COUNT. SYSID can be used with the TOKEN parameter to delete all messages originally issued from a particular system with the specified TOKEN. Authorized users may delete any messages originally issued under the same ASID when TOKEN and SYSID are specified. Unauthorized users may delete only those messages that were originally issued under the same jobstep TCB and ASID when TOKEN and SYSID are specified. If an address is used, the address points to a 1-byte field which contains the system id.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

SCOPE=SYSTEM
SCOPE=SYSTEMS
specifies how to process the DOM request. If SCOPE=SYSTEMS is specified, the DOM request is to be communicated to other processors. If SCOPE=SYSTEM is specified, the DOM request is not to be communicated to other processors. If SCOPE is not specified, the DOM request defaults to SCOPE=SYSTEMS.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Notes:

1. 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.
2. Any authorized DOM parameters that are specified by an unauthorized program will cause a 157 ABEND.

Example 1

Operation: Delete an operator message. The message id is in register 1.

```
DOM MSG=(1)
```

Example 2

Operation: Delete a list of operator messages.

```
DOM MSGLIST=ID2
```

Example 3

Operation: Delete four operator messages. The number of messages to be deleted is stored in the field named FOUR, and ID3 is the address of the list of message ids for the four messages.

```
DOM MSGLIST=ID3,COUNT=FOUR
```

Example 4

Operation: Delete a single message issued on a particular system. The message ID is in register 1, and the one-byte system id is stored in the field named TWO.

```
DOM MSG=(1),SYSID=TWO
```

Example 5

Operation: Delete all messages issued on a particular system. The one-byte system id is stored in the field named SYSNAME.

```
DOM SYSID=SYSNAME
```

Example 6

Operation: Delete all messages issued with a particular token on a particular system. The four-byte token is stored in TOKEN1, and the one-byte system id is in TWO.

```
DOM TOKEN=TOKEN1,SYSID=TWO
```

DSGNL — Issue Direct Signal

The DSGNL macro uses the signal processor (SIGP) to modify or sense the physical state of a specific processor in a multiprocessing configuration. The SIGP instruction order codes specified on the DSGNL macro are defined as direct services. Additional SIGP order codes defined as remote services are available through the RISGNL and RPSGNL macros. See *Principles of Operation* for an explanation of the order codes.

Programs executing in cross memory mode can issue this macro.

The DSGNL macro is written as follows:

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

SENSE START STOP RESTART SSS ICPUR CPUR STATUS PREFIX (0)	
,CPU=*PCCA addr*	*PCCA addr:* RX-type address, or register (1).
,PARAM=*addr* ,PARAM=(2)	*addr:* RX-type address, or register (2). **Note:** This parameter is required with PREFIX and STATUS only. It cannot be specified with any of the other parameters.

The parameters are explained as follows:

SENSE
START
STOP
RESTART
SSS
ICPUR
CPUR
PREFIX
STATUS
(0) specifies the action to be performed. If (0) is specified, the code indicating the desired function has already been loaded into bits 24-31 of register 0. (Only the direct class functions are valid.) The actions and codes are:

	Order Code	Action
SENSE	01	State of specified processor is to be sensed
START	04	Start function
STOP	05	Stop function
RESTART	06	Restart function
SSS	09	Stop and store status function
ICPUR	0B	Initial processor reset function
CPUR	0C	Processor reset function
PREFIX	0D	Set prefix from address
STATUS	0E	Store status at address

,CPU = *PCCA addr*
specifies the address of the physical configuration communication area (PCCA) of the processor on which the function is to be executed.

Note: The PCCA resides in storage below 16 megabytes.

,PARAM = *addr*
,PARAM = (2)
allows an address to be passed to the specified processor. If *addr* is coded, the word at that location is loaded into register 2 and passed to the specified processor. The contents of that location must contain a real address. If (2) is coded, the contents of register 2 is passed to the processor. Register 2 must also contain a real address.

When this parameter is used with PREFIX, the word passed to the specified processor is the address to which the processor's prefix register is to be set.

When this parameter is used with STATUS, the word passed to the specified processor is the real address at which the processor's status is to be stored.

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

Hexadecimal Code	Meaning
00	Function successfully initiated, but not necessarily completed.
04	Function not completed because the access path to the addressed processor was busy or the addressed processor was in a state where it could not accept and respond to the order code.
08	Function unsuccessfully initiated or successful SIGP SENSE request. Status is returned in register 0.
0C	Specified processor is either not installed, not configured into the system, or powered off.
14	MSSF is currently inoperative.

With a return code of 8, register 0 contains status information from the SIGP macro. The bit settings and meanings follow:

Bits	Meaning
0	Equipment check
1-21	Unassigned, contains zeros
22	Incorrect state
23	Invalid parameter
24	External call pending
25	Stopped
26	Operator intervening
27	Check stop
28	Not ready
29	MSSF currently inoperative
30	Invalid order code
31	Receiver check

Example

Operation: The processor whose PCCA address is in register 1 will be placed in the STOP state.

```
DSGNL STOP,CPU=(1)
```

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 "DSPSERV — Create, Delete, and Control Hiperspaces" on page 169.

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 ESA/370 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 *SPL: Application Development — Extended Addressability*.

Use the DSPSERV macro to:

- Create a data space (CREATE 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)

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.

Requirements for callers of DSPSERV are as follows:

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
- Create and delete a SCOPE=ALL and SCOPE=COMMON data space
- Assign a storage key to a data space
- Assign data space ownership to a TCB
- 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

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

Amode: 31-bit addressing

ASC mode: Primary or access register (AR)

Serialization: Enabled, unless you specify DSPSERV RELEASE with DISABLED=YES, and unlocked

Control parameters: Control parameters must be in the primary address space.

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 if the return code in GPR 15 is not 0; otherwise, used as a work register by the macro
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 a work register by the macro
2 - 13	Unchanged
14 - 15	Used as a work register by the macro

DSPSERV is also described in *Application Development Macro Reference*, with the exception of the LOAD and OUT requests and the DREF, SCOPE, KEY, CALLERKEY, TTOKEN, and DISABLED parameters. These parameters are restricted to supervisor state or PSW key 0-7 programs.

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

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

	Valid parameters (Required parameters are underlined.)
CREATE	STOKEN, NAME, TYPE, GENNAME, OUTNAME, BLOCKS, DREF, SCOPE, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, NUMBLKS
RELEASE	STOKEN, START, BLOCKS, DISABLED
DELETE	STOKEN, TTOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
LOAD	STOKEN, START, BLOCKS
OUT	STOKEN, START, BLOCKS
,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 ,GENNAME=COND ,GENNAME=YES	**Default:** GENNAME=NO
,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)* ,BLOCKS=*(max,init)* ,BLOCKS=*max* ,BLOCKS=(0,*init*) ,BLOCKS=0 ,BLOCKS=(0,*init-addr*) ,BLOCKS=(*size-addr*) ,BLOCKS=(*size*)	*max-addr:* RX-type address or register (2) - (12). *init-addr:* RX-type address or register (2) - (12). *max:* Number up to 524288. *init:* Number up to 524288. 0 specifies the installation default size. **Default for CREATE**: BLOCKS=0 *size-addr:* RX-type address or register (2) - (12). *size:* Number up to 524288.
,DREF=NO ,DREF=YES	**Default:** DREF=NO
,SCOPE=SINGLE ,SCOPE=ALL ,SCOPE=COMMON	**Default:** SCOPE=SINGLE
,CALLERKEY ,KEY=*key-addr*	**Default**: CALLERKEY *key-addr:* RX-type address or register (2) - (12).
,FPROT=YES ,FPROT=NO	**Default:** FPROT=YES
,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).
,VAR=NO ,VAR=YES	**Default:** VAR=NO
,DISABLED=NO ,DISABLED=YES	**Default:** DISABLED=NO
,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.

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, 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).

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 central, expanded, 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 and other data spaces, 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, the TCB that represents it must own the data space. Otherwise, the TCB that represents the caller must be in the home or primary address of the owner of the data 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.

,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, where the first three characters are any IBM component prefix.
- Names that begin with SYSAxxxx through SYSIxxxx, where the fourth through sixth characters are any IBM component prefix.
- 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 NAMES 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.

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.

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 and key 0-7 programs can create and delete SCOPE=ALL and SCOPE=COMMON data spaces.

If an address space contains a task that owns a SCOPE=ALL or SCOPE=COMMON data space, the address space should be non-swappable.

SCOPE is an optional parameter for DSPSERV CREATE; the default is SCOPE=SINGLE.

,CALLERKEY
,KEY=*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.

,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.

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 SMF 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.

,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.

,MF=S
Specifies the standard form of DSPSERV. The standard form places the parameters into an in-line parameter list.

Return and reason codes from DSPSERV CREATE:

Return code	Reason code	Meaning
00		DSPSERV CREATE completed successfully.
04	xx000Cxx	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.
08	xx0005xx	Creation of data space would violate installation criteria. See *System Modifications*.
08	xx0009xx	Specified data space name is not unique within the address space.
08	xx0012xx	The system's set of generated names for data spaces and hiperspaces has been temporarily exhausted.
0C	xx0006xx	The system cannot create any additional data spaces at this time because of a shortage of resources.
0C	xx0007xx	The system cannot obtain addressability to its data structures.
1D	xx0012xx	An unauthorized caller tried to create a shared standard hiperspace.
1D	xx0014xx	An attempt was made to create either a SCOPE=ALL or SCOPE=COMMON data space, or a shared scroll hiperspace while the owning address space was swappable.

Return and reason codes from DSPSERV EXTEND:

Return code	Reason code	Meaning
00		DSPSERV EXTEND completed successfully.
08	xx0502xx	Extending the data space would cause the data space and hiperspace limits for the address space to be exceeded.
08	xx0503xx	You are using VAR=YES to extend the current size of the data space; however, the data space is already the maximum size.

Example of Creating a Data Space

Create a data space named TEMP with a size of 10 million bytes.

```
         DSPSERV CREATE,NAME=DSPCNAME,STOKEN=DSPCSTKN,              X
               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
```

DSPSERV (List Form)

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

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

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

MF=(L,*list addr*) MF=(L,*list addr,attr*)	*list addr:* symbol. *attr*: 1- to 60-character input string. **Default:** 0D
,PLISTVER=0 ,PLISTVER=1	**Default:** PLISTVER=0

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.

,PLISTVER=0
,PLISTVER=1
Specifies the macro version associated with DSPSERV.

PLISTVER is an optional parameter that determines which parameter list the system generates. *init-addr* on the BLOCKS parameter is associated with macro version 1 that produces a 60-character parameter list; all other parameters are associated with the macro version 0 that produces a 56-character parameter list. Therefore, if you use the BLOCKS=(*max-addr,init-addr*) parameter on subsequent execute forms of DSPSERV, you *must* specify PLISTVER=1 on the list form. PLISTVER=0 is the default.

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.

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

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

	Valid parameters (Required parameters are underlined.)
CREATE	STOKEN, NAME, TYPE, GENNAME, OUTNAME, BLOCKS, DREF, SCOPE, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, NUMBLKS
RELEASE	STOKEN, START, BLOCKS, DISABLED
DELETE	STOKEN, TTOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
LOAD	STOKEN, START, BLOCKS
OUT	STOKEN, START, BLOCKS
,STOKEN=*stoken-addr*	*stoken-addr:* RX-type address or register (2) - (12).
,TYPE=BASIC	**Default:** TYPE=BASIC
,NAME=*name-addr*	*name-addr:* RX-type address or register (2) - (12).
,GENNAME=NO	**Default:** GENNAME=NO
,GENNAME=COND	
,GENNAME=YES	
,OUTNAME=*outname-addr*	*outname-addr:* RX-type address or register (2) - (12).
,START=*start-addr*	*start-addr:* RX-type address or register (2) - (12).
,BLOCKS=*(max-addr,init-addr)*	*max-addr:* RX-type address or register (2) - (12).
,BLOCKS=*(max,init)*	*init-addr:* RX-type address or register (2) - (12).
,BLOCKS=*max*	*max:* Number up to 524288.
,BLOCKS=(0,*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	
,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).

,VAR=NO ,VAR=YES	**Default:** VAR=NO
,DISABLED=NO ,DISABLED=YES	**Default:** DISABLED=NO
,MF=(E,*list addr*) ,MF=(E,*list addr*,COMPLETE)	

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

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the DSPSERV macro. *list addr* defines the area that the system uses for the parameter list.

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

DSPSERV — Create, Delete, and Control Hiperspaces

DSPSERV for dataspaces

To control the use of dataspaces, use the variation of the DSPSERV macro described under "DSPSERV — Create, Delete, and Control Data Spaces" on page 155.

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 expanded storage and auxiliary storage, if necessary. The HSTYPE=SCROLL parameter creates a standard hiperspace. The **ESO hiperspace** is backed only with expanded storage. HSTYPE=CACHE creates an ESO hiperspace. For more information on hiperspaces and how to use them, see *SPL: Application Development — Extended Addressability*. To learn the restrictions for the use of hiperspaces, see the description of the HSPSERV macro later in this book.

Use the DSPSERV macro to:

- Create a hiperspace (CREATE parameter)
- Delete a hiperspace (DELETE parameter)
- Release an area of a hiperspace (RELEASE parameter)
- Increase the current size of a hiperspace (EXTEND parameter)

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.

Requirements for callers of DSPSERV are as follows:

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

Cross memory mode: Any

Amode: 31-bit addressing

ASC mode: Primary or access register (AR)

Serialization: Enabled, unless you specify DSPSERV RELEASE with DISABLED=YES, and unlocked

Control parameters: Control parameters must be in the primary address space.

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 if the return code in GPR 15 is not 0; otherwise, used as a work register by the macro
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

DSPSERV is also described in *Application Development Macro Reference*, 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.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede DSPSERV.
DSPSERV	
b	One or more blanks must follow DSPSERV.
	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, 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 ,HSTYPE=CACHE	**Default:** HSTYPE=SCROLL
,SHARE=NO ,SHARE=YES	**Default:** SHARE=NO
,CASTOUT=YES ,CASTOUT=NO	**Default:** CASTOUT=YES
,NAME=*name-addr*	*name-addr:* RX-type address or register (2) - (12).
,GENNAME=NO ,GENNAME=COND ,GENNAME=YES	**Default:** GENNAME=NO
,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)* ,BLOCKS=*(max,init)* ,BLOCKS=*max* ,BLOCKS=(0,*init*) ,BLOCKS=0 ,BLOCKS=(0,*init-addr*) ,BLOCKS=(*size-addr*) ,BLOCKS=(*size*)	*max-addr:* RX-type address or register (2) - (12). *init-addr:* RX-type address or register (2) - (12). *max*: Number up to 524288. *init*: Number up to 524288. 0 specifies the installation default size. **Default for CREATE:** BLOCKS=0 *size-addr:* RX-type address or register (2) - (12). *size:* Number up to 524288.
,CALLERKEY ,KEY=*key-addr*	**Default**: CALLERKEY *key-addr:* RX-type address or register (2) - (12).
,FPROT=YES ,FPROT=NO	**Default:** FPROT=YES
,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).
,VAR=NO ,VAR=YES	**Default:** VAR=NO
,DISABLED=NO ,DISABLED=YES	**Default:** DISABLED=NO
,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.

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).

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 central, expanded, or auxiliary storage.

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.

,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 non-shared standard hiperspace (SHARE=NO) or a shared standard hiperspace (SHARE=YES). Generally, a program can share a **non-shared 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 non-shared 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. When the system needs the expanded storage for its own needs, it is less likely to take the expanded 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 expanded 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 expanded storage when a shortage arises.

Note: Specifying CASTOUT=NO places a heavy demand on expanded 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 non-swappable.)

CASTOUT=YES is the default.

,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, where the first three characters are any IBM component prefix.
- Names that begin with SYSAxxxx through SYSIxxxx, where the fourth through sixth characters are any IBM component prefix.
- 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 *'nCCCCXYZ'* where *'n'* is the numeral, *'CCCC'* is the 4-character string generated by the system, and XYZ comes from the name you supplied on NAME. See NAME for more information about naming conventions.

GENNAME=NO	The system does not generate a name. You *must* supply a name unique within the address space. GENNAME=NO is the default.
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.

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 SMF user 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 in 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.

,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.

,MF = S

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

Return and reason codes from DSPSERV CREATE:

Return code	Reason code	Meaning
00		DSPSERV CREATE completed successfully.
04	xx000Cxx	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.
08	xx0005xx	Creation of hiperspace would violate installation criteria. See *System Modifications*.
08	xx0009xx	The specified hiperspace name is not unique within the address space.
08	xx0010xx	ESO hiperspace creation rejected because there is no expanded storage on the system.
08	xx0012xx	The system's set of generated names for data spaces and hiperspaces has been temporarily exhausted.
0C	xx0006xx	The system cannot create any additional hiperspaces at this time because of a shortage of resources.
0C	xx0007xx	The system cannot obtain addressability to its own hiperspaces.

Return and reason codes from DSPSERV EXTEND:

Return code	Reason code	Meaning
00		DSPSERV EXTEND completed successfully.
08	xx0502xx	Extending the hiperspace size would cause the data space and hiperspace limits for the address space to be exceeded.
08	xx0503xx	You are using VAR = YES to extend the current size of the hiperspace; however, the hiperspace is already the maximum size.

Example of Creating a Hiperspace

Create a hiperspace named TEMP with a size of 10 million bytes.

```
         DSPSERV CREATE,NAME=HSPCNAME,STOKEN=HSPCSTKN,             X
              TYPE=HIPERSPACE,BLOCKS=HSPBLCKS,ORIGIN=HSPCORG
*        .
HSPCNAME DC   CL8'TEMP    '           HIPERSPACE NAME
HSPCSTKN DS   CL8                     HIPERSPACE STOKEN
HSPCORG  DS   F                       HIPERSPACE ORIGIN RETURNED
HSPCSIZE DC   F'10000000'
HSPBLCKS DC   A((HSPCSIZE+4095)/4096)  NUMBER OF BLOCKS NEEDED FOR
*                                     A 10 MILLION BYTE HIPERSPACE
```

DSPSERV (List Form)

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

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

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

MF=(L,*list addr*) MF=(L,*list addr,attr*)	*list addr:* symbol. *attr*: 1- to 60-character input string. **Default:** 0D
,PLISTVER=0 ,PLISTVER=1	**Default:** PLISTVER=0

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.

,PLISTVER=1
,PLISTVER=0
Specifies the macro version associated with DSPSERV.

PLISTVER is an optional parameter that determines which parameter list the system generates. Only *init-addr* on the BLOCKS parameter is associated with macro version 1 that produces a 60-character parameter list; all other parameters are associated with the macro version 0 that produces a 56-character parameter list. Therefore, if you use the BLOCKS=(*max-addr,init-addr*) parameter on subsequent execute forms of DSPSERV, you *must* specify PLISTVER=1 on the list form. PLISTVER=0 is the default.

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.

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

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

	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, 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 ,HSTYPE=CACHE	**Default:** HSTYPE=SCROLL
,SHARE=NO ,SHARE=YES	**Default:** SHARE=NO
,CASTOUT=YES ,CASTOUT=NO	**Default:** CASTOUT=YES
,NAME=*name-addr*	*name-addr:* RX-type address or register (2) - (12).
,GENNAME=NO ,GENNAME=COND ,GENNAME=YES	**Default:** GENNAME=NO
,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* ,CALLERKEY	*key-addr:* RX-type address or register (2) - (12). **Default**: CALLERKEY
,FPROT=YES ,FPROT=NO	**Default:** FPROT=YES
,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).

,VAR=NO
,VAR=YES

Default: VAR=NO

,DISABLED=NO
,DISABLED=YES

Default: DISABLED=NO

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

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

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the DSPSERV macro. *list addr* defines the area that the system uses for the parameter list.

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

DYNALLOC — Dynamic Allocation

See *SPL: Application Development Guide* for the description of this macro.

ENQ — Request Control of a Serially Reusable Resource

ENQ assigns control of one or more serially reusable resources to a task. If any of the resources are not available, the task might be placed in a wait condition until all of the requested resources are available. Once control of a resource has been assigned to a task, it remains with that task until one of the programs running under that task issues a DEQ macro to release the resource or the task terminates.

ENQ identifies the resource by a pair of names, the *qname* and the *rname*, and a scope value. The scope value determines what other tasks, address spaces, or systems can use the resource. All programs that share the resource must use the qname, rname, and scope value consistently. You can request either shared or exclusive use of a resource.

Use ENQ with RET=TEST to determine the status of the resource. Return codes tell whether the resource is immediately available or in use, and whether control has been previously requested by the active task in another ENQ macro.

ENQ with the MASID and MTCB parameters allows a further conditional control of a resource. One task, called the "issuing task" can issue an ENQ macro for a resource specifying the ASID and TCB of another task, called the "matching task". MTCB and MASID parameters are specified with RET=HAVE, RET=TEST, and/or ECB to provide additional return codes. If the issuing task does not receive control of the resource, it may receive a return code indicating that the resource is controlled by the matching task. Upon receiving this return code, the issuing task could use the resource, if serialization between itself and the matching task has been pre-arranged through a protocol.

Issuing two ENQ macros for the same resource without an intervening DEQ macro causes the task to abend, unless the second ENQ designates RET=TEST, USE, CHNG, or HAVE. If the task terminates, either normally or abnormally, while the task still has control of any serially reusable resources, all requests made by this task will automatically have DEQ processing performed for them. If resource input addresses are incorrect, the task abnormally terminates.

Global resource serialization counts and limits the number of concurrent resource requests from an address space. If an unconditional ENQ (an ENQ that uses the RET=NONE option) causes the count of concurrent resource requests to exceed the limit, the caller abends with a system code of X'538'. For more information, see the section on limiting concurrent requests for resources in *Application Development Guide*.

The description of the ENQ macro follows. The ENQ macro is also described in *Application Development Macro Reference* with the exception of the SMC, ECB, and TCB parameters. These parameters are restricted in use to programs that run in supervisor state, PSW key 0-7, or APF authorized and are therefore only described here.

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

(	
qname addr	*qname addr:* A-type address, or register (2) - (12).
,*rname addr*	*rname addr:* A-type address, or register (2) - (12).
, ,E ,S	**Default:** E
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12). **Default:** assembled length of *rname* **Note:** *rname length* must be coded if a register is specified for *rname addr*.
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
)	
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	**Default:** RET=NONE
,SMC=NONE ,SMC=STEP ,ECB=*ecb addr* ,TCB=*tcb addr*	*ecb addr:* A-type address, or register (2) - (12). *tcb addr:* A-type address, or register (2) - (12). **Default:** SMC=NONE **Note:** ECB cannot be specified with RET above. ECB and TCB can be specified together. If TCB is specified but not ECB, then RET=CHNG, TEST or USE must be specified above.
,MASID=*matching-asid addr* ,MTCB=*matching-tcb addr*	*matching-asid addr:* A-type address, or register (2) - (12). *matching-tcb addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

(
: specifies the beginning of the resource description.

qname addr
: specifies the address in virtual storage of an 8-character name. Every program issuing a request for a serially reusable resource must use the same *qname, rname*, and scope to represent the resource.

,*rname addr*

specifies the address in virtual storage of the name used in conjunction with *qname* to represent a single resource. The name can be qualified and must be from 1 to 255 bytes long and contain any hexadecimal character.

,
,E
,S

specifies whether the request is for exclusive (E) or shared (S) control of the resource. If the resource is modified while under control of the task, the request must be for exclusive control; if the resource is not modified, the request should be for shared control.

,*rname length*

specifies the length of the *rname* described above. If this parameter is omitted, the assembled length of the *rname* is used. You can specify a value between 1 and 255 to override the assembled length, or you may specify a value of 0. If you specified 0, the length of the *rname* must be contained in the first byte at the *rname addr* specified above.

,
,STEP
,SYSTEM
,SYSTEMS

specifies the scope of the resource.

STEP specifies that the resource can be used only within an address space. If STEP is specified, a request for the same *qname* and *rname* from a program in another address space denotes a different resource.

SYSTEM specifies that the resource can be used by more than one address space.

SYSTEMS specifies that the resource can be shared between systems.

STEP, SYSTEM, and SYSTEMS are mutually exclusive and do not refer to the same resource. If two macros specify the same *qname* and *rname*, but one specifies STEP and the other specifies SYSTEM or SYSTEMS, they are treated as requests for different resources.

When global resource serialization is active, scope conversion can occur. This could result in two requests with different scopes referring to the same resource. See *Planning: Global Resource Serialization* for details.

)

specifies the end of the resource description.

Note: Multiple resources can be specified in the ENQ macro. You can repeat the *qname addr*, *rname addr*, type of control, *rname length*, and scope until there is a maximum of 255 characters including the parentheses.

,RET=CHNG
,RET=HAVE
,RET=TEST
,RET=USE
,RET=NONE

specifies the type of request for all of the resources named above.

CHNG - the status of the resource specified is changed from shared to exclusive control.

HAVE - control of the resources is requested conditionally; that is, control is requested only if a request has not been made previously for the same task.

TEST - the availability of the resources is to be tested, but control of the resources is not requested.

USE - control of the resources is to be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition.

NONE - control of all the resources is unconditionally requested.

See "Return Codes" on page 185 for an explanation of the return codes for these requests.

,SMC = NONE
,SMC = STEP
,ECB = *ecb addr*
,TCB = *tcb addr*

specifies optional parameters available to the system programmer:

SMC specifies that the set must-complete function is not to be used (NONE) or that it is to place other tasks for the step nondispatchable until the requesting task has completed its operations on the resource (STEP).

When SMC = STEP is specified with RET = HAVE and the requesting task already has control of the resource, the SMC function is turned on and the task continues to control the resource.

SMC = and TCB = are mutually exclusive with the MASID parameter, therefore, hexadecimal return codes 20, 24, 28, and 44 will not be given by an ENQ using the SMC or TCB operands.

The return codes and status of the set must-complete function for the various RET = specifications are as follows:

	Hexadecimal Code	SMC Status
RET = CHNG	0	on
	4	off
	8	off
	14	off
RET = HAVE	0	on
	8	on
	14	off
RET = TEST	0	off
	4	off
	8	off
	14	off
RET = USE	0	on
	4	off
	8	off
	14	off
	18	off

ECB specifies the address of an ECB, and conditionally requests all of the resources named in the macro. If the return code for one or more requested resources is hexadecimal 4 or 24 and the request is not nullified by a corresponding DEQ, the ECB is posted when all the requested resources (specifically, those that initially received a return code of 4 or 24) are assigned to the requesting task.

If the ECB parameter is an A-type address, the address is the name of the fullword that is used as an ECB. If the operand is a register, then the register contains the address of the ECB.

TCB 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 ENQ is to be done.

Note: The TCB resides in storage below 16 megabytes.

,MASID = *matching-asid addr*

specifies the matching task (by defining a matching ASID) for the ENQ, if used in conjunction with the MTCB parameter. MASID defines the ASID of a task that may be using a resource desired by the issuer of the ENQ macro. If the MASID parameter is an A-type address, the address is the name of a fullword containing the ASID. If the operand is a register, then the register contains the ASID.

Note: MASID can only be specified if MTCB is also specified.

,MTCB = *matching-tcb addr*

specifies the matching task (by defining a matching TCB) for the ENQ, if used in conjunction with the MASID parameter. MTCB defines the TCB of a task that may be using a resource desired by the issuer of the ENQ macro.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the issuer of the ENQ and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

The MASID and MTCB parameters are specified with RET=HAVE, RET=TEST, and/or ECB= parameters to elicit additional return codes that provide information about the owner of the resource. If the MTCB parameter is an A-type address, the address is the name of a fullword containing the TCB. If the operand is a register, then the register contains the TCB.

Note: MTCB can only be specified if MASID is also specified.

,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 Codes

Return codes are provided by the control program only if you specify RET=TEST, RET=USE, RET=CHNG, RET=HAVE, or ECB=; otherwise return of the task to the active condition indicates that control of the resource has been assigned to the task. If all return codes for the resources named in the ENQ macro are 0, register 15 contains 0. If any of the return codes are not 0, register 15 contains the address of a storage area containing the return codes, as shown in Figure 9.

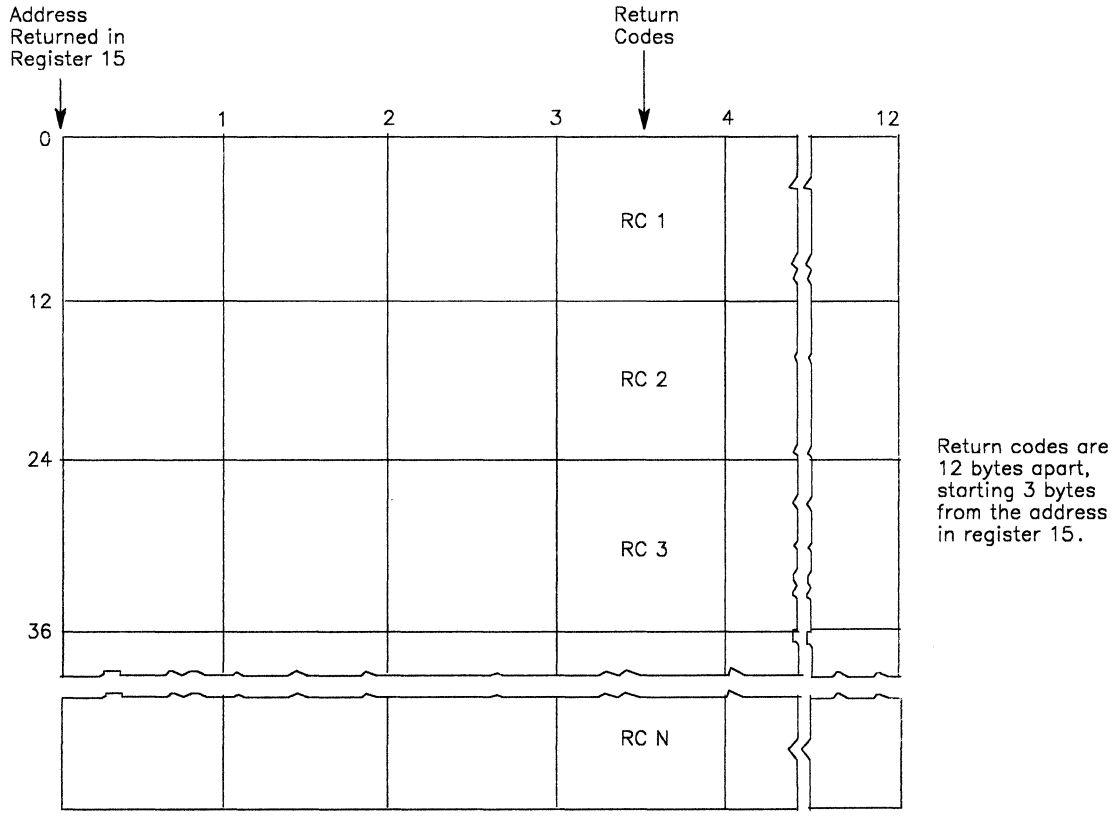


Figure 9. Return Code Area Used by ENQ

The return codes are placed in the parameter list resulting from the macro expansion in the same sequence as the resource names in the ENQ macro. The return codes are shown below.

Hexadecimal Code	Meaning
0	For RET=TEST, the resource is immediately available. For RET=USE, RET=HAVE, or ECB=, control of the resource has been assigned to the active task. For RET=CHNG, the status of the resource has been changed to exclusive. The ECB is not posted.
4	For RET=TEST or RET=USE, the resource is not immediately available. For RET=CHNG, the status cannot be changed to exclusive. For ECB=, the ECB will be posted when available.
8	For RET=TEST, RET=USE, RET=HAVE, or ECB=, a previous request for control of the same resource has been made for the same task. The task has control of resource. For RET=CHNG, the resource has not been enqueued. If bit 3 is on -- shared control of resource; if bit 3 of the first byte of the ENQ parameter list is off -- exclusive control. The ECB is not posted.
14	A previous request for control of the same resource has been made for the same task. The task does not have control of resource. The ECB is not posted.
18	For RET=HAVE, RET=USE, or ECB=, the limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. The ECB is not posted.
20	The matching task (the task specified in the MASID/MTCB parameters) owns the resource. The issuer of the ENQ macro may use the resource but it must ensure that the owning task does not terminate while the issuer of the ENQ macro is using the resource. If the issuer of the ENQ requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the issuer of the ENQ requested shared control and the owning task had requested shared control, this return code may indicate that a previous task had requested exclusive control. The issuing task must issue a DEQ to cancel this ENQ. The ECB will not be posted.
24	The issuing task will have exclusive control after the ECB is posted. The issuing task may use the resource but must ensure that the matching task does not terminate while the issuing task is using the resource. The issuing task must issue a DEQ to cancel the ENQ.
28	The issuing task cannot obtain exclusive control of the resource using the MASID/MTCB ENQ. The matching task's involvement with other tasks precludes control by the issuing task. This task must not issue a DEQ to cancel the ENQ. The ECB will not be posted.
44	The issuing task is violating a restriction of the MASID/MTCB ENQ in one or more of the following ways: • Another task has already issued this ENQ for this resource specifying the same MASID/MTCB. • The MASID/MTCB parameters specify a task that acquired control of the resource by using the MASID/MTCB ENQ. • The matching task requested ownership of the resource but has not yet been granted ownership. The ECB will not be posted. Return code 44 is never given by an ENQ RET=TEST, return code 4 is given instead.

Example 1

Operation: Unconditionally request exclusive control of a serially reusable resource that is known only within the address space (STEP), and place other tasks for the step nondispatchable until the requesting task has completed its operations on the resource.

```
ENQ  (MAJOR1,MINOR1,E,8,STEP),SMC=STEP
```

Example 2

Operation: Conditionally request control of a sharable resource in behalf of another task. The resource is known by more than one address space, and is only wanted if immediately available.

```
ENQ  (MAJOR2,MINOR2,S,4,SYSTEM),TCB=(R2),RET=USE
```

ENQ (List Form)

Use the list form of ENQ to construct a control program parameter list. Any number of resources can be specified in the ENQ macro, therefore, the number of *qname, rname*, and scope combinations in the list form of the ENQ macro must be equal to the maximum number of *qname, rname*, and scope combinations in any execute form of the macro that refers to that list form.

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

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENQ.
ENQ	
ƀ	One or more blanks must follow ENQ.
(	
qname addr	*qname addr:* A-type address, or register (2) - (12).
, ,*rname addr*	*rname addr:* A-type address, or register (2) - (12).
, ,E ,S	**Default:** E
, ,*rname length*	*rname length:* symbol or decimal digit. **Default:** assembled length of *rname*
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
)	
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	**Default:** RET=NONE
,SMC=NONE ,SMC=STEP	
,ECB=*ecb addr* ,TCB=0	*ecb addr:* A-type address. **Default:** SMC=NONE **Note:** ECB cannot be specified with RET above. **Note:** TCB or ECB must be specified on the list form if it is used on the execute form. ECB and TCB can be specified together. If TCB is specified but not ECB, then RET=CHNG, TEST or USE must be specified above.
,MASID=*0* ,MTCB=*0*	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the ENQ macro.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

ENQ (Execute Form)

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

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

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

(	**Note:** (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired then (,), and all parameters between (and) should not be specified. If something in the list is desired, the (,), and all parameters in the list should be specified as indicated at the left.
qname addr	*qname addr:* RX-type address, or register (2) - (12).
, ,*rname addr*	*rname addr:* RX-type address, or register (2) - (12).
, ,E ,S	
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12).
, ,STEP ,SYSTEM ,SYSTEMS	
)	**Note:** See note opposite (above.
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	
,SMC=NONE ,SMC=STEP ,ECB=*ecb addr* ,TCB=*tcb addr*	*ecb addr:* RX-type address, or register (2) - (12). *tcb addr:* RX-type address, or register (2) - (12). **Note:** ECB cannot be specified with RET above. **Note:** ECB and TCB can be specified together. If TCB is specified but not ECB, then RET=CHNG, TEST or USE must be specified above.
,MASID=*matching-asid addr* ,MTCB=*matching-tcb addr*	*matching-asid addr:* Rx-type address, or register (2)-(12). *matching-tcb addr:* Rx-type address, or register (2)-(12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) - (12).

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

,MF=(E,*ctrl addr)*
specifies the execute form of the ENQ macro using a remote control program parameter list.

Note: If ECB (or TCB) is specified in the execute form, ECB (or TCB=0) must be specified in the list form. If MASID and MTCB are specified, MASID=0 and MTCB=0 must be specified in the list form.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

ESPIE — Extended SPIE

The ESPIE macro extends the function of the SPIE (specify program interruption exits) macro to callers in 31-bit addressing mode. Callers in either 24-bit or 31-bit addressing mode can issue the ESPIE macro. Only callers in 24-bit addressing mode can issue the SPIE macro. For additional information concerning the relationship between the SPIE and the ESPIE macros, see "Interruption Services" in *SPL: Application Development Guide*.

The ESPIE macro performs the following functions using the options specified:

- Establishes an ESPIE environment (that is, identifies the interruption types that are to cause entry to the ESPIE exit routine) by executing the SET option of the ESPIE macro.
- Deletes an ESPIE environment (that is, cancels the current SPIE/ESPIE environment) by executing the RESET option of the ESPIE macro
- Determines the current SPIE/ESPIE environment by executing the TEST option of the ESPIE macro

The following description of the ESPIE macro also appears in *Application Development Macro Reference* with the exception of interruption type 17. This interruption type designates page faults, and its use is restricted to programs that are APF-authorized or execute in key 0-7.

SET Option

The SET option of the ESPIE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
SET	
,*exit addr*	*exit addr:* A-type address or register (2) - (12).
,*(interruptions)*	*interruptions:* decimal numbers 1 - 15 or 17 expressed as single values: (2, 3, 4, 7, 8, 9, 10) ranges of values: ((2, 4), (7, 10)) combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr:* A-type address or register (2) - (12).

The parameters are explained as follows:

SET
indicates that an ESPIE environment is to be established.

,*exit addr*
specifies the address of the exit routine to be given control when program interruptions of the type specified by *interruptions* occur. The exit routine will receive control in the same addressing mode as the issuer of the ESPIE macro.

,(interruptions)
indicates the interruption types that are being trapped. The interruption types are:

Number	Interruption Type
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow (maskable)
9	Fixed-point divide
10	Decimal overflow (maskable)
11	Decimal divide
12	Exponent overflow
13	Exponent underflow (maskable)
14	Significance (maskable)
15	Floating-point divide
17	Page fault

These interruption types can be designated as one or more single numbers, as one or more pairs of numbers (designating ranges of values), or as any combination of the two forms. For example, (4,8) indicates interruption types 4 and 8; ((4,8)) indicates interruption types 4 through 8.

If a program interruption type is maskable, the corresponding program mask bit in the PSW is set to 1. If a maskable interruption is not specified, the corresponding bit in the PSW is set to 0. Interruption types not specified above (except for type 17) are handled by the control program. The control program forces an abend with the program check as the completion code. If an ESTAE-type recovery routine is also active, the SDWA indicates a system-forced abnormal termination. The registers at the time of the error are those of the control program.

Note: For ESPIE and SPIE — If you are using vector instructions and an interruption of 8, 12, 13, 14, or 15, occurs, your recovery routine can check the exception extension code (the first byte of the two-byte interruption code in the EPIE or PIE) to determine whether the exception was a vector or scalar type of exception.

,PARAM = *list addr*
specifies the fullword address of a parameter list that is to be passed by the caller to the exit routine.

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.

On return from the SET option of the ESPIE macro, the registers contain the following information:

Register	Content
0	Used as a work register by the macro.
1	Token representing the previously active SPIE/ESPIE. environment
2-13	Unchanged.
14	Used as a work register by the macro.
15	Return code of 0.

Example 1
Operation: Give control to an exit routine for interruption types 1 and 4. EXIT is the location of the exit routine to be given control and PARMLIST is the location of the user-parameter list to be used by the exit routine.

```
ESPIE SET,EXIT,(1,4),PARAM=PARMLIST
```

Example 2
Operation: Give control to the exit routine located at EXIT when a page fault occurs.

```
ESPIE SET,EXIT,(17)
```

RESET Option

The RESET option of the ESPIE routine cancels the active SPIE/ESPIE environment and restores the SPIE/ESPIE environment specified by *token*.

The RESET option of the ESPIE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
RESET	
,token	*token:* RX-type address or register (1) or (2) - (12).

The parameters are explained as follows:

RESET
indicates that the current ESPIE environment is to be deleted and the previously active SPIE/ESPIE environment specified by *token* is to be re-established.

,token
specifies a fullword that contains a token representing the previously active SPIE/ESPIE environment. This is the same token that ESPIE processing returned to the caller when the ESPIE trap was established using the SET option of the ESPIE macro.

If the token is zero, all SPIEs and ESPIEs are deleted.

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.

On return from ESPIE RESET, the contents of the registers are as follows:

Register	Contents
0	Used as a work register by the macro.
1	Token identifying the new active SPIE/ESPIE environment.
2-13	Unchanged.
14	Used as a work register by the macro.
15	Return code of 0.

Example

Operation: Cancel the current SPIE/ESPIE environment and restore the SPIE/ESPIE environment represented by the contents of TOKEN.

```
ESPIE RESET,TOKEN
```

TEST Option

The TEST option of the ESPIE macro determines the active SPIE/ESPIE environment and returns the information in a four-byte parameter list.

The TEST option of the ESPIE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
TEST	
,parm addr	*parm addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained as follows:

TEST
indicates a request for information concerning the active or current SPIE/ESPIE environment. ESPIE processing returns this information to the caller in a four-word parameter list located at *parm addr*.

,parm addr
specifies the address of a four-word parameter list aligned on a fullword boundary. The parameter list has the following form:

Word	Content
0	Address of the exit routine (31-bit address with the high-order bit set to 0)
1	Address of the user-defined parameter list
2	Mask of program interruption types
3	Zero

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.

On return from ESPIE TEST, the registers contain the following information:

Register	Contents
0	Used as a work register by the macro.
1-13	Unchanged.
14	Used as a work register by the macro.
15	Return code as follows:

Code	Meaning
0	An ESPIE exit is active and the four-word parameter list contains the the information specified in the description of the *parm addr* parameter.
4	A SPIE exit is active. Word 1 of the parameter list described under *parm addr* contains the address of the current PICA. Words 0, 2, and 3 of the parameter list contain no relevant information.
8	No SPIE or ESPIE is active. The contents of the four-word parameter list contain no relevant information.

Example

Operation: Identify the active SPIE/ESPIE environment. Return the information about the exit routine in the four-word parameter list, PARMLIST. Also return, in register 15, an indication of whether a SPIE, ESPIE, or neither is active.

```
ESPIE TEST,PARMLIST
```

ESPIE (List Form)

The list form of the ESPIE macro builds a non-executable problem program parameter list that can be referred to or modified by the execute form of the ESPIE macro. The list form of ESPIE is valid only for ESPIE SET.

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

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

SET	
,*exit addr*	*exit addr:* A-type address. **Note:** This parameter must be specified on either the list or the execute form of the macro.
,(interruptions)	*interruptions:* decimal number 1 - 15 or 17 expressed as single values: (2, 3, 4, 7, 8, 9, 10) ranges of values: ((2, 4), (7, 10)) combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr:* A-type address.
,MF=L	

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

,MF=L
specifies the list form of the ESPIE macro.

Example

Operation: Build a non-executable problem program parameter list that will cause control to be transferred to the exit routine, EXIT, for the interruption types specified in the execute form of the macro. Provide the address of the user parameter list, PARMLIST.

```
LIST1 ESPIE SET,EXIT,,PARAM=PARMLIST,MF=L
```

ESPIE (Execute Form)

The execute form of the ESPIE macro can refer to and modify the parameter list constructed by the list form of the ESPIE macro. The execute form of ESPIE is valid only for ESPIE SET.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
SET	
,*exit addr*	*exit addr:* RX-type address or register (2) - (12). **Note:** This parameter must be specified on either the list or the execute form of the macro.
,(interruptions)	*interruptions:* decimal number 1 - 15 or 17 expressed as single values: (2, 3, 4, 7, 8, 9, 10) ranges of values: ((2, 4), (7, 10)) combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr:* RX-type address or register (1) or (2) - (12).
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl addr*)
specifies the execute form of the ESPIE macro using a remote control program parameter list.

Example

Operation: Give control to a installation exit routine for interruption types 1, 4, 6, 7, and 8. The exit routine address and the address of a user parameter list for the exit routine are provided in a remote control program parameter list at LIST1.

```
ESPIE SET,,(1,4,(6,8)),MF=(E,LIST1)
```

ESTAE and ESTAEX — Specify Task Abnormal Exit Extended

The ESTAE macro allows the user to intercept a scheduled ABEND. Control is given to a user-specified recovery routine that can, for example, perform pre-termination processing, diagnose the cause of ABEND, and specify a retry address to try to avoid the termination. These recovery routines operate in both problem program and supervisor modes.

The addressing mode in which the ESTAE macro expansion executes becomes the addressing mode in which the ESTAE exits and retry routines execute (that is, the ESTAE exits and retry routines execute in the same addressing mode as the issuer of the ESTAE macro.)

ESTAEX is the preferred interface. You can use ESTAE, however, if your program is in primary mode, and the primary, secondary, and home address spaces are the same. Depending on whether you code ESTAE or ESTAEX, the system passes the address of the user-specified parameter list differently. The SDWAPARM field in the SDWA contains either the address of the parameter list (ESTAE), or the address of a doubleword that contains the address and ALET of the parameter list (ESTAEX). See "Key Fields in the SDWA" in *SPL: Application Development Guide*

For information about how to select the macro for an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1. If you are executing in 31-bit addressing mode, you must use the MVS/XA™ version of this macro.

The descriptions of ESTAE and ESTAEX in this book are:

- The standard form of the ESTAE macro, which includes general information about the ESTAE and ESTAEX macros, with some specific information about the ESTAE macro. The syntax of the ESTAE macro is presented, and all ESTAE parameters are explained.
- The standard form of the ESTAEX macro, which includes information specific to the ESTAEX macro. The syntax of the ESTAEX macro is presented.
- The list form of ESTAE and ESTAEX
- The execute form of ESTAE and ESTAEX

  Comments in the syntax identify parameters that are not valid for certain ASC modes.

The description of the ESTAE macro follows. The ESTAE macro is also described in *Application Development Macro Reference* with the exception of the CANCEL, BRANCH, SVEAREA, KEY, RECORD, and TOKEN parameters. These parameters are restricted in use, and, therefore, are described only in here.

"ESTAE-Type Recovery Routines" in *SPL: Application Development Guide* provides more information.

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

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

exit addr 0	*exit addr:* A-type address, or register (2) - (12).
,CT ,OV	**Default:** CT
,PARAM=*list addr*	*list addr:* A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	**Default:** PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default:** ASYNCH=YES
,CANCEL=YES ,CANCEL=NO	**Default:** CANCEL=YES
,TERM=NO ,TERM=YES	**Default:** TERM=NO
,BRANCH=NO ,BRANCH=YES,SVEAREA=*save addr*	**Default:** BRANCH=NO *save addr:* A-type address, or register (2) - (12) or (13).
,KEY=SAVE ,KEY=*storage key*	*storage key:* any numeral in the range 0-15.
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,TOKEN=*token addr*	*token addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows.

exit addr
0
specifies the 31-bit address of an ESTAE recovery routine to be entered if the task issuing this macro terminates abnormally. The recovery routine executes in the addressing mode of the issuer of the ESTAE. If you specify 0, the most recent ESTAE routine is deleted.

,CT
,OV
specifies the creation of a new ESTAE exit (CT) or indicates that parameters passed in this ESTAE macro are to overlay the data contained in the previous ESTAE routine (OV).

,PARAM = *list addr*
specifies the 31-bit address of a user-defined list containing data to be used by the ESTAE routine when it is scheduled for execution.

,XCTL = NO
,XCTL = YES
specifies that the ESTAE environment will be deleted (NO) or will not be deleted (YES) if this program issues an XCTL macro.

,PURGE = NONE
,PURGE = QUIESCE
,PURGE = HALT
specifies that all outstanding requests for I/O operations are not to be saved when the ESTAE routine gets control (HALT) or that I/O processing is to be allowed to continue normally when the ESTAE routine gets control (NONE) or that all outstanding requests for I/O operations are to be saved when the ESTAE routine is taken (QUIESCE). If QUIESCE is specified, the user's retry routine can restore the outstanding I/O requests.

PURGE = NONE specifies that all control blocks affected by input/output processing can continue to change during ESTAE routine processing. If you specify PURGE = NONE, and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion develops when an input/output interruption occurs, even if the ESTAE routine is in progress. Thus, it will appear that the ESTAE routine failed when, in reality, input/output processing caused the failure.

Notes:

1. You should understand PURGE processing before using this parameter. For information on PURGE processing, see *System Programming Reference.*
2. If you specify PURGE = HALT, or PURGE = QUIESCE but I/O is not restored,
   - **While using SAM or ISAM,** only the input/output event on which the purge is done will be posted. Subsequent event control blocks (ECBs) will not be posted. If you issue further data management macros, such as GET/PUT, READ/WRITE or CLOSE, after a PURGE is issued during ESTAE recovery, a wait may occur in an access method module.
   - **While using ISAM,**
     - The ISAM check routine will treat purged I/O as normal I/O.
     - Part of the data set might be destroyed if the data set was being updated or added to when the failure occurred.

,ASYNCH = YES
,ASYNCH = NO
specifies that asynchronous exit processing will be allowed (YES) or prohibited (NO) while the user's ESTAE routine is executing.

ASYNCH = YES must be coded if:

- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the ESTAE 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 ESTAE routine issues the CHECK macro for any access method that requires asynchronous interruptions to complete normal input/output processing.

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

,CANCEL = YES
,CANCEL = NO
specifies whether you want to allow the recovery routine to be interrupted by cancel or detach processing.

To allow a recovery routine to be interrupted, specify CANCEL=YES.

To prevent a recovery routine from being interrupted, specify CANCEL=NO. If a cancel or detach is attempted against a recovery routine for which you have specified CANCEL=NO, MVS defers cancel and detach processing until the recovery routine returns control to the system.

Usage Notes:

1. If a recovery routine that runs under the CANCEL=NO option can be called by an unauthorized program running under the same task, IBM recommends that you specify ASYNCH=NO for each ESTAE(X) macro that the recovery routine issues. This also includes any ESTAE(X) macros issued by programs that the recovery routine calls.
2. If a recovery routine running under the CANCEL=NO option calls an unauthorized program, cancel and detach processing is also deferred for the called program.

,TERM=NO
,TERM=YES
specifies that the ESTAE routine will be scheduled (YES) or will not be scheduled (NO) in the following situations:

- Cancel by operator
- Forced logoff
- Expiration of job step timer
- Exceeding of wait time limit for job step
- ABEND condition because of DETACH of an incomplete subtask when the STAE option was not specified on the DETACH
- ABEND of the attaching task when the ESTAE macro was issued by a subtask
- ABEND of job step task when a non-job step task requested ABEND with the STEP option.

When the ESTAE routine is entered because of one of the preceding reasons, re-try is not permitted. If a dump is requested at the time of ABEND, it is taken before entry into the ESTAE routine.

Note: If DETACH was issued with the STAE parameter, the following occurs for the task to be detached:

- All ESTAE routines are entered.
- The most recently established STAE routine is entered.
- All STAI/ESTAI routines are entered unless one of the STAI routines issues return code 16.

In these cases, entry to the routine occurs before dumping and re-try is not permitted.

,BRANCH=NO
,BRANCH=YES,SVEAREA=*save addr*
specifies that an SVC entry to the ESTAE service routine is to be performed (NO) or that a branch entry is to be performed (YES). The save area is a 72-byte area used to save the general registers. If the caller is not in key zero, the KEY parameter must be specified.

BRANCH and SVEAREA are not valid on ESTAEX.

,KEY=SAVE
,KEY=*storage key*
specifies that supervisor state users who are not in key zero can use the branch entry interface to the ESTAE service routine.

If the user specifies KEY=SAVE, the system saves the current PSW protection key in register 2 and issues a set protection key instruction (SPKA) to change to protection key zero. When the ESTAE service routine returns control, it restores the original PSW key from register 2. Therefore, the user should save register 2 before the macro

expansion and restore it afterwards. Specifying KEY=SAVE destroys the contents of register 2 during the macro expansion.

On the other hand, if the user knows the current PSW protection key, he may specify it directly in the form KEY=(0-15) to eliminate saving and restoring the original protection key. This procedure eliminates an IPK instruction and prevents the use of register 2 in the macro expansion.

KEY is not valid on ESTAEX.

,RECORD=NO
,RECORD=YES
specifies that the system diagnostic work area (SDWA) is not to be written to SYS1.LOGREC (NO) or that the entire SDWA (including the fixed length base, the variable length recording area, and the recordable extensions) is to be written to SYS1.LOGREC (YES).

,TOKEN=*token addr*
specifies that a four-byte token is to be associated with the ESTAE routine. Unauthorized or accidental destruction of the ESTAE routine is prevented because the ESTAE cannot be canceled or overlaid unless the same token is specified.

With CT (create): ESTAE processing places the token created for this request in the location specified by *token addr* as well as in the ESTAE parameter list.

With OV (overlay): ESTAE processing locates the specified ESTAE routine for the current RB and replaces the routine information. If there are any newer ESTAE routines for the RB, they are deleted.

With 0 (cancel): ESTAE processing locates the specified ESTAE routine for the current RB and deletes the routine. Any newer ESTAE routines for the RB are deleted.

,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 specified are at the discretion of the user, and may be any valid coding values.

Control returns to the instruction following the ESTAE macro. When control returns, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of ESTAE request.
04	ESTAE OV was specified but ESTAE CT was performed. Register 0 contains one of the following reason codes:

Hexadecimal Code	Meaning
00	No valid SCB existed.
04	The last SCB was not owned by the user's RB.
08	The last SCB was not created at the current linkage stack level.
0C	The last SCB was not an ESTAE SCB.

0C	Delete (an exit address equal to zero) was specified, and either • There are no exits for this TCB, • The most recent exit is not owned by the caller, • The most recent exit is not an ESTAE exit, or • The ESTAE was created with the TOKEN parameter and on a delete request, either – The token was not specified or – The token does not match.
10	An unexpected error was encountered while processing this request.
14	ESTAE was unable to obtain storage for an SCB.
18	ESTAE OV request was invalid for one of the following reasons: • ESTAE OV with the TOKEN parameter was specified but – No SCB exists or – The SCB is not an ESTAE SCB created with the matching token value by the current RB. • ESTAE OV without the TOKEN parameter was specified but the SCB was created with the TOKEN parameter.
1C	ESTAE was unable to access the input parameter list.
20	XCTL=YES was rejected because the linkage stack was not at the same level as it was when the RB was created.
24	Delete (an exit address equal to zero) was specified but rejected because no ESTAEs were active for the current linkage stack level.
28	ESTAE OV was specified but rejected because no ESTAEs were active for the current linkage stack level.

Example 1

Operation: If an error occurs, pass control to the ESTAE routine specified by register 4, allow asynchronous exit processing, do not allow special error processing, do not branch enter, and default to CT (create) and PURGE = NONE.

```
ESTAE (4),ASYNCH=YES,TERM=NO,BRANCH=NO
```

Example 2

Operation: If an error occurs, pass control to the ESTAE routine specified by register 4. The address of the ESTAE parameter list is in register 2. Place the token associated with this ESTAE routine in TOKENFLD.

```
ESTAE (4),PARM=(2),TOKEN=TOKENFLD
```

Example 3

Operation: If an error occurs, pass control to the ESTAE routine labeled ADDR, allow synchronous exit processing, halt I/O, allow special error processing, branch enter, use the 72-byte save area at SADDR, and execute the execute form of the macro. EXEC is the label of the ESTAE parameter list built by a list form of the macro elsewhere in this program.

```
ESTAE ADDR,ASYNCH=YES,PURGE=HALT,TERM=YES,BRANCH=YES,    X
      SVEAREA=SADDR,MF=(E,EXEC)
```

Example 4

Operation: Request an overlay of the existing ESTAE recovery routine with the following options: the address of the parameter list is at PLIST, I/O will be halted, no asynchronous exits will be taken, ownership will be transferred to the new request block resulting from any XCTL macros.

```
ESTAE   ADDR,OV,PARAM=PLIST,XCTL=YES,PURGE=HALT,ASYNCH=NO
```

Example 5

Operation: Provide the pointer to the recovery code in the register called EXITPTR, place the address of the ESTAE parameter list in register 9. Register 8 points to the area where the ESTAE parameter list (created with the MF = L option) was moved.

```
ESTAE   (EXITPTR),PARAM=(9),MF=(E,(8))
```

ESTAEX — Specify Task Abnormal Exit Extended

The ESTAEX macro provides all of the function that ESTAE provides. Any program, whether its in AR mode, primary mode, or cross memory mode can issue ESTAEX. Callers of the ESTAEX macro must be enabled. IBM recommends that you always use ESTAEX instead of using ESTAE.

For callers in AR mode:

- Before issuing ESTAEX, issue SYSSTATE ASCENV=AR. The ASCENV=AR parameter on the SYSSTATE macro ensures that ESTAEX generates code appropriate for AR mode.
- ESTAEX system parameters must be located in the caller's primary address space.
- User parameters, specified on the PARAM keyword, can be located in any address space.

The parameters on the standard form of the ESTAEX macro are the same as for the standard form of the ESTAE macro, except BRANCH, SVEAREA, and KEY, which are not valid for ESTAEX.

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

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

exit addr 0	*exit addr:* A-type address, or register (2) - (12).
,CT ,OV	**Default:** CT
,PARAM=*list addr*	*list addr:* A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	**Default:** PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default:** ASYNCH=YES
,CANCEL=YES ,CANCEL=NO	**Default:** CANCEL=YES
,TERM=NO ,TERM=YES	**Default:** TERM=NO
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,TOKEN=*token addr*	*token addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under the syntax for the standard form of the ESTAE macro. However, when control is returned to the instruction following the ESTAEX, the return code in register 15 may be different. The following are the return codes for ESTAEX:

Hexadecimal Code	Meaning
00	Successful completion of ESTAEX request.
04	ESTAEX OV was specified but ESTAEX CT was performed. Register 0 contains one of the following reason codes:

Hexadecimal Code	Meaning
00	No valid SCB existed.
04	The last SCB was not owned by the user's RB.
08	The last SCB was not created at the current linkage stack level.
0C	The last SCB was not an ESTAE SCB.

08	An invalid type of ESTAEX request was detected.
0C	Delete (an exit address equal to zero) was specified, and either • There are no exits for this TCB, • The most recent exit is not owned by the caller, • The most recent exit is not an ESTAE exit, or • The ESTAE was created with the TOKEN parameter and on a delete request, either – The token was not specified or – The token does not match.
10	An unexpected error was encountered while processing this request.
14	ESTAEX was unable to obtain storage for an SCB.
18	ESTAEX OV was requested either • With the TOKEN parameter specified and the SCB is not owned by the current RB or • Without the TOKEN parameter specified but the SCB was created with the TOKEN parameter.
1C	ESTAEX was unable to access the input parameter list.
20	XCTL=YES was ignored because the linkage stack was not at the same level as it was when the RB was created.
24	Delete (an exit address equal to zero) was specified but rejected because no ESTAEs were active for the current linkage stack level.
28	The caller was disabled.
2C	The caller was locked.
30	The caller had FRRs on the current FRR stack.
34	The caller was in SRB mode.

ESTAE and ESTAEX (List Form)

The list form of ESTAE or ESTAEX is used to construct a remote control parameter list.

The list form of ESTAE or ESTAEX is written as follows:

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

b                                One or more blanks must precede ESTAE or ESTAEX.

ESTAE
ESTAEX

b                                One or more blanks must follow ESTAE or ESTAEX.

   exit addr                     exit addr: A-type address.

   ,PARAM=list addr              list addr: A-type address.

   ,PURGE=NONE                   Default: PURGE=NONE
   ,PURGE=QUIESCE
   ,PURGE=HALT

   ,ASYNCH=YES                   Default: ASYNCH=YES
   ,ASYNCH=NO

   ,CANCEL=YES                   Default: CANCEL=YES
   ,CANCEL=NO

   ,TERM=NO                      Default: TERM=NO
   ,TERM=YES

   ,RECORD=NO                    Default: RECORD=NO
   ,RECORD=YES

   ,RELATED=value                value: any valid macro keyword specification.

,MF=L
```

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

,MF=L
specifies the list form of the ESTAE or ESTAEX macro.

ESTAE or ESTAEX (Execute Form)

A remote control parameter list is used in, and can be modified by, the execute form of the ESTAE or ESTAEX macro. The control parameter list can be generated by the list form of the ESTAE or ESTAEX macro. Any combination of exit addr, PARAM, XCTL, PURGE, ASYNCH, TERM, RECORD, and TOKEN can be specified to dynamically change the contents of the remote ESTAE or ESTAEX parameter list. If the TOKEN parameter was previously specified and is to be used again without change, TKNPASS=YES must be coded. Any fields not specified on the macro remain as they were before the current ESTAE or ESTAEX request was made.

The execute form of the ESTAE or ESTAEX macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESTAE or ESTAEX.
ESTAE ESTAEX	
b	One or more blanks must follow ESTAE or ESTAEX.
exit addr 0	*exit addr:* RX-type address, or register (2) - (12).
,CT ,OV	
,PARAM=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	
,ASYNCH=YES ,ASYNCH=NO	
,CANCEL=YES ,CANCEL=NO	**Default:** CANCEL=YES
,TERM=NO ,TERM=YES	
,BRANCH=NO ,BRANCH=YES,SVEAREA=*save addr*	**Note:** BRANCH and SVEAREA are not valid on ESTAEX. *save addr:* RX-type address, or register (2) - (12) or (13).
,KEY=SAVE ,KEY=*storage key*	*storage key:* any numeral in the range 0-15. **Note:** KEY is not valid on ESTAEX.
,RECORD=NO ,RECORD=YES	
,TOKEN=*token addr*	*token addr:* RX-type address, or register (2) - (12).
,TKNPASS=NO ,TKNPASS=YES	**Default:** TKNPASS=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the ESTAE or ESTAEX macro, with the following exceptions:

,TKNPASS = NO
,TKNPASS = YES
specifies that a previously-specified token, indicated in the parameter list, should be ignored (NO), or should remain part of the specification (YES).

,MF = (E,*ctrl addr)*
specifies the execute form of the ESTAE or ESTAEX macro using a remote control parameter list.

ETCON — Connect Entry Table

The ETCON macro connects one or more previously created entry tables to the specified linkage table indexes in the current home address space. If an entry table is connected to a system linkage index (an index reserved with the SYSTEM=YES option of the LXRES macro), the entry table is connected to the linkage table of every address space, both present and future.

The restrictions on the use of the ETCON macro are as follows:

- If an entry table contains entries that cause address space switches, the entry table owner must have previously established authorization to issue PT and SSAR instructions to the home address space.
- An entry table can be connected only once to a single linkage table.
- The linkage index and the entry table being connected must be under the same ownership.

Any violation of these restrictions causes the caller to be abnormally terminated.

The connection created by the ETCON macro remains in effect until one of the following occurs:

- The ETDIS macro removes the connection.
- The entry table owner terminates.
- The address space to which the table is connected terminates unless the connection was to a system linkage index.
- The system is re-IPLed.

The caller must be in supervisor state or PKM 0-7, executing in primary mode, enabled, and unlocked. The parameter list passed to the ETCON macro must be addressable in primary mode at the time the macro is issued. Register 13 must point to a standard register save area that must also be addressable in primary mode.

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

The ETCON macro is written as follows:

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

TKLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,LXLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

TKLIST = *address*
specifies the address of a list of fullword tokens representing the entry tables to be connected to the linkage table. The first entry in the list must be the number of tokens that follow (from 1 to 32). The tokens are the values returned in register 0 when the ETCRE macro is issued.

,LXLIST = *addr*
specifies the address of a list of linkage index values to which the specified entry tables are to be connected. The list contains fullword entries, the first of which must be the number of linkage index values that follow (from 1 to 32). The number of linkage indexes must be the same as the number of tokens. The first entry table is connected to the first linkage index; the second entry table is connected to the second linkage index, and so on.

,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.

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The specified connections were successfully made.

ETCON (List Form)

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

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

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

TKLIST=*addr*	*addr:* A-type address.
,LXLIST=*addr*	*addr:* A-type address.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the ETCON macro.

ETCON (Execute Form)

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

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

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

TKLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,LXLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*cntl addr*)	*cntl addr:* RX-type address or register (0) - (12).

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

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

ETCRE — Create Entry Table

The ETCRE macro builds a program call entry table based upon descriptions of each entry. A token representing the created entry table is returned to the requestor. You must use this token in all subsequent references to the entry table.

Before issuing ETCRE, the caller must create the ETD parameter list that ETCRE uses as input. The parameter list defines the names and characteristics of the program call (PC) routines that the entry table will define. To create the parameter list, the caller can issue the ETDEF macro or can code the data constants needed to define the list. If data constants are coded, the caller can use mapping macro IHAETD to map them.

The created entry table is owned by the cross memory resource ownership task in the current home address space. When the cross memory resource ownership task terminates, entry tables are disconnected and freed.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The ETD specified by ENTRIES must also be addressable in primary mode when the macro is issued.

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	The 32-bit token associtated with the new entry table
1	Used as a work register by the macro
2 - 13	Unchanged
14	Used as a work register by the macro
15	Return code

Note: Programs written before MVS/SP Version 3, which use data constants to define the parameter list (the resulting ETD was called a format 0 ETD) and which use IHAETD to map the data area, will still work. For information about the format 0 ETD, see *Diagnosis: Data Areas*.

The ETCRE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETCRE.
ETCRE	
b	One or more blanks must follow ETCRE.
ENTRIES=*addr*	*addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

ENTRIES = *addr*
specifies the address of the parameter list that defines the PC routines.

An entry index value that does not have a description results in an invalid entry in the entry table. If the program name field in an ETD entry contains zeroes, an invalid entry is created for that entry index. A program call to an invalid entry causes the caller to be abnormally terminated. The ETCRE caller is abnormally terminated if any of the reserved fields are nonzero or if the system cannot locate the specified program name.

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

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The entry table is successfully created.

Example

The following example shows the relationship between the ETCRE and the ETDEF macros. ETDEF builds an entry table descriptor (ETD) that contains two ETD entries. The first entry, associated with PROGRAM1, is for a PC routine that runs in supervisor state. The second entry, associated with PROGRAM2, is for a PC routine that runs in problem state.

```
*
* CREATE THE ENTRY TABLE
*
         .
         .
         LA    2,ETSTART
         ETCRE ENTRIES=(2)
         .
         .
*
* DEFINE START OF ETD
*
ETSTART  ETDEF TYPE=INITIAL              START ETD
*
* DEFINE ENTRIES
*
ETEX2    ETDEF TYPE=ENTRY,PROGRAM='PROGRAM1',AKM=(0:15)
         ETDEF TYPE=ENTRY,PROGRAM='PROGRAM2',AKM=(0:7)
*
* DEFINE END OF ETD
*
        ETDEF TYPE=FINAL
```

ETDEF — Create an Entry Table Descriptor (ETD)

The ETDEF macro builds and modifies the parameter that the ETCRE macro uses to build an **entry table.** The parameter, called the **entry table descriptor (ETD),** consists of a header, followed by one or more entries, called **ETD entries**, each one describing a PC routine. The address of the ETD is input to the ENTRIES parameter on the ETCRE macro.

The TYPE parameter on the ETDEF macro determines which process the ETDEF macro is to perform:

- ETDEF TYPE = INITIAL generates the header for the ETD. (Issue this macro once for each ETD.)
- ETDEF TYPE = ENTRY generates one ETD entry. (You can issue this macro up to 128 times for each ETD.)
- ETDEF TYPE = FINAL terminates the ETD. (Issue this macro once for each ETD.)
- ETDEF TYPE = SET,ETEADR replaces the entire contents of an existing ETD entry.
- ETDEF TYPE = SET,HEADER changes an existing ETD header.

You can create an ETD at time of compile through TYPE = INITIAL, TYPE = ENTRY, and TYPE = FINAL parameters and initialize the information for the entries at time of execution through TYPE = SET,ETEADR. Therefore, ETDEF with the TYPE = INITIAL, TYPE = ENTRY, and TYPE = FINAL parameters works like a list form of the macro. However, unlike the execute form of a macro, which changes only the values you specify, the TYPE = SET form of ETDEF completely *replaces* the contents of an ETD entry, taking the default values for any parameters you omit. This section describes the two forms separately.

Although ETDEF is the preferred programming interface, if you have an existing ETD and you want to update the parameters (for example, change the user parameter), you might choose to use the IHAETD mapping macro instead of ETDEF. If you change an existing ETD, without using any of the function of MVS/SP Version 3, you can use IHAETD with the format number of "0". The format of IHAETD is in *Diagnosis: Data Areas* under "ETD".

Note: When changing code to use ETDEF in place of the IHAETD mapping macro, be sure to specify PC = BASIC so that the PC does not become a stacking PC. If you want to change an existing PC routine to a stacking PC, be sure to change the PT instruction in the PC routine to a PR.

The caller of the ETDEF macro has the following requirements:

Authorization:	Problem or Supervisor state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	31-bit or 24-bit
ASC mode:	Primary
Serialization:	Not applicable

The ETDEF macro does not use any registers, except for those you use to specify parameters.

TYPE = INITIAL, TYPE = ENTRY, and TYPE = FINAL Parameters

The ETDEF macro with the TYPE = INITIAL, TYPE = ENTRY, and TYPE = FINAL options works like a list form of a macro. This form is described as follows:

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

	Valid parameters (Required parameters are underlined)
TYPE = INITIAL TYPE = ENTRY	RELATED PROGRAM or ROUTINE, AKM, EKM, ARR, ASCMODE, EAX, EK, PARM1, PARM2, PC, PKM, SASN, SSWITCH, STATE, RELATED, ASYNCH, CANCEL
TYPE = FINAL	RELATED
,AKM = *key-list*	key-list: List of keys or key ranges where a key is a number 0 - 15.
,ARR = *arr*	*arr:* A-type address, or alphanumeric character string enclosed by single quotation marks.
,ASYNCH = YES ,ASYNCH = NO	**Default:** ASYNCH = YES Valid only when ARR is also coded.
,CANCEL = YES ,CANCEL = NO	**Default:** CANCEL = YES Valid only when ARR is also coded.
,ASCMODE = PRIMARY ,ASCMODE = AR	**Default:** ASCMODE = PRIMARY
,EAX = *eax-value*	*eax-value:* Half-word decimal digit.
,EK = *entry-key*	*entry-key:* Decimal digit 0 - 15.
,EKM = *key-list*	*key-list:* List of keys or key ranges where a key is a number 0 - 15. **Note**: EKM is required with PKM = REPLACE.
,PARM1 = *user-parm1*	*user-parm1:* A-type address or string of up to 4 characters enclosed by single quotation marks.
,PARM2 = *user-parm2*	*user-parm2:* A-type address or string of up to 4 characters enclosed by single quotation marks.
,PC = STACKING ,PC = BASIC	**Default:** PC = STACKING
,PROGRAM = *pgm-name*	*pgm name:* String of up to 8 alphanumeric characters, optionally enclosed by single quotation marks.
,ROUTINE = *rtn-addr*	*rtn addr:* A-type address.
,PKM = OR ,PKM = REPLACE	**Default:** PKM = OR
,RAMODE = 31 ,RAMODE = 24	**Default:** RAMODE = 31
,RELATED = *value*	*value:* Any valid macro parameter specification.
,SASN = OLD ,SASN = NEW	**Default:** SASN = OLD
,SSWITCH = NO ,SSWITCH = YES	**Default:** SSWITCH = NO

,STATE=PROBLEM
,STATE=SUPERVISOR

Default: STATE=PROBLEM

TYPE=INITIAL
generates the header for the ETD.

TYPE=ENTRY
generates an ETD entry. The system uses the defaults for any parameters you do not specify on the ETDEF TYPE=ENTRY macro. When you later specify ETDEF TYPE=SET, that macro initializes the **entire** ETD entry.

TYPE=FINAL
specifies that the ETD is complete.

,AKM=*key-list*
specifies a list of keys (0 through 15) or key ranges, optionally enclosed in parentheses, that identifies the authorized keys in which a problem program can use the PC routine. For example, AKM=(2,(3),5:8,(10:12),15) would authorize keys 2, 3, 5, 6, 7, 8, 10, 11, 12, and 15.

,ARR=*arr*
specifies the associated recovery routine (ARR) that receives control if the stacking-PC routine abends. You can use the A-type address of the routine, or the name of the routine (an alphanumeric character string) enclosed in single quotation marks. If you use the name of the program, the program must be on the active LPA queue (FLPA or MLPA) or be in the PLPA or nucleus. The recovery routine will be entered in 31-bit mode. ARR is not valid with PC=BASIC.

,ASYNCH=YES
,ASYNCH=NO
specifies whether or not the ARR can be interrupted by asynchronous exits. ASYNCH=YES specifies that the ARR can be interrupted by asynchronous exits. ASYNCH=NO specifies that the ARR cannot be interrupted by asynchronous exits. ASYNCH=YES is the default. ASYNCH is valid only with ARR.

,CANCEL=YES
,CANCEL=NO
specifies whether or not the ARR can be interrupted by CANCEL/DETACH processing. CANCEL=YES specifies that the ARR can be interrupted by CANCEL/DETACH processing. CANCEL=NO specifies that the ARR cannot be interrupted by CANCEL/DETACH processing. CANCEL=YES is the default. CANCEL is valid only with ARR. To specify CANCEL=NO, one of the following conditions must be true for the stacking PC routine protected by the ARR:

- The stacking PC routine runs in supervisor state.
- The entry key for the stacking PC routine is a system key
- The stacking PC routine runs with a system key valid for the entry key mask that will either replace or be ORed with the PKM.

,ASCMODE=PRIMARY
,ASCMODE=AR
specifies that the stacking PC routine will execute in primary ASC mode (ASCMODE=PRIMARY) or in AR ASC mode (ASCMODE=AR). ASCMODE=AR is not valid with PC=BASIC. ASCMODE=PRIMARY is the default.

,EAX=*eax-value*
specifies the extended authorization index (EAX) that the stacking PC routine uses. Specify an EAX that is owned by the home address space of the issuer of the ETCRE macro. An EAX of X'0000' means the PC routine is not EAX-authorized. If EAX is not specified, the PC routine has the same EAX as the issuer of the PC instruction. EAX is not valid with PC=BASIC.

,EK = *entry-key*
specifies the PSW key (0 through 15) that the PC routine will run in. EK is not valid with PC = BASIC.

,EKM = *key-list*
specifies a list of keys (0 through 15) or key ranges, optionally enclosed in parentheses, that identify the entry key mask (EKM). When the PC routine is invoked, the keys specified identify either the additional keys that are to be ORed into the PKM (if PKM = OR is also specified or taken as the default) or the keys that should replace the PKM (if PKM = REPLACE is specified). EKM is required when you specify PKM = REPLACE.

,PARM1 = *user-parm1*
specifies the address or character string to be placed in the first word of the latent parameter area associated with this ETD entry.

Addressability to the latent parameter area is through the current primary address space. The latent parameter address is set in general register 4 as a result of the PC instruction, although AR4 is unchanged by the PC instruction. If the PC routine runs in AR mode, set the access register corresponding to the latent parameter area to 0 before the PC routine attempts to use it.

,PARM2 = *user-parm2*
specifies the address or character string to be placed in the second word of the latent parameter area associated with this ETD entry.

Addressability to the latent parameter area is through the current primary address space. The latent parameter address is set in general register 4 as a result of the PC instruction, although AR4 is unchanged by the PC instruction. If the PC routine runs in AR mode, set the access register corresponding to the latent parameter area to 0 before the PC routine attempts to use it.

,PROGRAM = *pgm-name*
,ROUTINE = *rtn-address*
specifies the PC routine. When you specify PROGRAM, the PC routine must be on the active LPA queue (FLPA or MLPA) or be in the PLPA or nucleus. The same restriction applies also to ROUTINE, unless this is a space-switching PC or the PC is to be used only in the address space that established it. In other words, the PC routine for a space-switching PC can reside in the private area of the address space in which it will run, but the ROUTINE parameter must be used to specify it.

On TYPE = ENTRY or TYPE = SET,ETEADR, either PROGRAM or ROUTINE is required.

,PC = STACKING
,PC = BASIC
indicates that this is a stacking PC (STACKING) or not a stacking PC (BASIC). Some parameters apply only to a stacking PC. STACKING is the default.

,PKM = OR
,PKM = REPLACE
indicates either that the entry key mask (EKM) is ORed with the PSW key mask (PKM) or replaces the current PKM. PKM = REPLACE is not valid with PC = BASIC. PKM = OR is the default.

,RAMODE = 31
,RAMODE = 24
specifies the AMODE of the routine specified on the ROUTINE parameter. RAMODE is valid only with ROUTINE. RAMODE = 31 is the default.

,SASN = OLD
,SASN = NEW
specifies whether the stacking PC routine will execute with SASN equal to the caller's PASN (SASN = OLD), or with SASN equal to the PASN of the stacking PC routine (SASN = NEW). SASN = NEW is not valid with PC = BASIC. SASN = OLD is the default.

,SSWITCH = NO
,SSWITCH = YES
specifies whether or not the PC routine switches address spaces. If SSWITCH = NO is specified, the PC does not switch address spaces. If SSWITCH = YES is specified, the PC routine will execute in the address space of the creator of the entry table with the authority of that address space. SSWITCH = NO is the default.

,STATE = PROBLEM
,STATE = SUPERVISOR
specifies which state the PC routine will receive control in either problem state (PROBLEM) or supervisor state (SUPERVISOR). The default is STATE = PROBLEM.

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

An example of using the ETDEF macro follows the description of the TYPE = SET parameter.

TYPE = SET Parameter

The ETDEF macro with the SET parameter works similar to the execute form of a macro *with this important distinction:* **the TYPE = SET form totally replaces an ETD entry and takes default values for ALL parameters you omit.** The normal execute form of a macro changes ***only*** the values you specify. SET is described as follows:

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

	Valid parameters (Required parameters are underlined)
TYPE = SET,ETEADR = *entry-addr*	PROGRAM or ROUTINE, AKM, EKM, ARR, ASCMODE, EAX, EK, PARM1, PARM2, PC, PKM, RAMODE, SASN, SSWITCH, STATE, RELATED, ASYNCH, CANCEL *entry-addr*: RX-type address or register (1) - (15).
TYPE = SET,HEADER = *header-addr*	NUMETE, RELATED *header-addr*: RX-type address or register (1) - (15).
,AKM = *key-list*	*key-list:* List of keys or key ranges where a key is a decimal digit 0 - 15.
,ARR = *arr*	*arr:* A-type address, register (2)-(12), or alphanumeric character string, enclosed by single quotation marks.
,ASYNCH = YES ,ASYNCH = NO	**Default:** ASYNCH = YES Valid only when ARR is also coded.
,CANCEL = YES ,CANCEL = NO	**Default:** CANCEL = YES Valid only when ARR is also coded.
,ASCMODE = PRIMARY ,ASCMODE = AR	**Default:** ASCMODE = PRIMARY
,EAX = *eax-value*	*eax-value:* Half-word decimal digit or register (2)-(12)
,EK = *entry-key*	*entry-key:* Decimal digit 0 - 15.
,EKM = *key-list*	*key-list:* List of keys or key ranges where a key is a decimal digit 0 -15. **Note:** EKM is required with PKM = REPLACE.

,NUMETE = *nbr-of-entries*	*nbr-of-entries*: Symbol, decimal number, or register (2)-(12). **Note**: NUMETE is required with HEADER.
,PARM1 = *user-parm1*	*user-parm1:* A-type address, register (2)-(12), or string of up to 4 characters enclosed by single quotation marks.
,PARM2 = *user-parm2*	*user-parm2:* A-type address, register (2)-(12), or string of up to 4 characters enclosed by single quotation marks.
,PC = STACKING ,PC = BASIC	**Default:** PC = STACKING
,PROGRAM = *pgm-name*	*pgm name:* String of up to 8 alphanumeric characters, optionally enclosed by single quotation marks.
,ROUTINE = *rtn-addr*	*rtn addr:* A-type address or registers (2)-(12)
,PKM = OR ,PKM = REPLACE	**Default:** PKM = OR
,RAMODE = 31 ,RAMODE = 24	**Default:** RAMODE = 31
,RELATED = *value*	*value:* Any valid macro parameter specification.
,SASN = OLD ,SASN = NEW	**Default:** SASN = OLD
,SSWITCH = NO ,SSWITCH = YES	**Default:** SSWITCH = NO
,STATE = PROBLEM ,STATE = SUPERVISOR	**Default:** STATE = PROBLEM

TYPE = SET,ETEADR = *entry-addr*

specifies the address of the ETD entry. ETDEF TYPE = SET,ETEADR sets the **entire** ETD entry that you generated through ETDEF TYPE = ENTRY macro. ETDEF TYPE = SET,ETEADR will set the ETD entry to the parameters you specify **and to the *defaults* on all parameters you omit.** That is, the system uses the default value, not the existing value, for any parameter that you omit.

TYPE = SET,HEADER = *header-addr*

changes the size of the ETD. Use TYPE = SET,HEADER to decrease the size of the ETD from the size you originally established on ETDEF TYPE = INITIAL.

,NUMETE = *nbr-of-entries*

specifies the number of contiguous entries in the ETD. *nbr-of-entries* is a decimal value from 1 to 128. NUMETE is required with the HEADER parameter. Use it to specify the number of entries you will use. It does not change the physical size of the table.

Example: Define an entry table that has three entries. The PC routine called PCPGM receives control from a program with PSW key authorization of 8, the PC routine named OTHERTN receives control from a program with PSW authorization keys of 0 through 15, and the third PC routine called PCRTN receives control in PSW authorization key 0. The fourth ETDEF is there to show that the number of entries can be changed with ETDEF SET. (Perhaps, due to some input parameter, only a subset of all possible PC routines are set up. On another invocation of the program, perhaps all entries would be used.) The entries use all defaults other than those on the AKM parameter.

```
MYPGM   CSECT
        LOAD  EP='PCPGM'
        LR    2,0
        ETDEF TYPE=SET,HEADER=MYETDS,NUMETE=3
        ETDEF TYPE=SET,ETEADR=FIRST,ROUTINE=(2),AKM=8
        ETCRE ENTRIES=MYETDS
        RETURN
         .
         .
         .
*            DATA DEFINITIONS FOR PROGRAM
         .
MYETDS  ETDEF TYPE=INITIAL
FIRST   ETDEF TYPE=ENTRY,ROUTINE=0,AKM=8
SECOND  ETDEF TYPE=ENTRY,PROGRAM=OTHERTN,AKM=0:15
THIRD   ETDEF TYPE=ENTRY,ROUTINE=PCRTN,AKM=0
FOURTH  ETDEF TYPE=ENTRY,ROUTINE=0,AKM=0
        ETDEF TYPE=FINAL
*
*
PCRTN   DS    0H
         .
         .
*     PC ROUTINE CODE
         .
         .
        END   MYPGM
```

Note that the combination of TYPE = INITIAL, ENTRY, and FINAL is essentially the list form of the macro and TYPE = SET is the execute form.

ETDES — Destroy Entry Table

The ETDES macro destroys a previously-created entry table. Only the address space that owns the entry table can destroy it. At the time ETDES is issued, the entry table must not be connected to any linkage tables unless PURGE=YES is coded. If any outstanding connections still exist and PURGE=YES is not coded, the entry table is not destroyed and the caller is abnormally terminated.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed to ETDES must also be addressable in primary mode at the time ETDES is issued.

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

The ETDES macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDES.
ETDES	
ƀ	One or more blanks must follow ETDES.
TOKEN=*addr*	*addr:* RX-type address or register (0) - (12).
,PURGE=NO ,PURGE=YES	**Default:** PURGE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

TOKEN=*addr*
specifies the address of the fullword token (returned by the ETCRE macro) associated with the entry table to be destroyed.

,PURGE=NO
,PURGE=YES
specifies whether (YES) or not (NO) the entry table is to be disconnected from all linkage tables and then destroyed.

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

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

Hexadecimal Code	Meaning
0	The specified entry table was destroyed. There were no connections to linkage indexes.
4	The specified entry table was destroyed. There were connections to linkage indexes, PURGE=YES was specified, and the entry table was disconnected.

ETDES (List Form)

The list form of the ETDES macro constructs a non-executable parameter list. The execute form of the macro can refer to this parameter list, or a copy of it for reentrant programs.

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

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

TOKEN=*addr*	*addr:* A-type address.
,PURGE=NO ,PURGE=YES	**Default:** PURGE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the ETDES macro.

ETDES (Execute Form)

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

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

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

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

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

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

ETDIS — Disconnect Entry Table

The ETDIS macro disconnects one or more entry tables from the home address space's linkage table.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed by the requestor must also be addressable in primary mode at the time the macro is issued.

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

The ETDIS macro is written as follows:

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

b                               One or more blanks must precede ETDIS.

ETDIS

b                               One or more blanks must follow ETDIS.

TKLIST=addr                     addr: RX-type address or register (0) - (12).

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

The parameters are explained as follows:

TKLIST = *addr*
specifies the address of a list of 1 to 32 fullword tokens, returned by the ETCRE macro, identifying the entry tables to be disconnected from the home address space's linkage table. The first entry of the list must be a fullword count of the number of tokens (1 to 32) in the list.

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

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The entry table is successfully disconnected.

EVENTS — Wait for One or More Events to Complete

The EVENTS macro is a functional specialization of the WAIT ECBLIST= macro facility with the advantages of notifying the program that events have completed and the order in which they completed.

The macro performs the following functions:

- Creates and deletes EVENTS tables.
- Initializes and maintains a list of completed event control blocks.
- Provides for single or multiple ECB processing.

The description of the EVENTS macro follows. The EVENTS macro is also described in *Application Development Macro Reference* with the exception of the BRANCH=YES parameter. This parameter is restricted to programs that run in supervisor state, key 0, and own the LOCAL lock.

Note: CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

For information about how to use this macro on an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.

The EVENTS macro is written as follows:

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

ENTRIES=*n* ENTRIES=*addr*	*n:* decimal digits 1-32767 *addr:* register (2) - (12).
ENTRIES=DEL,TABLE=*tab addr* TABLE=*tab addr*	*tab addr:* symbol, RX-type address, or register (2) - (12). **Note:** If the ENTRIES parameter is specified as indicated in the first two formats, no other parameters may be specified.
,ECB=*ecb addr* ,LAST=*last addr*	*ecb addr:* symbol, RX-type address, or register (2) - (12). *last addr:* symbol, RX-type address, or register (2) - (12). **Note:** If LAST is specified, WAIT must also be specified.
,WAIT=YES ,WAIT=NO	
,BRANCH=NO ,BRANCH=YES	**Default:** BRANCH=NO

The parameters are explained below:

ENTRIES=*n*
ENTRIES=*addr*
specifies either a register or a decimal number from 1 to 32,767 which specifies the maximum number of completed ECB addresses that can be processed in an EVENTS table concurrently.

Note: When this parameter is specified, no other parameter should be specified.

ENTRIES = DEL,TABLE = *tab addr*
specifies that the EVENTS table whose address is specified by TABLE = *tab addr* is to be deleted. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

Notes:

1. When this parameter is specified, no other parameter should be specified.
2. TABLE resides in 24-bit addressable storage.

TABLE = *tab addr*
specifies either a register number or the address of a word containing the address of the EVENTS table associated with the request. The address specified with the operand TABLE must be that of an EVENTS table created by this task.

Note: TABLE resides in 24-bit addressable storage.

,WAIT = NO
,WAIT = YES
specifies whether or not to put the issuing program in a wait state when there are no completed events in the EVENTS TABLE (specified by the TABLE = parameter).

,ECB = *ecb addr*
specifies either a register number or the address of a word containing the address of an event control block. The EVENTS macro should be used to initialize any event-type ECB. To avoid the accidental destruction of bit settings by a system service such as an access method, the ECB should be initialized after the system service that will post the ECB has been initiated (thus making the ECB eligible for posting) and before the EVENTS macro is issued to wait on the EVENTS table.

Notes:

1. Register 1 should not be specified for the ECB address.
2. This parameter may not be specified with the LAST = parameter.
3. The ECB can reside above or below 16 megabytes.
4. If only ECB initialization is being requested, neither WAIT = NO nor WAIT = YES should be specified, to prevent any unnecessary WAIT processing from occurring.

,LAST = *last addr*
specifies either a register number or the address of a word containing the address of the last EVENT parameter list entry processed.

Notes:

1. Register 1 should not be specified for the LAST address.
2. This parameter should not be specified with the ECB = parameter.
3. The WAIT macro must also be specified.
4. LAST resides in 24-bit addressable storage.

,BRANCH = NO
,BRANCH = YES
specifies that an SVC entry (BRANCH = NO) or a branch entry (BRANCH = YES) is to be performed.

Example 1

The following shows total processing via EVENTS

EVENTS and ECB Initialization

```
        EVENTS    ENTRIES=1000

        ST        R1,TABADD

        WRITE     ECBA

        LA        R2,ECBA...

        EVENTS    TABLE=TABADD,ECB=(R2)
```

Parameter List Processing

```
        EVENTS    TABLE=TABADD,WAIT=YES

        LR        R3,R1          PARMLIST ADDR
        B         LOOP2          GO TO PROCESS ECB
LOOP1   EVENTS    TABLE=TABADD,WAIT=YES,LAST=(R3)
        LR        R3,R1          SAVE POINTER
LOOP2   EQU       *              PROCESS COMPLETED EVENTS

        TM        0(R3),X'80'    TEST FOR MORE EVENTS
        BO        LOOP1          IF NONE, GO WAIT
        LA        R3,4(,R3)      GET NEXT ENTRY
        B         LOOP2          GO PROCESS NEXT ENTRY
```

Deleting EVENTS Table

```
        EVENTS    TABLE=TABADD,ENTRIES=DEL

TABADD  DS        F
```

Example 2

Processing One ECB at a Time.

```
                  EVENTS    ENTRIES=10
                  ST        1,TABLE

        NEXTREC   GET       TPDATA,KEY
                  ENQ       (RESOURCE,ELEMENT,E,,SYSTEM)
                  READ      DECBRW,KU,,'S',MF=E
                  LA        3,DECBRW
                  EVENTS    TABLE=TABLE,ECB=(3),WAIT=YES

                  WRITE     DECBRW,K,MF=E
                  LA        3,DECBRW
        RETEST    EVENTS    TABLE=TABLE,ECB=(3),WAIT=NO
                  LTR       1,1
                  BNZ       NEXTREC

                  B         RETEST

        TABLE     DS        F
```

EXTRACT — Extract TCB Information

The EXTRACT macro causes the control program to provide information from specified fields of the task control block or a subsidiary control block for either the active task or one of its subtasks. The control program places the information in an area that the problem program provides. For a description of this area see "Providing an EXTRACT Answer Area" in *SPL: Application Development Guide.* When EXTRACT is issued, its parameter list can reside in 24- or 31-bit addressable storage.

Notes:

Product-Sensitive Programming Interface

1. If the EXTRACT macro is used to obtain the TIOT in order to find the UCB, it is the user's responsibility to ensure that the TIOT contains the UCB address. To find the UCB address, see the topic "Finding the UCB Address" in *SPL: Application Development Guide.*

End of Product-Sensitive Programming Interface

2. Programs that reside in 24- and 31-bit addressable storage can issue the standard form of the macro.

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

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

answer addr	*answer addr:* A-type address, or register (2) - (12).
,'S'. ,*tcb addr*	*tcb addr:* A-type address, or register (2) - (12). **Default:** 'S'.
,FIELDS=*(tcb info)*	*tcb info:* any combination of the following, separated by commas: ALL PRI GRS CMC FRS TIOT AETX COMM TSO PSB TJID ASID

The parameters are explained as follows:

answer addr
specifies the address of the answer area to contain the requested information. The area is one or more fullwords, starting on a fullword boundary. The number of fullwords must be the same as the number of fields specified in the FIELDS parameter, unless ALL is coded. If ALL is coded, seven fullwords are required.

,'S'.
,tcb addr
specifies the address of a fullword on a fullword boundary containing the address of a task control block for a subtask of the active task. If 'S'. is coded or is the default, no address is specified and the active task is assumed.

,FIELDS = *(tcb info)*

specifies the task control block information requested:

ALL requests information from the GRS, FRS, reserved, AETX, PRI, CMC, and TIOT fields. (If ALL is specified, 7 words are required just for ALL.)

GRS is the address of the save area used by the control program to save the general registers 0-15 when the task is not active.

FRS is the address of the save area used by the control program to save the floating point registers 0, 2, 4, and 6 when the task is not active.

AETX is the address of the end of task exit routine specified in the ETXR parameter of the ATTACH macro used to create the task.

PRI is the current limit (third byte) and dispatching (fourth byte) priorities of the task. The two high-order bytes are set to zero.

CMC is the task completion code. If the task is not complete, the field is set to zero.

TIOT is the address of the task input/output table.

COMM is the address of the command scheduler communications list. The list consists of a pointer to the communications event control block and a pointer to the command input buffer, and a token. (If a token exists, the high order bit of the token field is set to one). The token is used only with internal START commands. See "Issuing an Internal START or REPLY Command" in *SPL: Application Development Guide*.

TSO is the address of a byte in which a high bit of 1 indicates a TSO address space, and a high bit of 0 indicates a non-TSO address space.

PSB is the address of the TSO protected step block.

TJID is the address space identifier (ASID) for a TSO address space, and zero for a non-TSO address space.

ASID is the address space identifier.

Example 1

Operation: Provide information from all the fields of the indicated TCB except ASID. WHERE is the label of the answer area, ADDRESS is the label of a fullword that contains the address of the subtask TCB for which information is to be extracted.

```
EXTRACT WHERE,ADDRESS,FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 2

Operation: Provide information from the current TCB, as above.

```
EXTRACT WHERE,'S',FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 3

Operation: Provide information from the command scheduler communications list. ANSWER is the label of the answer area and TCBADDR is the label of a fullword that contains the address of the subtask TCB from which information is to be extracted.

```
EXTRACT ANSWER,TCBADDR,FIELDS=(COMM)
```

EXTRACT (List Form)

The list form of the EXTRACT macro is used to construct a remote control program parameter list.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede EXTRACT.
EXTRACT	
b	One or more blanks must follow EXTRACT.
answer addr	*answer addr:* A-type address.
,'S'. *,tcb addr*	*tcb addr:* A-type address. **Default:** 'S'.
,FIELDS=*(tcb info)*	tcb info: any combination of the following, separated by commas: ALL PRI GRS CMC FRS TIOT AETX COMM TSO PSB TJID ASID
,MF=L	

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

,MF=L
specifies the list form of the EXTRACT macro.

EXTRACT (Execute Form)

The execute form of the EXTRACT macro uses, and can modify, a remote control program parameter list. If the FIELDS parameter, restricted in use, is coded in the execute form, any TCB information specified in a previous FIELDS parameter is canceled and must be respecified if required for this execution of the macro.

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

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

answer addr	*answer addr:* RX-type address, or register (2) - (12).
,'S'. *,tcb addr*	*tcb addr:* RX-type address, or register (2) - (12).
,FIELDS=*(tcb info)*	tcb info: any combination of the following, separated by commas: ALL PRI GRS CMC FRS TIOT AETX COMM TSO PSB TJID ASID
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl addr*)
specifies the execute form of the EXTRACT macro using a remote control program parameter list.

FESTAE — Fast Extended STAE

The FESTAE macro allows an SVC to establish an ESTAE recovery routine with minimal overhead and no locking requirements. The ESTAE routine activated by FESTAE receives control in the same sequence and under the same conditions as though created by the ESTAE macro. The FESTAE macro can be issued in cross memory mode as long as the currently addressable address space is the home address space. For more information, see *SPL: Application Development Guide*.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode. FESTAE users executing in 31-bit addressing mode must recompile using the MVS/XA FESTAE macro expansion so that the exit routine gets control in 31-bit addressing mode.

For information about how to use this macro on an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.

The FESTAE macro expansion has no external linkage. The macro is written as follows:

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

0,WRKREG=*work reg addr* EXITADR=*exit addr*	*work reg addr:* Register (1) - (14). *exit addr:* Register (1) - (14).
,RBADDR=*svrb addr*	*svrb addr:* Register (1) - (14).
,TCBADDR=*tcb addr*	*tcb addr:* Register (1) - (14).
,PARAM=*list addr*	*list addr:* Register (1) - (14).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=NONE ,PURGE=HALT ,PURGE=QUIESCE	**Default:** PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Default:** TERM=NO
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,ERRET=*label*	*label:* any valid assembler name.

The parameters are explained as follows:

0,WRKREG=*work reg addr*
specifies that the current ESTAE routine be canceled if it was created by FESTAE. An error occurs if the current ESTAE routine was not created by FESTAE. A work register must be specified for use by the FESTAE macro expansion.

,EXITADR = *exit addr*
specifies a register that contains the address of an ESTAE routine to be entered if the task terminates abnormally. This register is used subsequently as a work register.

,RBADDR = *svrb addr*
specifies a register that contains the address of the current SVRB prefix. RBADDR must be specified if EXITADR has also been specified. The specified register is not altered.

,TCBADDR = *tcb addr*
specifies the register containing the current TCB address. This register is not altered, and its specification results in the generation of more efficient code.

Note: The TCB resides in storage below 16 megabytes.

,PARAM = *list addr*
specifies the register containing the address of a user-defined parameter list that contains data to be used by the ESTAE routine. The routine receives this address when it is scheduled for execution.

The use of this parameter list is optional, but the user should zero out any spurious data it might contain whether or not he intends to use it. If the user does not select the PARAM option, the routine receives instead the 24-byte parameter area in the SVRB. In this case, the user must locate this SVRB parameter area and initialize it with appropriate data.

,ERRET = *label*
specifies a label within the CSECT issuing the FESTAE for which addressability has been established. The FESTAE macro branches to this label if it is returning a code other than zero. This option saves the user the instructions necessary to check the return code. If the user does not specify the ERRET option, control returns instead to the instruction immediately following the FESTAE macro. The return code is in register 15.

All the other FESTAE parameters have the same meaning as their ESTAE counterparts.

Upon conclusion of FESTAE processing, control resumes at the instruction following the FESTAE macro. Register 15 then contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of the FESTAE request.
08	A previous create has been issued with FESTAE for this SVRB; the request has been ignored.
0C	Cancel has been specified under one of the following conditions: 1. There is no exit for this TCB. 2. The most recent exit is not owned by the caller. 3. The most recent exit was not created by FESTAE.

Example

Operation: In case of an abnormal termination, execute the ESTAE routine specified by register 2, allow asynchronous processing, do not allow special error processing, default to PURGE=NONE, and pass the parameter list pointed to by register 7 to the ESTAE routine.

```
FESTAE  EXITADR=(REG2),RBADDR=(REG3),TCBADDR=(REG6),    X
        PARAM=(REG7),ASYNCH=YES,TERM=NO
```

FRACHECK - Check User's Authorization (for RACF Release 1.8.1 or earlier)

Note: The RACROUTE macro is the preferred programming interface.

This macro description applies to RACF Release 1.8.1 or earlier. Your program can invoke the FRACHECK macro directly; however, IBM recommends that you invoke the equivalent function through the RACROUTE macro, using the REQUEST = FASTAUTH parameter. See "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 439 for the applicable RACROUTE macro description.

If you have RACF Release 1.9 installed on your system, you can still invoke the FRACHECK macro directly; however, to use the new Release 1.9 functions, you must use the RACROUTE macro and specify REQUEST = FASTAUTH. See the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST = FASTAUTH:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 449**
- **"RACROUTE REQUEST = FASTAUTH — Verifies Access to Resources (for RACF Release 1.9)" on page 539.**

The FRACHECK macro checks a user's authorization for access to a resource. FRACHECK verifies access to those resources whose RACF profiles have been brought into main storage by the RACLIST facility. FRACHECK is a branch entered service that does not save registers upon entry. FRACHECK uses but does not restore registers 0-5, 14, and 15 FRACHECK does not alter registers 6-13.

CAUTION:
The FRACHECK macro executes in the addressing mode of the caller. Therefore, to access profiles that reside above 16 megabytes, the program that issues FRACHECK must be running in 31-bit addressing mode when it issues FRACHECK.

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

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

ENTITY=*entity addr*	*entity addr:* A-type address or register (2) - (12).
,CLASS='*classname*' ,CLASS=*classname addr*	*classname:* DASDVOL or TAPEVOL. *classname addr:* A-type address or register (2) - (12).
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* registers (2) - (12). **Default:** ATTR=READ
,ACEE=*acee addr*	*acee addr:* A-type address or register (2) - (12).
,WKAREA=*area addr*	*area addr:* A-type address or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address or register (2) - (12).
,INSTLN=*parm list addr*	*parm list addr:* A-type address or register (2) - (12).
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6

The parameters are explained as follows:

ENTITY =*entity addr*
specifies that RACF authorization checking is to be performed for the resource whose name is pointed to by the specified address. The resource name is a 6-byte volume serial number for CLASS='DASDVOL' or CLASS='TAPEVOL'. The name must be left justified and padded with blanks. The length of all other resource names is determined from the class descriptor tables.

,CLASS ='*classname*'
,CLASS =*classname addr*
specifies that RACF authorization checking is to be performed for a resource of the specified class. If you specify an address, the address must point to an 8-byte field containing the classname.

,ATTR =READ
,ATTR =UPDATE
,ATTR =CONTROL
,ATTR =ALTER
,ATTR =(*reg***)**
specifies the access authority required by the user or group accessing the resource:

READ — RACF user or group can open the resource only to read.

UPDATE — RACF user or group can open the resource to read or write.

CONTROL — For VSAM data sets, RACF user or group has authority equivalent to the VSAM control password. For non-VSAM data sets and other resources, RACF user or group has UPDATE authority.

ALTER — RACF user or group has total control over the resource.

If you specify a register, the register must contain one of the following codes in the low-order byte of the register:

X'02' – READ
X'04' – UPDATE
X'08' – CONTROL
X'80' – ALTER

,ACEE =*acee addr*
specifies the address of the accessor control environment element (ACEE) to be used to check authorization and to locate the in-storage profiles (RACLIST output) for the specified classes. If you specify an ACEE, it is used for authorization checking. If the specified ACEE has an in-storage profile list for the specified class, it is used to locate the resource. If you do not specify an ACEE or if there is no in-storage profile list for the specified class in the ACEE, RACF uses the TASK ACEE (TCBSENV) pointer in the extended TCB. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE for the address space to obtain the list of the in-storage profiles. The ASXBSENV field of the address space extension block points to the main ACEE.

,WKAREA =*area addr*
specifies the address of a 16 word work area that FRACHECK will use which contains the following information:

Word 12 contains the reason code that ICHRFC00 will pass back to the FRACHECK caller via Register 0.

Word 13 contains the return code that FRACHECK passes back to the caller in register 15.

Word 14 contains the address of the in-storage profile used to determine authorization, or zero if no profile was found.

Word 15 contains a value provided by a pre-processing installation exit, or zero if there was no pre-processing exit. This will be passed back to the caller in register 1.

,APPL ='*applname*'
,APPL =*applname addr*
specifies the name of the application requesting the authorization checking. This information is not used for the authorization checking process but is made available to the installation exit(s). If you specify an address, the address should point to an 8-byte area containing the application name, left justified and padded with blanks, if necessary.

,INSTLN =*parm list addr*
specifies the address of an area that contains information for the FRACHECK installation exit. This address is passed to the exit routine when the exit is given control. Application or installation programs use the INSTLN parameter to pass information to the FRACHECK installation exit.

,RELEASE =**1.6**|**1.7**|**1.8**|**1.8.1**
specifies the RACF release level of the parameter list that FRACHECK generates.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 10 on page 246.

When you specify the RELEASE parameter, checking is done at assembly time. To get execution-time validation of the compatibility between the list and execute forms of the FRACHECK macro, specify CHECK subparameter on the execute form of the macro.

Parameters for RELEASE=1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Figure 10. FRACHECK Parameters for RELEASE=1.6 and Later

Parameter	RELEASE=1.6	RELEASE=1.7	RELEASE=1.8 or 1.8.1
ACEE=	X	X	X
APPL=	X	X	X
ATTR=	X	X	X
CLASS=	X	X	X
ENTITY=	X	X	X
INSTLN=	X	X	X
RELEASE=	X	X	X
WKAREA=	X	X	X

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code:

Hexadecimal Code — **Meaning**

00 — The user or group is authorized to use the resource.

Reason Code	Meaning
0	The FRACHECK return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is not within the scope of the audit/global audit specification.
4	The FRACHECK return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is within the scope of the audit/global audit specification. The FRACHECK caller should log the attempt by issuing a RACHECK for the resource that the caller is attempting to access.

04 — The resource or classname is not defined to RACF.

08 — The user or group is not authorized to use the resource.

Reason Code	Meaning
0	The FRACHECK return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is not within the scope of the audit/global audit specification.
4	The FRACHECK return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is within the scope of the audit/global audit specification. The FRACHECK caller should log the attempt by issuing a RACHECK for the resource that the caller is attempting to access.

0C — RACF is not active.

10 — FRACHECK installation exit error occurred.

14 — RACF CVT does not exist (RACF is not installed or insufficient level of RACF is installed).

64 — Indicates that you specified the CHECK subparameter of the RELEASE parameter on the execute form of the FRACHECK macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

FRACHECK (List Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede FRACHECK.
FRACHECK	
b	One or more blanks must follow FRACHECK.
ENTITY=*entity addr*	*entity addr:* A-type address.
,CLASS='*classname*' ,CLASS=*classname addr*	*classname:* DASDVOL or TAPEVOL. *classname addr:* A-type address.
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER	**Default:** ATTR=READ
,ACEE=*acee addr*	*acee addr:* A-type address.
,WKAREA=*area addr*	*area addr:* A-type address.
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address.
,INSTLN=*parm list addr*	*parm list addr:* A-type address.
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,MF=L	

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

,MF =L
specifies the list form of the FRACHECK macro.

FRACHECK (Execute Form)

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

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

ENTITY=*entity addr*	*entity addr:* RX-type address or register (2) - (12).
,CLASS=*classname addr*	*classname addr:* RX-type address or register (2) - (12).
,ATTR=(*reg*)	*reg:* register (2) - (12).
,ACEE=*acee addr*	*acee addr:* RX-type address or register (2) - (12).
,WKAREA=*area addr*	*area addr:* RX-type address or register (2) - (12).
,APPL=*applname addr*	*applname addr:* RX-type address or register (2) - (12).
,INSTLN=*parm list addr*	*parm list addr:* RX-type address or register (2) - (12).
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address or register (1) - (12).

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

,MF =(E,*ctrl addr*)
specifies the execute form of the FRACHECK macro, using a remote control program parameter list.

,RELEASE =(*number*,CHECK)
,RELEASE =1.6|1.7|1.8|1.8.1
,RELEASE= (,CHECK)
specifies the RACF release level of the parameter list that FRACHECK generates.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 10.

When you specify the RELEASE parameter, checking is done at assembly time. To get execution-time validation of the compatibility between the list and execute forms of the FRACHECK macro, specify the CHECK subparameter on the execute form of the macro.

When you request CHECK processing and the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE parameter, the execute form of the macro will not be generated. Instead, FRACHECK generates a return code of X'64'.

FREEMAIN — Free Virtual Storage

The FREEMAIN macro releases one or more areas of virtual storage, or an entire virtual storage subpool, previously assigned to the active task as a result of a GETMAIN macro. FREEMAIN is supported in a cross memory environment.

The active task is abnormally terminated if the specified virtual storage does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not currently allocated to the active task. Register 15 is set to 0 to indicate successful completion. For a conditional FREEMAIN, register 15 is set to 4 if the specified area or subpool is not currently allocated to the active task.

In the parameters discussed below, EU, LU, and VU specify unconditional requests and result in the same processing as E, L, and V, respectively. The two formats for these requests are available to maintain compatibility with the GETMAIN formats. Users of the FREEMAIN macro who are freeing virtual storage with addresses greater than 16 megabytes must use either the RC or RU form of the macro. All addresses specified with the RC or RU form of the macro are treated as 31-bit addresses.

The FREEMAIN macro is available to callers in either primary address space control (ASC) mode or access register (AR) ASC mode. Callers in AR mode, however, are limited to requests to release global (common area) storage and must use the branch entry. Before invoking the FREEMAIN macro in AR mode, the caller must issue SYSSTATE ASCENV=AR to tell the system to generate FREEMAIN code that is appropriate for AR ASC mode.

The description of the FREEMAIN macro follows. The FREEMAIN macro is also described in *Application Development Macro Reference* with the exception of the BRANCH and KEY parameters. These parameters are restricted to programs running supervisor state, key 0 and, therefore, are only described here.

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

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

LC,LA=*length addr* LU,LA=*length addr* L,LA=*length addr* VC VU V EC,LV=*length value* EU,LV=*length value* E,LV=*length value* RC,LV=*length value* RC,SP=*subpool nmbr* RU,LV=*length value* RU,SP=*subpool nmbr* R,LV=*length value* R,SP=*subpool nmbr*	*length addr:* A-type address, or register (2) - (12). *length value:* symbol, decimal number, or register (2) - (12). If R is specified, register (0) may also be specified. *subpool nmbr:* symbol, decimal number, or register (0) or (2) - (12). **Notes:** 1. If R,SP=(0) is specified with no other parameters, the high order byte of register 0 must contain the subpool number and the low order 3 bytes must contain zero. 2. If RC,SP=*subpool numbr* or RU,SP=*subpool nmbr* or R,SP=*subpool nmbr* is specified, no other parameters except RELATED can be specified. 3. RC and RU are the only parameters that can be used to free storage above 16 megabytes.

,A=*addr*	*addr:* A-type address, or register (2) - (12). **Note:** If R, RC, or RU is coded, register (1) can also be specified.
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal number, or register (2) - (12). If R,SP=(0) is specified, the high order byte of register 0 must contain the subpool number and the low order 3 bytes must contain the length value.
,BRANCH=YES ,BRANCH=(YES,GLOBAL)	**Note:** BRANCH=(YES,GLOBAL) may be specified with RC or RU above. Also, the macro expansion uses register 4 for the address of the global save areas pointed to by the CVT. The previous contents of register 4 is overridden.
,KEY=*nmbr*	*nmbr:* decimal numbers 0-15, or register (2) - (12). **Note:** This parameter may be specified only with BRANCH and RC or RU above.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

LC,LA = *length addr*
LU,LA = *length addr*
L,LA = *length addr*
VC
VU
V
EC,LV = *length value*
EU,LV = *length value*
E,LV = *length value*
RC,LV = *length value*
RC,SP = *subpool nmbr*
RU,LV = *length value*
RU,SP = *subpool nmbr*
R,LV = *length value*
R,SP = *subpool nmbr*

specifies the type of FREEMAIN request:

LC, LU, and L indicate conditional (LC) and unconditional (LU and L) list requests and specify release of one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas must be provided in a corresponding list whose address is specified in the A parameter. All virtual storage areas must start on a doubleword boundary.

VC, VU, and V indicate conditional (VC) and unconditional (VU and V) variable requests and specify release of single areas of virtual storage. The address and length of the virtual storage area are provided at the address specified in the A parameter.

EC, EU, and E indicate conditional (EC) and unconditional (EU and E) element requests and specify release of single areas of virtual storage. The length of the single virtual storage area is indicated in the LV parameter. The address of the virtual storage area is provided at the address indicated in the A parameter.

RC, RU, and R indicate conditional (RC) and unconditional (RU and R) register requests and specify release of single areas of virtual storage from the subpool indicated, or specifies release of the entire subpool indicated. If the release is not for the entire subpool, the address of the virtual storage area is indicated in the A parameter. The length of the area is indicated in the LV parameter. The virtual storage area must start on a doubleword boundary.

Notes:

1. A conditional request indicates that the task is not to be abnormally terminated if the virtual storage being freed is not allocated to the active task. However some abends cannot be prevented. An unconditional request indicates that the task is to be abnormally terminated in this situation.
2. Callers in either 24-bit or 31-bit addressing mode can use RC or RU to free storage above 16 megabytes.
3. If the address of the area to be freed is greater than 16 megabytes, you must use RC or RU.

LA specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each virtual storage area to be released; the high-order bit in the last word must be set to 1 to indicate the end of the list. Each word must contain the required length in the low-order three bytes. The fullwords in this list must correspond with the fullwords in the associated list specified in the A parameter. The words must not be in the area to be released. If this rule is violated and if the words are the last allocated items on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an 0C4.

LV specifies the length, in bytes, of the virtual storage area being released. The value should be a multiple of 8; if it is not, the control program uses the next high multiple of 8. If R is coded, LV=(0) may be designated; the high-order byte of register 0 must contain the subpool number, and the low-order three bytes must contain the length (in this case, the SP parameter is invalid).

,A=*addr*
specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. The input must not reside within the area to be released. If this rule is violated and if the input is within the area and is the last allocated item on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an 0C4.

- If E, EC, EU, R, RC, or RU is designated, one word, which contains the address of the virtual storage area to be released, is required.
- If V, VC, or VU is coded, two words are required; the first word contains the address of the virtual storage area to be released, and the second word contains the length of the area to be released.
- If L, LC, or LU is coded, one word is required for each virtual storage area to be released; each word contains the address of one virtual storage area.
- If R, RC, or RU is coded, any of the registers 1 through 12 can be designated, in which case the address of the virtual storage area, not the address of the fullword, must have previously been loaded into the register.

,SP=*subpool nmbr*
specifies the subpool number of the virtual area to be released. The subpool number can be between 0 and 255. The SP parameter is optional and if omitted, subpool 0 is assumed. If R is coded, SP=(0) can be designated, in which case the subpool number must be previously loaded into the low-order byte of register 0.

For subpool freemains, the SP parameter specifies the number of the subpool to be released. Subpool freemains can be issued only for the following subpools: 1-127, 203, 204, 213, 214, 223, 224, 229, 230, 233, 236, 237, 240, and 250-253; and if the caller is in key 0, supervisor state. Any attempt to issue a subpool freemain for any other subpool causes a 478 or 40A abend. (See *SPL: Application Development Guide* for a list of the characteristics of the valid subpools.) If R,SP=(0) is specified with no other parameters, the high-order byte of register 0 must contain the subpool number and the low-order 3 bytes must contain zero.

Note: Callers executing in supervisor state and key zero, who specify subpool 0, will free storage from subpool 252. Therefore, when requesting a dump of this storage via the SDUMP macro, they must specify subpool 252 rather than 0.

,BRANCH=YES
,BRANCH=(YES,GLOBAL)

specifies that a branch entry is to be used. If (YES,GLOBAL) is specified, the entry point to service global storage requests without the need for the local address space lock will be used. The caller must be disabled.

If BRANCH=YES is specified, the caller must pre-load register 4 with the TCB address, pre-load register 7 with the ASCB address, and hold the local address space lock of the ASCB address specified in register 7 prior to entry. Register 3 will be destroyed if RC or RU was specified.

Callers in cross memory mode can use the BRANCH=YES parameter of the FREEMAIN macro. If the caller is in cross memory mode, the storage is freed in the currently addressable address space. The caller must hold the CML lock for the currently addressable address space; load register 7 with the address of the ASCB of the currently addressable address space; and load register 4 with zero or the address of a TCB in the currently addressable address space. If register 4 contains a zero, the storage that is freed is associated with the current job step task that owns the cross memory resources in the currently addressable address space (that is, the TCB anchored in ASCBXTCB).

If BRANCH=(YES,GLOBAL) is specified, registers 4 and 7 need not contain the TCB and ASCB addresses; and registers 3 and 4 are changed when control is returned to the caller. Additionally, the SP parameter may only designate subpools 226, 227, 228, 231, 239, 241, 245, 247, or 248.

,KEY=*key nmbr*

specifies the key (in bits 24-27 of the register) in which the requested storage was obtained. This parameter applies to subpools 227, 228, 229, 230, 231, and 241, and allows both global and local storage to be freed in the specified storage protection key.

,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.

When control is returned, register 15 contains one of the following return codes. A code other than 0 is possible only for conditional forms.

Hexadecimal Code	Meaning
00	Virtual storage was freed.
04	Not all virtual storage was freed.
08	Part of area being freed is still fixed. This condition usually causes an A78, A05, or A0A abend.
0C	Page table is paged out.

Example 1

Operation: Free 400 bytes of storage addressed by register 2 via a branch entry. If the storage is successfully freed, register 15 contains 0; otherwise, register 15 contains a nonzero value.

```
FREEMAIN EC,LV=400,A=(2),BRANCH=YES
```

Example 2

Operation: Free 48 bytes of the storage (addressed by register 5) in subpool 231. Register 3 has been preset to contain the storage key of the storage to be released. If the request is unsuccessful, the caller is abnormally terminated.

```
FREEMAIN RU,LV=48,A=(5),SP=231,KEY=(3),BRANCH=(YES,GLOBAL)
```

FREEMAIN (List Form)

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

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

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

LC LU L VC VU V EC EU E	
,LA = *length addr* ,LV = *length value*	*length addr:* A-type address. *length value:* symbol or decimal number. **Notes:** 1. LA may only be specified with LC, LU, or L above. **2.** LV may only be specified with EC, EU, or E above.
,A = *addr*	*addr:* A-type address.
,SP = *subpool nmbr*	*subpool nmbr:* symbol or decimal number.
,RELATED = *value*	*value:* any valid macro keyword specification.
,MF = L	

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

,MF = L
specifies the list form of the FREEMAIN macro.

FREEMAIN (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the FREEMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN.

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

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

LC LU L VC VU V EC EU E	
,LA=*length addr* ,LV=*length value*	*length addr:* RX-type address or register (2) - (12). *length value:* symbol, decimal number, or register (2) - (12). **Notes:** 1. LA may only be specified with LC, LU, or L above. 2. LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr:* RX-type address, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal number, or register (0) or (2) - (12).
,BRANCH=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*ctrl prog*)	*ctrl prog:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl prog*)
specifies the execute form of the FREEMAIN macro using a remote control program parameter list.

GETMAIN — Allocate Virtual Storage

The GETMAIN macro requests the control program to allocate one or more areas of virtual storage to the active task. For task related subpools, the virtual storage areas are allocated from the specified subpool in the virtual storage area assigned to the associated job step. The virtual storage areas each begin on a doubleword or page boundary and are not cleared to 0 when allocated. The total of the lengths specified must not exceed the length available. For most subpools, the storage will be released when the task assigned ownership terminates, or through the use of the FREEMAIN or STORAGE RELEASE macro. For information on the use of the GETMAIN macro, see *SPL: Application Development Guide*.

The options R, LC, LU, VC, VU, EC, or EU can be used by callers in either 24-bit or 31-bit addressing mode. If one of these options is specified, storage area addresses and lengths will be treated as 24-bit addresses and values. The parameter list addresses and the pointers to the length and address lists in the parameter lists (if present) will be treated as 31-bit addresses if the caller's addressing mode is 31-bit; otherwise, they will be treated as 24-bit addresses.

The options RU, RC, VRU, and VRC can be used by callers in either 24-bit or 31-bit addressing mode. However, all values and addresses will be treated as 31-bit values and addresses. The GETMAIN macro is also described in *Application Development Macro Reference* with the exception of the BRANCH and KEY parameters. These parameters are restricted in use to programs running supervisor state and key 0.

The GETMAIN macro is available to callers in either primary address space control (ASC) mode or access register (AR) ASC mode. Callers in AR mode, however, are limited to requests to obtain global (common area) storage and must use the branch entry. Before issuing the GETMAIN macro in AR mode, issue SYSSTATE ASCENV=AR to tell the system to generate GETMAIN code that is appropriate for AR mode.

The description of the GETMAIN macro follows.

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

LC,LA=*length addr*,A=*addr* LU,LA=*length addr*,A=*addr* VC,LA=*length addr*,A=*addr* VU,LA=*length addr*,A=*addr* EC,LV=*length value*,A=*addr* EU,LV=*length value*,A=*addr* RC,LV=*length value* RU,LV=*length value* R,LV=*length value*	*length addr:* A-type address, or register (2) - (12). *length value:* symbol, decimal number, or register (2) - (12). If RC or RU is specified, register (0) may also be specified. *addr:* A-type address or register (2) - (12). **Note:** RC, RU, VRC, or VRU must be used for address greater than 16 megabytes.
VRC,LV=*(maximum length value, minimum length value)* VRU,LV=*(maximum length value, minimum length value)*	*maximum length value:* symbol, decimal number, or register (2) - (12). *minimum length value:* symbol decimal number, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal number, or register (2) - (12). **Note:** If R,LV=(0) is specified above, SP may not be specified.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD **Note:** This parameter may not be specified with R above.

,BRANCH=YES ,BRANCH=(YES,GLOBAL)	**Note:** BRANCH=(YES,GLOBAL) may only be specified with RC, RU, VRC, or VRU above. Also, the macro expansion uses register 4 for the address of the global save area pointed to by the CVT. The previous contents of register 4 is overridden. The macro expansion also uses register 3.
,KEY=*key number*	*key number:* decimal numbers 0-15, or register (2) - (12). **Default:** KEY=0 **Note:** This parameter may be specified only with BRANCH and RC, RU, VRC, or VRU; and subpools 226, 227, 228, 229, 230, 231, and 241.
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=(ANY) ,LOC=(ANY,ANY) ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES **Note:** This parameter can only be used with RC, RU VRC, or VRU. On all other forms, LOC=BELOW is used.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

LC,LA = *length addr*, **A** = *addr*
LU,LA = *length addr*, **A** = *addr*
VC,LA = *length addr*, **A** = *addr*
VU,LA = *length addr*, **A** = *addr*
EC,LV = *length value*, **A** = *addr*
EU,LV = *length value*, **A** = *addr*
RC,LV = *length value*
RU,LV = *length value*
R,LV = *length value*
VRC,LV = *(maximum length value,minimum length value)*
VRU,LV = *(maximum length value,minimum length value)*
specifies the type of GETMAIN request:

LC and LU indicate conditional (LC) and unconditional (LU) list requests, and specify requests for one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas is returned in a list beginning at the address specified in the A parameter. No virtual storage is allocated unless all of the requests in the list can be satisfied.

VC and VU indicate conditional (VC) and unconditional (VU) variable requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is between the two values at the address specified in the LA parameter. The address and actual length of the allocated virtual storage area are returned by the control program at the address indicated in the A parameter.

EC and EU indicate conditional (EC) and unconditional (EU) element requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is indicated by the parameter, LV=*length value*. The address of the allocated virtual storage area is returned at the address indicated in the A parameter. RU and R indicate unconditional register requests. RC, RU, and R specify requests for single areas of virtual storage. The length of the single virtual area is indicated by the parameter, LV=*length value*. The address of the allocated virtual storage area is returned in register 1.

VRC and VRU indicate variable register conditional (VRC) and unconditional (VRU) requests for a single area of virtual storage. The length returned will be between the maximum and minimum lengths specified by the parameter LV=(*maximum length value, minimum length value*). The address of the allocated virtual storage is returned in register 1 and the length in register 0.

Notes:

1. A conditional request indicates that the task is not to be abnormally terminated if virtual storage is not allocated to the active task. An unconditional request indicates that the task is to be abnormally terminated in this situation.
2. The LC, LU, VC, VU, EC, EU, and R forms of the GETMAIN macro can only be used to obtain virtual storage with addresses below 16 megabytes. The RC, RU, VRC, and VRU forms of the GETMAIN macro can be used to obtain virtual storage with addresses above 16 megabytes.

LA specifies the virtual storage address of consecutive fullwords starting on a fullword boundary. Each fullword must contain the required length in the low-order three bytes, with the high-order byte set to 0. The lengths should be multiples of 8; if they are not, the control program uses the next higher multiple of 8. If VC or VU was coded, two words are required. The first word contains the minimum length required, the second word contains the maximum length. If LC or LU was coded, one word is required for each virtual storage area requested; the high-order bit of the last word must be set to 1 to indicate the end of the list. The list must not overlap the virtual storage area specified in the A parameter.

LV=*length value* specifies the length, in bytes, of the requested virtual storage. The number should be a multiple of 8; if it is not, the control program uses the next higher multiple of 8. If R is specified, LV=(0) may be coded; the low-order three bytes of register 0 must contain the length, and the high-order byte must contain the subpool number. LV=(*maximum length value, minimum length value*) specifies the maximum and minimum values of the length of the storage request.

The A parameter specifies the virtual storage address of consecutive fullwords, starting on a fullword boundary. The control program places the address of the virtual storage area allocated in one or more words. If E was coded, one word is required. If LC or LU was coded, one word is required for each entry in the LA list. If VC or VU was coded, two words are required. The first word contains the address of the virtual storage area, and the second word contains the length actually allocated. The list must not overlap the virtual storage area specified in the LA parameter.

,SP=*subpool nmbr*
specifies the number of the subpool from which the virtual storage area is to be allocated. The subpool number must be a valid subpool number between 0 and 255. See "Virtual Storage Management" in *SPL: Application Development Guide* for a list of the valid subpools. If this parameter is omitted, subpool 0 is assumed.

Notes:

1. Callers executing in supervisor state and key zero, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage via the SDUMP macro, they must specify subpool 252 rather than 0.
2. Storage requested from subpool 250 is always assigned from subpool 0 regardless of the caller's state or PSW key.

,BNDRY=DBLWD
,BNDRY=PAGE
specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

If the request specifies one of the LSQA or SQA subpools, the system ignores the BNDRY=PAGE keyword. Requests for storage from these subpools are then fulfilled from a single page, unless the request is greater than a page. See *SPL: Application Development Guide* for a list of the LSQA and SQA subpools.

,BRANCH=YES
,BRANCH=(YES,GLOBAL)
specifies that a branch entry is to be used. If (YES,GLOBAL) is specified, the entry point to service global storage requests without the need for the **local** lock is used. The caller must be disabled. If BRANCH=YES is specified, the caller must pre-load register 4 with the TCB address, pre-load register 7 with the ASCB address, and hold

the **local** lock prior to entry. The contents of register 3 is destroyed if RC, RU, VRC, or VRU is specified.

Callers in cross memory mode can use the BRANCH=YES parameter of the GETMAIN macro. If the caller is in cross memory mode, the storage is allocated in the currently addressable address space. The caller must hold the CML lock for the currently addressable address space; load register 7 with the address of the ASCB of the currently addressable address space; and load register 4 with zero or the address of a TCB in the currently addressable address space. If register 4 contains a zero, the allocated storage is associated with the current job step task that owns the cross memory resources in the currently addressable address space (that is, the TCB anchored in ASCBXTCB).

If BRANCH=(YES,GLOBAL) is specified, registers 4 and 7 need not contain the TCB and ASCB addresses; and registers 3 and 4 are changed when control returns to the caller. The caller must be disabled. Additionally, the SP parameter may only designate subpools 226, 227, 228, 231, 239, 241, 245, 247, or 248. Branch entry is available only to callers in supervisor state and PSW key 0.

,KEY=*key nmbr*
specifies the key (in bits 24-27 of the register) in which the requested storage is to be obtained. This parameter applies to subpools 227, 228, 229, 230, 231, and 241, and allows both global and local storage to be obtained in the specified storage protection key. The KEY parameter cannot be specified unless the BRANCH parameter is also specified.

,LOC=BELOW
,LOC=(BELOW,ANY)
,LOC=ANY
,LOC=(ANY,ANY)
,LOC=RES
,LOC=(RES,ANY)
specifies the location of virtual storage and central (also called real) storage. This is especially helpful for callers with 24-bit dependencies. When LOC is specified, central storage is allocated anywhere until the storage is fixed (by the PGFIX, PGFIXA, or PGSER macros). You can specify the location of central storage (after the storage is fixed) and virtual storage (whether or not the storage is fixed) in the following manner.

LOC=BELOW indicates that central and virtual storage are to be located below 16 megabytes. LOC=BELOW must not be used to allocate disabled reference (DREF) storage.

LOC=(BELOW,ANY) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere.

LOC=ANY and LOC=(ANY,ANY) indicate that virtual and central storage can be located anywhere.

Note: When you specify LOC=ANY, the actual location of the virtual storage (that is, whether it is above or below 16Mb) depends on the subpool you specify on the SP parameter:

- Some subpools (for example, subpool 226) are supported **only below** 16Mb. For these subpools, GETMAIN locates virtual storage below 16Mb, regardless of how you specify LOC.
- Some subpools (for example, 203-204) are supported **only above** 16Mb. For these subpools, GETMAIN locates virtual storage above 16Mb. If you specify LOC=BELOW for one of these subpools, the system abends your program.

All other subpools are supported both above and below 16Mb. For these subpools, specifying LOC=ANY causes GETMAIN to try to allocate virtual storage above 16Mb. If the attempt fails, GETMAIN tries to allocate virtual storage below 16Mb. If this attempt also fails, GETMAIN does not allocate any storage.

LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be located below 16 megabytes. If the caller resides above 16 megabytes, virtual and central storage are to be located anywhere. LOC=RES must not be used (or defaulted) to allocate disabled reference (DREF) storage when the caller resides below 16 megabytes.

LOC=(RES,ANY) 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 either case, central storage can be located anywhere.

,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.

When control is returned on conditional type requests (LC, EC, VC, RC, VRC), register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Virtual storage requested was allocated
04	No virtual storage was allocated
08	Central storage was not available for backing the request
0C	Page table is paged out

The contents of registers 0, 1, and 15 are not preserved when the GETMAIN macro is issued.

Example 1
Operation: Obtain 248 bytes of storage from the user's region via a branch entry. If the routine is in supervisor state, subpool 252 is used; otherwise, subpool 0 is used. If the storage cannot be obtained, the caller is abnormally terminated.

```
GETMAIN EU,LV=248,A=AREAADDR,BRANCH=YES
```

Example 2
Operation: Obtain one page of storage from the common service area, and cause the acquired storage to be initialized with a storage key of 9. A return code of 0 (if successful) or 4 (if unsuccessful) is returned.

```
GETMAIN RC,LV=4096,SP=231,BRANCH=(YES,GLOBAL),BNDRY=PAGE,KEY=9
```

Example 3
Operation: Obtain 400 bytes of storage from subpool 10. If the storage is available, the address will be returned in register 1 and register 15 will contain 0; if storage is not available, register 15 will contain 4.

```
GETMAIN   RC,LV=400,SP=10
```

Example 4

Operation: Obtain 48 bytes of storage from default subpool 0. If the storage is available, the address will be stored in the word at AREAADDR; if the storage is not available, the task will be abnormally terminated.

```
         GETMAIN EU,LV=48,A=AREAADDR
         .
         .
         .
AREAADDR DS      F
```

Example 5

Operation: Obtain a maximum of 4096 or a minimum of 1024 bytes of virtual storage, with addresses above or below 16 megabytes. Indicate that if the storage is fixed, the frames used to back the virtual may also be located either above or below 16 megabytes. If the storage is available, the address will be returned in register 1 and the length of the storage allocated will be returned in register 0; if the storage is not available, the task will be terminated.

```
GETMAIN VRU,LV=(4096,1024),LOC=ANY
```

GETMAIN (List Form)

Use the list form of the GETMAIN macro to construct a control program parameter list. The list form of the GETMAIN macro cannot be used to allocate virtual storage with addresses greater than 16 megabytes.

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

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

LC LU VC VU EC EU	
,LA=*length addr* ,LV=*length value*	*length addr:* A-type address. *length value:* symbol or decimal number. **Notes:** 1. LA may not be specified with EC or EU above. **2.** LV may not be specified with LC, LU, VC or VU above.
,A=*addr*	*addr:* A-type address.
,SP=*subpool nmbr*	*subpool nmbr:* symbol or decimal number.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the GETMAIN macro.

GETMAIN (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the GETMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN. The execute form of the GETMAIN macro cannot be used to allocate virtual storage with addresses greater than 16 megabytes.

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

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

LC LU VC VU EC EU	
,LA=*length addr* ,LV=*length value*	*length addr:* RX-type address or register (2) - (12). *length value:* symbol, decimal number, or register (2) - (12). **Note:** LA may not be specified with EC or EU above. **Note:** LV may not be specified with LC, LU, VC, or VU above.
,A=*addr*	*addr:* RX-type address, or register (2) - (12).
,SP=*subpool nmbr*	subpool nmbr: symbol, decimal number, or register (0) or (2) - (12).
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD
,BRANCH=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*ctrl prog*)	*ctrl prog:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl prog*)
specifies the execute form of the GETMAIN macro using a remote control program parameter list.

GQSCAN — Extract Information From Global Resource Serialization Queue

Use the GQSCAN macro to obtain the status of resources and requestors of those resources. The GQSCAN macro, in conjunction with the ISGRIB mapping macro, enables you to obtain resource information from system control blocks without knowing the exact structure or location of the control blocks.

The issuer of the GQSCAN macro must be executing in primary mode. To use SCOPE=GLOBAL and SCOPE=LOCAL, you must be in supervisor state or key 0 and you should be aware that improper use of these parameters degrades system performance.

Global resource serialization counts and limits the number of outstanding global resource serialization requests. A global resource serialization request is any ENQ, RESERVE, or GQSCAN that causes an element to be inserted into a queue in the global resource serialization request queue area. See "Limiting Global Resource Serialization Requests" in *SPL: Application Development Guide*.

Register 13 must contain the address of an 18-word save area, which can be provided through the use of standard linkage conventions.

On return, register 0 contains two halfword values. The first (high order) halfword contains the length of the fixed portion of each RIB returned; the second (low order) halfword contains the length of each RIBE returned. Register 1 contains the number of RIBs that were copied into the area provided. Register 15 contains the return code. In order to interpret the data that the GQSCAN service routine returns in the user-specified area, you must include the ISGRIB mapping macro as a DSECT in your program. ISGRIB maps the resource information block (RIB) and the resource information block extent (RIBE) as shown in *Diagnosis: Data Areas*.

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

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

AREA=*(area addr,area size)*	*area addr:* A-type address or register (2) - (12). *area size:* symbol, decimal digit, or register (2) - (12). **Note:** AREA cannot be specified with QUIT=YES.
,REQLIM=*value*	value: symbol, decimal digit, register (2) - (12), or the word MAX.
,REQLIM=MAX	**Default:** REQLIM=MAX
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS ,SCOPE=LOCAL ,SCOPE=GLOBAL	**Default:** SCOPE=STEP
,RESERVE=YES	**Default:** All resources requested with RESERVE and all resources requested with ENQ.
,RESERVE=NO	
,RESNAME=*(qname addr[,rname addr, rname length], [GENERIC|SPECIFIC], qname length)*	qname addr: RX-type address or register (2) - (12). *rname addr:* RX-type address or register (2) - (12). *rname length*: decimal digit, register (12). **Default:** assembled length of rname. **Default:** *qname length* of eight.

,SYSNAME=*(sysname addr [,asid value])*	*sysname addr*: RX-type address or register (2) - (12). *asid value:* symbol, decimal digit, or register (2) - (12). **Notes:** *rname addr* can be provided only when *qname addr* is used. *rname length* must be coded if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES ,QUIT=NO	**Default:** QUIT=NO **Note:** QUIT=YES is mutually exclusive with all parameters but TOKEN.
,REQCNT=*value* ,OWNERCT=*value*,WAITCNT=*value* ,OWNERCT=*value* ,WAITCNT=*value*	*value:* decimal digit or register (2) - (12). **Default:** REQCNT=0
,TOKEN=*addr*	*addr:* RX-type address or register (2) - (12).

The parameters are explained as follows:

AREA=*(area addr,area size)*
specifies the location and size of the area where information extracted from the global resource serialization resource queues is to be placed. The minimum size is the amount needed to describe a single resource, approximately 296 bytes, which is the length of the fixed portions of the RIB and the maximum size rname rounded up to a fullword value.

,REQLIM=*value*
,REQLIM=MAX
specifies the maximum number of owners and waiters to be returned for each resource, which can be any value between 0 and 2^{15}-1. MAX specifies 2^{15}-1.

,SCOPE=ALL
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
,SCOPE=LOCAL
,SCOPE=GLOBAL
specifies that you want information only for resources having the indicated scope. STEP, SYSTEM, or SYSTEMS is the scope specified on the resource request. If you specify SCOPE=ALL (meaning STEP, SYSTEM, and SYSTEMS), the system returns information for all resources the system recognizes that have the specified RESNAME, RESERVE, or SYSNAME characteristics. If you specify SCOPE=LOCAL, information is returned about this system's resources that are not being shared with other systems in the ring. If you specify SCOPE=GLOBAL, information is returned about resources that are being shared with other systems in the ring. Remember that entries in the resource name lists can cause the scope to change.

,RESERVE=YES
,RESERVE=NO
,RESNAME=*(qname addr[,rname addr,rname length],[GENERIC|SPECIFIC], qname length)*
,SYSNAME=*(sysname addr [,asid value])*
For most requests, RESERVE=YES specifies that information is to be returned for resources requested with the RESERVE macro. If a RESERVE macro is issued for a device that is not shared, global resource serialization treats the RESERVE request as an ENQ and the GQSCAN macro does not return information for the resource request when RESERVE=YES.

RESERVE=NO specifies that information is to be returned for resources requested with the ENQ macro.

RESNAME (with *rname)* indicates the name of one resource.

The *qname addr* specifies the virtual storage address of the 8-character major name of the requested resource.

The *rname addr* specifies the virtual storage address of a 1 to 255-byte minor name used in conjunction with the major name to represent a single resource. Information returned is for a single resource unless you specify SCOPE=ALL, in which case it could be for three resources (STEP, SYSTEM, and SYSTEMS) or SCOPE=LOCAL in which case it could be for two resources (STEP and SYSTEM) if there is a matching name in each of these categories. If the name specified by *rname* is defined by an EQU assembler instruction, the *rname length* must be specified.

The *rname length* specifies the length of the minor name. If you use the register form, the low-order (rightmost) byte contains the length. The length must match the *rname length* specified on ENQ or RESERVE.

GENERIC specifies that the rname of the requested resource must match but only for the length specified. For example, an ENQ for SYS1.PROCLIB would match the GQSCAN rname specified as SYS1 for an rname length of 4.

SPECIFIC specifies that the rname of the requested resource must exactly match the GQSCAN rname.

Note: GENERIC and SPECIFIC are mutually exclusive.

The qname length specifies the length of the qname in the resource name that must match the GQSCAN name.

SYSNAME specifies that information is to be returned for resources requested by tasks running on an MVS system whose system name matches the one specified by SYSNAME, where *sysname addr* is the address of an 8-byte field that contains the system name, and asid value specifies a 4-byte address space identifier, right justified. Information returned includes only those resources whose *sysname addr* and *asid value* match the ones specified. SYSNAME=0 or SYSNAME=(0,asid value), specifies that the system name is that of the system on which GQSCAN is issued.

,QUIT=YES
,QUIT=NO
indicates whether or not you want to terminate the current global resource serialization queue scan. If QUIT=YES is specified with TOKEN, GQSCAN processing terminates the current GRS queue scan and releases the storage allocated to accumulate the information specified in the token.

,REQCNT=*value*
,OWNERCT=*value*,**WAITCNT=***value*
,OWNERCT=*value*
,WAITCNT=*value*
specifies that you only want information about resources that fall into the following categories:

- The total number of requestors (that is, owners plus waiters) is greater than or equal to REQCNT.
- The total number of owners is greater than or equal to OWNERCT.
- The total number of waiters is greater than or equal to WAITCNT.

If you do not specify REQCNT, you can specify both OWNERCT and WAITCNT. If you specify REQCNT, you cannot specify either OWNERCT or WAITCNT.

,TOKEN=*addr*
specifies the address of a fullword of storage that the GQSCAN service routine can use in subsequent invocation to provide you with any remaining information. If the token is zero, the scan starts at the beginning of the resource queue. You must zero the token each time you want the scan to start over. If the token is not zero, the scan resumes at the point from which it left off.

When GQSCAN returns control, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
0	Queue scan processing is complete. Data is now in the area you specified. On a resumed GQSCAN, the code signifies that there are no more resources to match your request.
4	Queue scan processing is complete. No resources matched your request.
8	The area you specified was filled before queue scan processing completed. If you specified TOKEN, process the information in the area and issue GQSCAN again specifying the TOKEN returned to you. If you did not specify TOKEN, you must begin again and either specify a larger area or specify a TOKEN.
C	Queue scan encountered an abnormal situation while processing. The information in your area is not meaningful. The values in registers 0 and 1 are also meaningless.
10	An invalid SYSNAME was specified as input to queue scan. The information in your area is not meaningful.
14	The area you specified was filled before queue scan processing completed. Your request specified TOKEN=, but you have too many outstanding ENQ or RESERVE and GQSCAN requests. The information in your area is valid but incomplete. The scan cannot be resumed.

GQSCAN (List Form)

The list form of the GQSCAN macro is used to construct a non-executable parameter list. This parameter list, or a copy of it for reentrant programs, can be referred to by the execute form of the GQSCAN macro.

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

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

AREA=*(area addr, area size)*	*area addr:* A-type address. *area size:* symbol, decimal digit. **Notes:** 1. This parameter cannot be specified with QUIT=YES. 2. AREA is required on either the list or the execute form of the macro.
,REQLIM=*value* ,REQLIM=MAX	*value:* symbol, decimal digit or the word MAX. **Default:** REQLIM=MAX
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS	**Default:** SCOPE=STEP
,RESERVE=YES ,RESERVE=NO	**Default:** All resources requested with RESERVE and all resources requested with ENQ.
,RESNAME=*(qname addr* [,*rname addr, rname length*], [*GENERIC|SPECIFIC*], *qname length)*	*qname addr:* A-type address. *rname addr:* A-type address. *rname length:* decimal digit. **Default:** assembled length of rname. **Default:** *qname length* of eight.
,SYSNAME=*(sysname addr*[,*asid value*]*)*	*sysname addr:* A-type address. *asid value:* symbol, decimal digit. **Notes:** *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES ,QUIT=NO	**Default:** QUIT=NO **Note:** Only TOKEN and MF=L can be specified with QUIT=YES.
,REQCNT=*value*	*value:* decimal digit. **Default:** REQCNT=0
,OWNERCT=*value* ,WAITCNT=*value*	
,TOKEN=*addr*	*addr:* RX-type address.
,MF=L	

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

,MF=L
specifies the list form of the GQSCAN macro.

GQSCAN (Execute Form)

The execute form of the GQSCAN macro can refer to and modify a remote parameter list built by the list form of the macro. There are no defaults for any of the parameters in the execute form of the macro.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede GQSCAN.
GQSCAN	
b	One or more blanks must follow GQSCAN.
AREA=*(area addr,area size)*	*area addr:* RX-type address or register (2) - (12). *area size:* symbol, decimal digit, or register (2) - (12). **Notes:** 1. AREA cannot be specified with QUIT=YES. **2.** AREA is required on either the list or the execute form of the macro.
,REQLIM=*value* ,REQLIM=MAX	value: symbol, decimal digit, register (2) - (12), or the word MAX.
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS ,SCOPE=LOCAL ,SCOPE=GLOBAL	**Note:** SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.
,RESERVE=YES ,RESERVE=NO ,RESNAME=*(qname addr*[,*rname addr,rname length*], [*GENERIC*|*SPECIFIC*], *qname length)* ,SYSNAME=*(sysname addr*[,*asid value*]*)*	**Default:** All resources requested with RESERVE and all resources requested with ENQ. qname addr: RX-type address or register (2) - (12). *rname addr:* RX-type address or register (2) - (12). rname length: decimal digit, register (2) - (12). **Default:** assembled length of rname. sysname addr: RX-type address or register (2) - (12). *asid value:* symbol, decimal digit, or register (2) - (12). **Note:** *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES ,QUIT=NO	**Default:** QUIT=NO **Note:** Only TOKEN and MF=(E, *parm list addr*) can be specified with QUIT=YES.
,REQCNT=*value*	*value:* decimal digit or register (2) - (12).
,OWNERCT=*value* ,WAITCNT=*value*	
,TOKEN=*addr*	*addr:* RX-type address of register (2) - (12).
,MF=(E,*parm list addr*)	*parm list addr:* RX-type address or register (2) - (12).

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

,MF=(E,*parm list addr)*
specifies the execute form of the GQSCAN macro. This form uses a remote parameter list defined by parm list addr.

GTRACE — GTF Trace Recording

Use the GTRACE macro to record system or application errors through the generalized trace facility (GTF). The GTRACE macro provides two separate functions, depending on the keyword specified:

- GTRACE TEST indicates whether the operator requested a specific user event.
- GTRACE DATA generates GTF trace records for specific events.

Refer to *Service Aids* and *Planning: Dump and Trace Services* for information about using GTF.

Refer to *Diagnosis: Using Dumps and Traces* for descriptions of GTF records.

The following description of the GTRACE macro is divided into two sections, one for each function of the macro. The TEST function has only one form, while the DATA function has standard, list, and execute forms.

GTRACE TEST

The TEST function of the GTRACE macro indicates whether the operator requested a particular user event in response to the USRP option. The system returns the test result as a return code in register 15.

GTRACE DATA always records data for any specified event if any user tracing is active. By issuing GTRACE TEST and checking the return code, you can determine whether you should subsequently issue GTRACE DATA to write the record: If the return code indicates that tracing has been requested by USRP for the specified user event, then issue GTRACE DATA.

Requirements

The requirements for the caller are:

Authorization:	Problem or supervisor state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Control parameters must be in the primary address space.

Restrictions and Limitations

None.

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 registers contain:

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

Programming Requirements

- Include the CVT and the MCHEAD mapping macros.
- When you code the CVT mapping macro, you must not specify PREFIX=YES.

Performance Implications

None.

Syntax Diagram

The TEST function of the GTRACE macro is coded as follows:

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

TEST=YES	
,ID=*id*	*id*: symbol, decimal digit, or hexadecimal number.

Parameter Descriptions

The parameters are explained as follows:

TEST=YES
specifies the test function of the GTRACE macro.

,ID=*id*
specifies the event ID for the user event that is to be tested. Decimal event IDs 0 through 1023 (X'3FF') are available for user events. You can specify the ID in decimal or in hexadecimal. Use the expression X'*id*' to specify a hexadecimal number.

Return Codes

Return codes are as follows:

Return Code	Meaning
0	Tracing has not been requested by USRP for the specified user event.
4	Tracing has been requested by USRP for the specified user event.

GTRACE DATA

The DATA function of the GTRACE macro records system or problem program data in the GTF trace buffers. GTRACE DATA can trace up to 256 bytes of data.

GTRACE DATA writes the record to the GTF data set even if the record's event ID (EID) is excluded from a USRP list in the GTF trace options. Therefore, before using GTRACE DATA to record data, you might want to issue GTRACE TEST to determine if GTF should store data for this event ID.

Requirements

The requirements for the caller are:

Authorization:	Problem or supervisor state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit. The caller must be in 31-bit mode for GTRACE to record data above 16 megabytes.
ASC mode:	Primary
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Control parameters must be in the primary address space

Restrictions and Limitations

None.

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 registers contain:

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

Programming Requirements

None.

Performance Implications

None.

Syntax Diagram

The standard form of the DATA function of the GTRACE macro is coded as follows:

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

DATA=*addr*	*addr*: A-type address of register (2) - (12)
,LNG=*nbr*	*nbr*: symbol, decimal number, hexadecimal number, or register (2) - (12)
,ID=*id*	*id*: symbol, decimal number, or hexadecimal number.
,FID=*fidname*	*fidname*: symbol, decimal number, hexadecimal number, or register (2) - (12)
,PAGEIN=NO ,PAGEIN=YES	**Default**: PAGEIN=NO

Parameter Descriptions

The parameters are explained as follows:

DATA=*addr*
specifies the virtual storage address of the data that is to be recorded.

,LNG=*nbr*
specifies the number of data bytes (1 through 256) to be recorded from the address specified by the DATA parameter. You can specify the number in decimal or in hexadecimal. If the number is hexadecimal, use the expression X'*nbr*' to specify the number.

Note: When you specify LNG, the trace record contains the number of bytes that you specify plus 12 bytes, which is the size of the trace record header. The header consists of a 4-byte ASCB address followed by an 8-byte jobname. Thus, if you specify LNG=256, the trace record has 268 (256+12) bytes.

,ID=*id*
specifies the event ID that is to be recorded with the data bytes. Decimal event ids 0 through 1023 (X'3FF') are available for user events. You can specify the ID in decimal or in hexadecimal. Use the expression X'*id*' to specify a hexadecimal number.

,FID=*fidname*
specifies the format appendage that controls the formatting of this record. Formatting occurs when the trace output is processed by GTF trace. The format appendage name is formed by appending the 2-digit FID value to the names AMDUSR, HMDUSR, and IMDUSR. Assign FID values as follows:

XX'00'	The record is to be dumped in hexadecimal.
XX'01 to XX'50'	The record contains user format identifiers.

Note: If you code FID without any *fidname*, or if you omit the FID parameter, the system supplies a default *fidname* of zero.

,PAGEIN=NO
,PAGEIN=YES
specifies that paged-out user data is to be processed (YES) or not to be processed (NO). To insure that all user data is traced, specify YES.

Return Codes

Return codes are as follows:

Return Code	Meaning
0	GTF is active. The data was recorded in GTF trace buffers.
4	GTF is not active. No data was recorded.
8	The value of the LNG keyword is not valid. It must be a number from 1 through 256. No data was recorded.
C	The value of the DATA keyword is not valid. It does not represent an area of storage that the calling program can refer to. No data was recorded.
10	The value of the FID keyword is not valid. It must be a number from X'0' through X'FF'. No data was recorded.
18	All GTF buffers are full. No data was recorded.
1C	The address of the parameter list for GTF is not valid. The parameter list is not in storage that the caller can refer to, or its format is not valid. No data was recorded.
20	Some of the data to be recorded was paged out. No data was recorded. This return code is not valid with PAGEIN=YES.

Example

Operation: Use GTRACE to record 200 bytes of user data plus 12 bytes for the trace record header. The user data is found at symbolic address AREA. Use an event identifier of 37. Use the formatting appendage named IMDUSR40 to control the formatting.

```
GTRACE DATA=AREA,LNG=200,ID=37,FID=X'40'
```

GTRACE DATA (List Form)

Use the list form of the GTRACE DATA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form of the macro uses to store the parameters.

Syntax Diagram

The list form of the DATA function of the GTRACE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede GTRACE.
GTRACE	
b	One or more blanks must follow GTRACE.
DATA=*addr*	*addr*: A-type address or register (2) - (12)
,LNG=*nbr*	*nbr*: symbol, decimal number, hexadecimal number, or register (2) - (12)
,FID=*fidname*	*fidname*: symbol, decimal number, hexadecimal number, or register (2) - (12)
,MF=L	

Parameter Descriptions

The parameters are described under the standard form of the GTRACE DATA macro, with the following exception:

,MF=L
specifies the list form of the GTRACE DATA macro.

GTRACE DATA (Execute Form)

Use the execute form of the GTRACE DATA 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 Diagram

The execute form of the DATA function of the GTRACE macro is written as follows:

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

DATA=*addr*	*addr*: A-type address or register (2) - (12)
,LNG=*nbr*	*nbr*: symbol, decimal number, hexadecimal number, or register (2) - (12)
,ID=*id*	*id*: symbol, decimal number, or hexadecimal number.
,FID=*fidname*	*fidname*: symbol, decimal number, hexadecimal number, or register (2) - (12) **Note:** If you omit the FID parameter on the execute form of GTRACE, the FID value defaults to zero. This default overlays the FID value that you specify on the list form of GTRACE. If you want the system to obtain the FID value from the remote problem-program parameter list, then you must specify the FID parameter as a null value by coding FID= without any *fidname*.
,PAGEIN=NO ,PAGEIN=YES	**Default**: PAGEIN=NO
,MF=(E,*parm list addr*)	*parm list addr*: A-type address or register (2) - (12)

Parameter Descriptions

The parameters are described under the standard form of the GTRACE DATA macro, with the following exception:

,MF=(E,*parm list addr*)
specifies the execute form of the GTRACE DATA macro using a remote problem-program parameter list.

HSPSERV — Read from and Write to a Hiperspace

HSPSERV transfers data between virtual storage areas in address spaces and hiperspaces. It reads data from a hiperspace to an address space and it writes data to a hiperspace from an address space.

A hiperspace can be either a **standard hiperspace**, of which there are two types, shared and non-shared, or an **ESO** (expanded storage only) **hiperspace**:

- The non-shared standard hiperspace, and the shared standard hiperspace are backed with expanded storage and, if necessary, auxiliary storage. Through the buffer area in the address space, your program can view or **scroll** through the hiperspace. HSPSERV SWRITE and HSPSERV SREAD transfer data to and from a standard hiperspace. You create a standard hiperspace through the HSTYPE=SCROLL parameter on the DSPSERV macro. The description of HSPSERV macro for standard hiperspaces begins on this page.
- The ESO hiperspace is backed only with expanded storage. It is a high-speed buffer area or **cache** for data that your program needs. HSPSERV CWRITE and HSPSERV CREAD transfer data to and from an ESO hiperspace. You create an ESO hiperspace through the HSTYPE=CACHE parameter on the DSPSERV macro. The description of the HSPSERV macro for ESO hiperspaces begins on 282.

The STOKEN parameter identifies the specific hiperspace to be read from or written to. The HSPALET parameter specifies an optional ALET for the hiperspace. The RANGLIST parameter identifies the storage range(s) in the address space and the storage range(s) in the hiperspace. A storage range consists of contiguous 4K byte blocks starting on a 4K byte boundary.

If you code the HSPALET parameter on the HSPSERV macro, you must first code the SYSSTATE macro to indicate the ASC mode of your program.

HSPSERV is also described in *Application Development Macro Reference*, with the exception of the parameters that are valid only for supervisor state or PSW key 0 through 7 programs: CREAD, CWRITE, ADDRSP, and KEEP. For more information about hiperspaces and data spaces see *SPL: Application Development — Extended Addressability*.

Read and Write Services for Standard Hiperspaces

The requirements for the caller who specifies SREAD and SWRITE are:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Can be in cross memory mode, except for a non-shared standard hiperspace for which an ALET is not used (that is, the HSPALET parameter is omitted).
Amode:	31-bit addressing
ASC mode:	Primary or AR
Serialization:	Enabled and unlocked
Control parameters:	The control parameters must be in the caller's primary address space. If the caller's PSW key is not zero, the PSW key must match the storage key associated with the control parameters.

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

If you coded the HSPALET parameter, HSPSERV processing might have changed the contents of the parameter list and the list of ranges.

The following figure describes the characteristics and restrictions for the use of standard hiperspaces, the hiperspaces that allow your program to **scroll** through large areas of data.

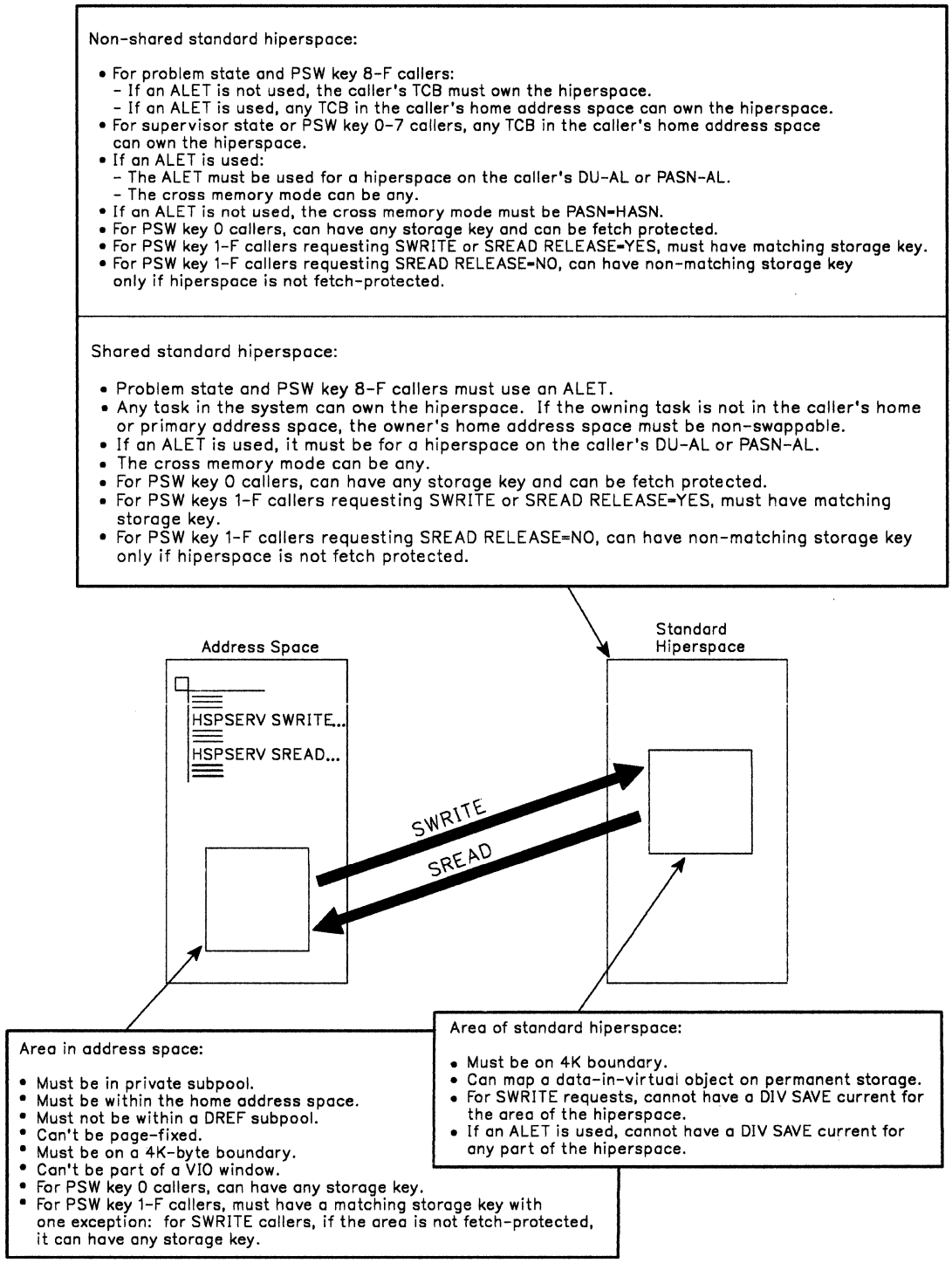


Figure 11. Characteristics and Restrictions for Standard Hiperspaces

The standard form of the HSPSERV macro for standard hiperspaces is written as follows:

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

b                                One or more blanks must precede HSPSERV.

HSPSERV

b                                One or more blanks must follow HSPSERV.

SREAD
SWRITE

,STOKEN=stoken-addr              stoken-addr: RX-type address or register (2) - (12).

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

   ,NUMRANGE=n                   n: Number from 1 to 50.
   ,NUMRANGE=num-addr            num-addr: RX-type address or register (2) - (12).
                                 Default: NUMRANGE=1.

,RANGLIST=list addr              list addr: RX-type address or register (2) - 12).

   ,RELEASE=NO                   Default: RELEASE=NO.
   ,RELEASE=YES

   ,RETCODE=ret-addr             ret-addr: RX-type address or register (2) - (12).

   ,RSNCODE=rsn-addr             rsn-addr: RX-type address or register (2) - (12).

   ,MF=S
```

The parameters are explained as follows:

SREAD
Requests that the system read data from a standard hiperspace to an address space.

STOKEN and RANGLIST are required parameters on the SREAD request. NUMRANGE, HSPALET, RELEASE, RSNCODE, and RETCODE are optional parameters.

SWRITE
Requests that the system write data to a standard hiperspace from an address space.

Notes

- When HSPSERV returns to the caller after the SWRITE operation, the contents of the address space storage range are not preserved. You can use the address space area again.
- If the hiperspace maps a data-in-virtual object, do not issue an SWRITE request while a DIV SAVE request is current.

STOKEN and RANGLIST are required parameters on the SWRITE request. NUMRANGE, HSPALET, RETCODE, and RSNCODE are optional parameters.

,STOKEN = *stoken-addr*
Specifies the address of the eight-character variable that contains the STOKEN for the standard hiperspace from which the data is to be read or into which the data is to be written. Restrictions on standard hiperspaces are described in Figure 11 on page 279.

,HSPALET = *alet-addr*
Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

The HSPALET parameter is optional except for the following case: If the caller accesses a shared hiperspace, is in problem state and has PSW key 8 - F, HSPALET is required.

Use of the HSPALET parameter requires that the caller provide a 144-byte save area in the caller's primary address space. AR/GPR 13 must provide addressability to this area regardless of the caller's ASC mode. GPR 13 must contain the address of the area and AR 13 must contain 0.

If you code HSPALET, do not code RELEASE=YES.

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

,NUMRANGE=*n*
,NUMRANGE=*num-addr*

Specifies the number of entries, from 1 to 50, or specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries. The default is NUMRANGE=1.

If you omit NUMRANGE, then HSPSERV reads or writes one virtual range.

,RANGLIST=*list addr*

Specifies a fullword that contains an **address of** a list of ranges (up to 50) that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list. The range list consists of a number of entries (specified by NUMRANGE) where each entry consists of three words as follows:

First Word The starting virtual address in the address space into which the data is to be read or from which the data is to be written.

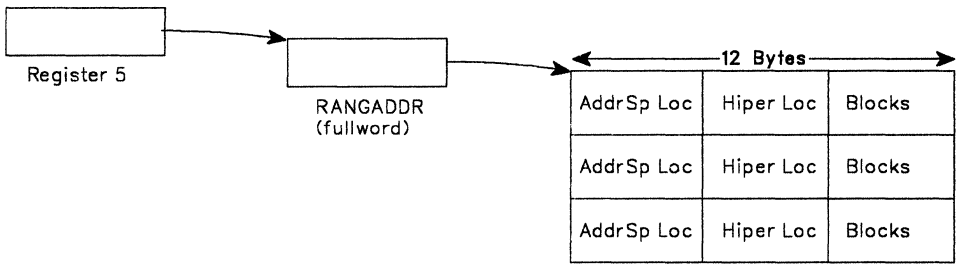
Second Word The starting virtual address in the hiperspace from which the system is to read or into which the system is to write.

Third Word The number of blocks the system is to read or write.

Note that the address is the block number followed by 12 binary zeroes.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:

```
NUMRANGE=3, RANGLIST=(5)
              or
 NUMRANGE=3, RANGLIST=RANGADDR
```



Restrictions on the areas in the address space and the hiperspace are described in Figure 11 on page 279.

The range list must be addressable in the caller's primary address space.

,RELEASE=NO
,RELEASE=YES

Specifies whether or not the system is to release the hiperspace pages after it completes the SREAD operation. RELEASE is valid only with SREAD.

RELEASE=NO specifies that the system does not release the hiperspace pages after it completes the SREAD operation. Unless a subsequent SWRITE request changes the data, the same data will be available again on the next SREAD request. RELEASE=NO is the default.

RELEASE=YES specifies that, after the SREAD request, the system is to release the storage that backed the data in the hiperspace. If you code RELEASE=YES, do not code HSPALET.

,RSNCODE= *rsn-addr*
Specifies the location where the system is to copy the reason code from register 0.

,RETCODE= *ret-addr*
Specifies the location where the system is to copy the return code from register 15.

,MF=S
Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the service.

When control is returned from HSPSERV SREAD or HSPSERV SWRITE, register 15 contains one of the following return codes:

Code	Meaning
00	HSPSERV completed successfully.
08	The system rejected the HSPSERV request. See the reason codes.
0C	System failure due to environmental problems.

For the SREAD and SWRITE requests, the system might return one of the following reason codes in register 0 with a "08" return code:

Reason codes:	Description:
xxxx05xx	The system rejects the request. A hiperspace page is unavailable because of a system error.
xxxx06xx	The system rejects the request. An address space page is unavailable because of a system error.

Read and Write Services for ESO Hiperspaces

The requirements for the caller who requests CREAD and CWRITE are:

Authorization:	Supervisor state or PSW key 0 - 7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
Amode:	31-bit addressing
ASC mode:	Primary or AR mode
Serialization:	Enabled or disabled; no locking requirements
Control parameters:	The parameter list and range list must be in non-pageable storage. If the caller specifies HSPALET and is disabled, the save area must also be in non-pageable storage. The parameter list, range list, and save area must all be in the common area or in the private area of the caller's primary address space.

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

The following figure describes the characteristics and restrictions for the use of ESO hiperspaces, the hiperspaces that act as a high-speed buffer or **cache** for data.

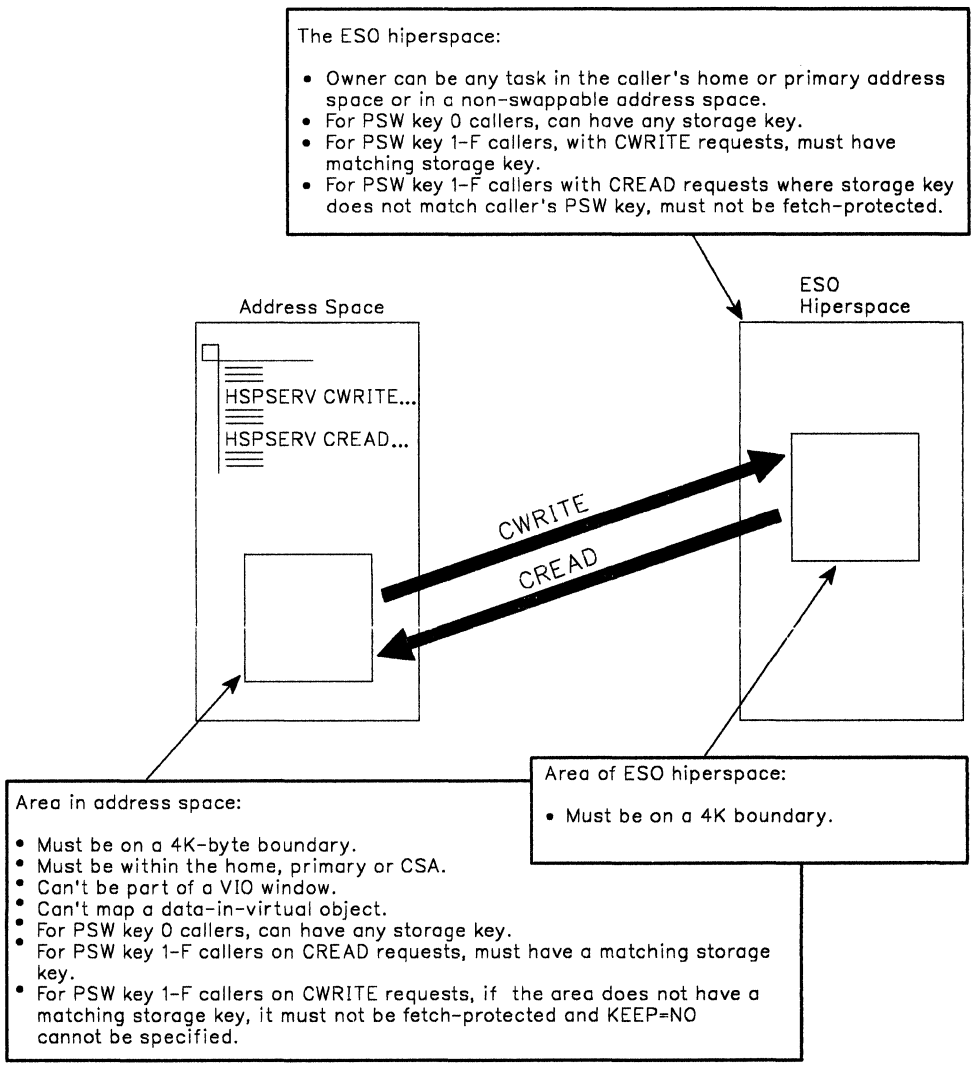


Figure 12. Characteristics and Restrictions for ESO Hiperspaces

The standard form of the HSPSERV macro for ESO hiperspaces follows.

CAUTION:
Code the parameters on the HSPSERV CREAD and HSPSERV CWRITE macros very carefully. Read the requirements for the address space buffer and the hiperspace, as listed in Figure 12 on page 283. For performance reasons, the system does not verify the location of the addresses you specify on these macros. Incorrect coding can cause damage to the system.

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

CREAD CWRITE	
,STOKEN=*stoken-addr*	*stoken-addr:* RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=*n* ,NUMRANGE=*num-addr*	*n*: A number from 1 to 50. *num-addr:* RX-type address or register (2) - (12). **Default:** NUMRANGE=1.
,RANGLIST=*list addr*	*list addr:* RX-type address or register (2) - (12).
,ADDRSP=HOME ,ADDRSP=PRIMARY ,ADDRSP=COMMON	**Default:** ADDRSP=HOME.
,KEEP=YES ,KEEP=NO	**Default:** KEEP=YES.
,RETCODE=*ret-addr*	*ret-addr:* RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr:* RX-type address or register (2) - (12).
,MF=S	

The parameters are explained as follows:

CREAD
Requests that the system read data from an ESO hiperspace

If all blocks requested to be read are available in the hiperspace, then the system performs the read operation. However, if one or more of the blocks to be read are no longer available in the hiperspace, then the system returns a failing return code. See return code 08. In this case, the system does not tell you which blocks it successfully reads, if any.

STOKEN and RANGLIST are required parameters on the CREAD request. ADDRSP, NUMRANGE, RSNCODE, and RETCODE are optional parameters.

CWRITE
Requests that the system write data to an ESO hiperspace. If the system cannot write all the requested blocks to the hiperspace, then it doesn't write any and rejects the request (See return code 08). In this case, the data in the specified range in the hiperspace is unpredictable. Therefore, after an unsuccessful write, do not issue another CREAD against the failing hiperspace range of virtual storage until an intervening CWRITE is successful.

STOKEN and RANGLIST are required parameters on the CWRITE request. ADDRSP, NUMRANGE, KEEP, RSNCODE, and RETCODE are optional parameters.

,STOKEN = *stoken-addr*
Specifies the address of the 8-character variable that contains the STOKEN for the ESO hiperspace from which the data is to be read or into which the data is to be written. Restrictions on the hiperspace are described in Figure 12 on page 283.

,HSPALET = *alet-addr*
Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

Use of the HSPALET parameter requires that the caller provide a 144-byte save area in non-pageable storage in the caller's primary address space or in the common area. AR/GPR 13 must provide addressability to this area regardless of the caller's ASC mode. GPR 13 must contain the address of the area and AR 13 must contain 0.

If you code HSPALET, do not code RELEASE = YES.

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

,NUMRANGE = *n*
,NUMRANGE = *num-addr*
Specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries, or specifies the number of entries, from 1 to 50. The default is NUMRANGE = 1.

If you omit NUMRANGE, then HSPSERV reads or writes one virtual range.

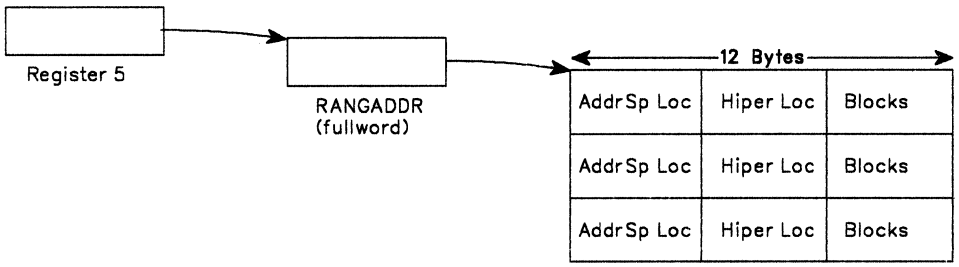
,RANGLIST = *list addr*
Specifies a fullword that contains the **address of** a parameter area in non-pageable storage that contain a list of up to 50 ranges that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list.

The range list consists of a number of entries (specified by NUMRANGE) where each entry consists of three words as follows:

First Word	The starting virtual address in the address space into which the data is to be read or from which the data is to be written.
Second Word	The starting virtual address in the hiperspace from which the system is to read or into which the system is to write.
Third Word	The number of blocks the system is to read or write.

An example of how to code the RANGLIST parameter when NUMRANGE = 3 is as follows:

```
NUMRANGE=3, RANGLIST=(5)
              or
 NUMRANGE=3, RANGLIST=RANGADDR
```



Restrictions on the areas in the address space and the hiperspace are described in Figure 12 on page 283.

The range list must be in the common area or in the private area of the caller's primary address space.

,ADDRSP = HOME
,ADDRSP = PRIMARY
,ADDRSP = COMMON
Specifies the location of the virtual storage range from which the system is to read or into which the system is to write. The location can be the caller's home address space (ADDRSP = HOME), the caller's primary address space (ADDRSP = PRIMARY), or the CSA (ADDRSP = COMMON). The default is ADDRSP = HOME.

,KEEP = YES
,KEEP = NO
Specifies whether or not the system preserves the source data in the virtual storage of the address space after it completes the CWRITE request. KEEP is valid only on the CWRITE request.

If you specify KEEP = YES, the data in the specified address space is unchanged and available for reference. The default is KEEP = YES.

If you specify KEEP = NO, the system might not preserve the data in the address space. If your program will reuse the same virtual storage area after the CWRITE request completes, use KEEP = NO.

,RSNCODE = *rsn-addr*
Specifies the location where the system is to copy the reason code from register 0.

,RETCODE = *ret-addr*
Specifies the location where the system is to copy the return code from register 15.

,MF = S
Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the macro service.

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

Code	Meaning
00	HSPSERV completed successfully.
08	The system rejected the HSPSERV request. See the reason code.

For the CREAD and CWRITE requests, the system might return one of the following reason codes with a "08" return code:

Reason codes:	Description:
xxxx01xx	The hiperspace data you requested is not available (CREAD request).
xxxx02xx	The system rejects the request because an address space page is not currently backed by central (also called real) or expanded storage. You can repeat the HSPSERV request after you reference the range(s), which causes the system to page the storage in (CWRITE request).
xxxx03xx	The system rejects the request because the necessary system resources are not currently available (CWRITE request).
xxxx04xx	The system rejects the request.
xxxx05xx	The system rejects the request. A hiperspace page is unavailable because of a system error.
xxxx06xx	The system rejects the request. An address space page is unavailable because of a system error.

HSPSERV Macro (List Form)

The list form of the HSPSERV macro creates a control parameter list. The modify and execute forms of the macro modify this parameter list.

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

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

MF=(L,*list addr*) MF=(L,*list addr,attr*)	*list addr:* symbol. *attr*: 1- to 60-character input string. **Default**: 0D
,PLISTVER=*vernum*	*vernum*: parameter list version 0 or 1 **Default:** Version that allows all specified parameters

Parameters for the list form of HSPSERV are as follows:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
Specifies the list form of HSPSERV. The list form defines an area that the system uses as a parameter list. *list addr* is the address of the storage area for the parameter list.

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

,PLISTVER=*vernum*
specifies the macro version associated with HSPSERV. PLISTVER is an optional parameter that determines which parameter list the system generates. Specify 0 if you use parameters only from this group:

ADDRSP
CREAD
CWRITE
KEEP
MF
NUMRANGE
PLISTVER
RANGLIST
RELEASE
RETCODE
RSNCODE
SREAD
STOKEN
SWRITE

If you use the HSPALET parameter, specify 1.

The default is the version that allows all of the parameters specified on the invocation to be processed.

HSPSERV Macro (Modify Form)

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

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

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

SREAD SWRITE CREAD CWRITE	
,STOKEN=*stoken-addr*	*stoken-addr:* RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=1 ,NUMRANGE=*num-addr*	**Default:** NUMRANGE=1. *num-addr:* RX-type address or register (2) - (12).
,RANGLIST=*list addr*	*list addr:* RX-type address or register (2) - (12).
,RELEASE=NO ,RELEASE=YES	**Default:** RELEASE=NO.
,ADDRSP=HOME ,ADDRSP=PRIMARY ,ADDRSP=COMMON	**Default:** ADDRSP=HOME.
,KEEP=YES ,KEEP=NO	**Default:** KEEP=YES.
,RETCODE=*ret-addr*	*ret-addr:* RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr:* RX-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE) ,MF=(M,*list addr*,NOCHECK)	*list addr*: RX-type address or register (2) - (12). **Default**: COMPLETE.

Parameters for the modify form of HSPSERV 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 macro. *list-addr* is the address of a non-pageable storage area for the parameter list that the system generated from the list form of the macro.

COMPLETE specifies that the system checks for required parameters and supplies the optional parameters that you did not specify. NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify. COMPLETE is the default.

HSPSERV Macro (Execute Form)

The execute form of the HSPSERV macro changes parameters in the control parameter list that the system created through the list form of the macro and performs the specified operation.

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

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

SREAD SWRITE CREAD CWRITE	
,STOKEN=*stoken-addr*	*stoken-addr:* RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=1 ,NUMRANGE=*num-addr*	**Default:** NUMRANGE=1. *num-addr:* RX-type address or register (2) - (12).
,RANGLIST=*list-addr*	*list-addr:* RX-type address or register (2) - (12).
,RELEASE=NO ,RELEASE=YES	**Default:** RELEASE=NO.
,ADDRSP=HOME ,ADDRSP=PRIMARY ,ADDRSP=COMMON	**Default:** ADDRSP=HOME.
,KEEP=YES ,KEEP=NO	**Default:** KEEP=YES.
,RETCODE=*ret-addr*	*ret-addr:* RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr:* RX-type address or register (2) - (12).
,MF=(E,*list-addr*,COMPLETE) ,MF=(E,*list-addr*,NOCHECK)	*list-addr*: RX-type address or register (2) - (12). **Default**: COMPLETE.

Parameters for the execute form of HSPSERV are described in the standard form of the macro with the following exceptions:

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

Specifies the execute form of the macro. This form generates code to place the parameters into the parameter list. *list-addr* is the address of a non-pageable storage area for the parameter list.

COMPLETE specifies that the system checks for required parameters and supplies the optional parameters that you did not specify.

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

IEFQMREQ — Invoke SWA Manager in Move Mode

This macro is used to invoke the Move SWA manager in move mode. The IEFQMREQ macro, which has no parameters, is written as follows:

name	*name:*
ḃ	One or more blanks must precede IEFQMREQ.
IEFQMREQ	
ḃ	One or more blanks must follow IEFQMREQ.

Register 1 must contain the address of the queue manager parameter area (QMPA). Register 13 must contain the address of a standard 18 word save area.

For additional information on the use of IEFQMREQ, see *SPL: Application Development Guide*.

IOSINFO — Obtain the Subchannel Number for a UCB

The IOSINFO macro obtains the subchannel number for a specified unit control block (UCB). The macro returns the subsystem identification word (SID), which identifies the subchannel number of the UCB, in a user-specified location. The SID is a fullword value whose first halfword contains X'0001' and ending halfword contains the subchannel number.

The issuer of IOSINFO must be executing:

- In 31-bit addressing mode
- In either task mode or SRB mode
- Locked or unlocked

Additionally, the issuing program must include the CVT and IHAPSA mapping macros. All addresses must be 31-bit addresses.

Before entry to this macro, register 13 must contain the address of a standard 18-word save area.

The IOSINFO macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSINFO.
IOSINFO	
b	One or more blanks must follow IOSINFO.
FUNCTN=SUBCHNO	
,UCB=*ucb addr*	*ucb addr:* A-type address or register (0) - (15).
,OUTPUT=*output addr*	*output addr:* A-type address or register (0) - (14).
,RTNCODE=*rtncode addr*	*rtncode addr:* A-type address or register (0) - (15).

The parameters are explained as follows:

FUNCTN=SUBCHNO
specifies that a subchannel number is to be obtained.

,UCB=*ucb addr*
specifies the address of a fullword on a fullword boundary containing the address of a unit control block (UCB).

,OUTPUT=*output addr*
specifies the address of a fullword on a fullword boundary that will contain the subsystem identification word (SID) upon completion.

The SID is a fullword value that identifies the subchannel. The first halfword is X'0001', and the last halfword contains the subchannel number.

The output address must reside in 31-bit addressable storage.

,RTNCODE=*rtncode addr*
specifies the address of a fullword on a fullword boundary that will contain the return code upon completion.

The return code address must reside in 31-bit addressable storage.

After completion, the contents of the registers are as follows:

- Register 0 is used as a work register by the macro.
- Register 1 (unless the return code is 4) contains the SID.
- Registers 2-13 are unchanged.
- Register 14 is used as a work register by the macro.
- Register 15 contains a return code.

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

Hexadecimal Code	Meaning
00	The address specified on the OUTPUT parameter contains the SID.*
04	The UCB was disassociated from the subchannel at the time of the IOSINFO service routine invocation.

* In some cases, the subchannel number in the SID might not be valid. Any disassociation of the UCB and the subchannel means the subchannel number in the SID is not valid. If the UCB is disassociated from the subchannel after the IOSINFO service routine invocation, no notification can be given.

Example 1

Operation: Obtain the subchannel number for a UCB whose address is in register 1. Specify the SID output to be placed in register 2 and the return code to be placed in register 3.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=(1),OUTPUT=(2),RTNCODE=(3)
```

Example 2

Operation: Obtain the subchannel number for a UCB whose address is in location ADDR. Specify the SID output to be placed in location ADDX and the return code to be placed in register 3.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=ADDR,OUTPUT=ADDX,RTNCODE=(3)
```

Example 3

Operation: Obtain the subchannel number for a UCB whose address is in register 2. Specify the SID output to be placed in register 3 and the return code to be placed in location ADDR.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=(2),OUTPUT=(3),RTNCODE=ADDR
```

IOSLOOK — Locate Unit Control Block

The IOSLOOK macro locates the unit control block (UCB) associated with a device number. To use IOSLOOK, you must be executing in supervisor state. Register 13 must point to a 16-word save area where the macro stores registers 0 through 15 at offset 0. You must also include a DSECT for both the CVT (using the CVT mapping macro) and the IOCOM (using the IECDIOCM mapping macro).

The IOSLOOK macro is written as follows:

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

DEV=*(reg)*	*reg:* Register (0) - (12), (14), (15). **Default:** DEV=(6).
,UCB=*(reg)*	*reg:* Register (0) - (12). **Default:** UCB=(7).

The parameters are explained as follows:

DEV=*(reg)*
specifies a general purpose register, symbolic or absolute, that contains the hexadecimal device number, right justified. If this parameter is omitted, register 6 is assumed.

,UCB=*(reg)*
specifies a general purpose register, symbolic or absolute, that will be used to return the address of the UCB common segment. If this parameter is omitted, register 7 is assumed.

If the UCB address cannot be found, then the contents of this register are unpredictable.

Note: The UCB must reside in 24-bit addressable storage.

When control returns, register 15 contains one of the following return codes.

Hexadecimal Code	Meaning
00	UCB address was found
04	Device number was invalid or no UCB exists.

Example

Operation: Find the UCB address for device 250. Register 2 contains the value X'00000250'. The UCB address is to be returned in register 5 and UCBPTR is equated to 5.

```
IOSLOOK DEV=(2),UCB=(UCBPTR)
```

ITTFMTB — Generate Component Trace Format Table

ITTFMTB generates a table called the component trace format table. It can also generate a map of the table. IPCS uses this table to control the formatting of trace data for program events that occur when the system runs. When you use ITTFMTB to generate information in the table, you are specifying the formatting style of the trace data. For information on IPCS, see *IPCS User's Guide* and *IPCS Planning and Customization*.

Invoke the macro once to define the beginning of the table and once to define the end of the table. In between, you can invoke the macro repeatedly to define the individual formats for the various traceable events.

The ITTFMTB macro is written as follows:

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

MAP TABLEDATA=*tabname* EVENTDATA=*eventid* TABLEEND	Required choice. Select one of four options. *tabname*: Symbol up to eight characters long. *eventid*: A-type address.
,ENTRYLENGTH=*elength*	Optional with TABLEDATA and not otherwise allowed. *elength*: A-type address.
,LOCBUFNAME=*bufname* ,LOCBUFADDR=*bufaddr*	Required choice with TABLEDATA and not otherwise allowed. *bufname*: Symbol up to eight characters long. *bufaddr*: A-type address.
,FILTERNAME=*pgmname* ,FILTERADDR=*pgmaddr*	Optional choice with TABLEDATA and not otherwise allowed. *pgmname*: Symbol up to eight characters long. *pgmaddr*: A-type address.
,MNEMONIC=*mnemonic*	Required with EVENTDATA and not otherwise allowed. *mnemonic*: Symbol up to 32 characters long.
,DESCRIPTION=*text*	Required with EVENTDATA and not otherwise allowed. *text*: Symbol up to 32 characters long.
,MODELNAME=*modelname* ,MODELADDR=*modeladdr*	Optional choice with EVENTDATA and not otherwise allowed. *modelname*: Symbol up to eight characters long. *modeladdr*: A-type address.
,FORMATNAME=*pgmname* ,FORMATADDR=*pgmaddr*	Optional choice with EVENTDATA and not otherwise allowed. *pgmname*: Symbol up to eight characters long. *pgmaddr*: A-type address.
,OFFSETASID=*(ids)*	Optional with EVENTDATA and not otherwise allowed. *ids*: One or more A-type addresses, separated by commas.
,OFFSETJOBNAME=*(offsets)*	Optional with EVENTDATA and not otherwise allowed. *offsets*: One or more A-type addresses, separated by commas.
,VIEWSUMMARY=*scode*	Optional with EVENTDATA and not otherwise allowed. *scode*: A-type address.
,VIEWFULL=*fcode*	Optional with EVENTDATA and not otherwise allowed. *fcode*: A-type address.

,COMPONENTDATA=*cdata*	Optional with EVENTDATA and not otherwise allowed. *cdata*: A-type address.
,EXCEPTION ,NOEXCEPTION	Optional choice with EVENTDATA and n . otherwise allowed.

The parameters are explained as follows:

name
Is an optional 1 to 8 alphanumeric character input string, start'ng in column 1, that is the assembler label on the ITTFMTB macro.

MAP
Specifies that a map of a format table is to be generated.

TABLEDATA=*tabname*
Specifies that the definition of an initialized format table is to be started. When you specify TABLEDATA, you also specify the name to be associated with the table and certain data that appears only once in the table.

EVENTDATA=*eventid*
Specifies the event identifier that is associated with a component trace event.

TABLEEND
Specifies the end of the definition of the format table.

,LOCBUFNAME=*bufname*
Specifies the name of the locate buffer exit routine that is loaded by the IPCS CTRACE subcommand. IPCS calls this routine to locate a component's trace buffers in a dump.

,LOCBUFADDR=*bufaddr*
Specifies the address of the locate buffer exit routine. IPCS calls this routine to locate a component's trace buffers in a dump.

,FILTERNAME=*pgmname*
Specifies the name of the component filter exit routine that is loaded by the IPCS CTRACE subcommand. IPCS calls this routine to provide component-specific filtering for that component's trace entries. No component filter exit is supplied if you do not specify one.

,FILTERADDR=*pgmaddr*
Specifies the address of the component filter exit routine. IPCS calls this routine to provide component-specific filtering for that component's trace entries. No component filter exit is supplied if you do not specify one.

,ENTRYLENGTH=*elength*
When *elength* is not zero, this parameter specifies the length of the fixed-length component trace entries that the component maintains. When *elength* is zero, it indicates that the component trace entries vary in length. A default of zero is assumed.

,MNEMONIC=*mnemonic*
Specifies a mnemonic name for the type of event being described. This name is the first information to be formatted on a line associated with an event entry of this type. The name permits the reader of formatted component traces to rapidly scan the output for patterns of events and events of particular interest.

,DESCRIPTION=*text*
Specifies descriptive, literal text to be associated with the type of trace entry being described. When this type of trace entry is formatted, the text appears at the end of the first line of the output. It helps the reader of the output to understand the significance of an entry, without having to access separate reference materials.

,MODELNAME=*modelname*
Specifies the name of the model that is to be used to format this trace entry. No model is used if MODELNAME or MODELADDR is not specified.

,MODELADDR = *modeladdr*

Specifies the address of the model to be used to format this trace entry. No model is used if MODELADDR or MODELNAME is not specified.

,FORMATNAME = *pgmname*

Specifies the name of the formatter routine that formats this trace entry. No formatter routine is called if FORMATNAME or FORMATADDR is not specified.

,FORMATADDR = *pgmaddr*

Specifies the address of the formatter routine that formats this trace entry. No formatter routine is called if FORMATADDR or FORMATNAME is not specified.

,OFFSETASID = *(ids)*

If you want ASID filtering to be performed (as requested by an IPCS CTRACE subcommand), use this parameter to specify the offsets to the ASID fields. The ASID fields occur at various offsets in the trace entry. Specify up to 5 offsets. An offset value may not exceed decimal 65,535. If you do not specify OFFSETASID, ASID filtering is not performed

,OFFSETJOBNAME = *(offsets)*

If you want job name filtering to be performed (as requested by an IPCS CTRACE subcommand), use this parameter to specify the offsets to the job name fields. The job name fields occur at various offsets in the trace entry. Specify up to 5 offsets. An offset value may not exceed decimal 65,535. If you do not specify OFFSETJOBNAME, job name filtering is not performed.

,VIEWSUMMARY = *scode*

Specifies the halfword view that the model processor uses to format summary fields from the trace entry. A default of X'8000' for *scode* is used if you do not specify this parameter.

,VIEWFULL = *fcode*

Specifies a halfword view (used by model processor) to format all fields from the trace entry. A default of X'0200' for *fcode* is used if you do not specify this parameter.

,COMPONENTDATA = *cdata*

This parameter is reserved for use by the component. If this parameter is not specified, a default of zero is assumed for *cdata* indicating that no component data is associated with the trace entry.

,EXCEPTION
,NOEXCEPTION

EXCEPTION specifies that this trace entry records an exceptional event. When the IPCS CTRACE subcommand is invoked with the EXCEPTION filtering option, only trace entries with the EXCEPTION attribute are formatted.

NOEXCEPTION specifies that the trace entries being described record normal events. These entries will not be formatted when the IPCS CTRACE subcommand is invoked with the EXCEPTION of the filtering option. The default is NOEXCEPTION.

LLACOPY — Library Lookaside Refresh

The LLACOPY macro obtains new PDS directory entries from DASD and uses them to synchronously refresh the LLA directory. LLACOPY returns the new directory entries to the caller. LLACOPY uses BLDL list entries to update the LLA directory.

The requirements for the caller are:

Authorization:	Supervisor state, key 0
Dispatchable unit mode:	Task mode
Cross memory mode:	PASN = HASN = SASN
Amode:	24-bit or 31-bit addressing mode
ASC mode:	Primary
Serialization:	Enabled, unlocked
Control parameters:	Not applicable

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

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

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

The parameters are explained as follows:

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

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

,RETCODE= *ret code*
specifies the output variable into which the system copies the return code from general purpose register 15.

,RSNCODE= *rsn code*
specifies the output variable into which the system copies the reason code from general purpose register 0.

,MF=S
specifies the standard form of LLACOPY. The standard form places the parameters into an in-line parameter list.

When LLACOPY returns control to the system, register 15 contains one of the following return codes:

Code	Meaning
0	LLACOPY found all requested directory entries and copied the new entries into the caller's BLDL directory. If LLA was available, LLACOPY refreshed the LLA directory for the given members in the PDS concatenation that the open DCB defined.
4	LLACOPY did not find all the requested directory entries, and may not have found any entries. It copies into the caller's BLDL list entries which it did find. If LLA was available, LLACOPY refreshed the LLA display for the entries which it found, and removed from the LLA directory any members whose directory entries it did not find.
8	LLACOPY failed because of either an I/O error or storage constraints. LLACOPY does not update the BLDL directory or refresh the LLA directory.

When LLACOPY returns control to the system, and you have received a return code of 8, register 0 may contain one of the following reason codes:

Code	Meaning
0	LLACOPY detected a permanent I/O error when trying to search the directory.
4	LLACOPY did not have sufficient virtual storage to complete.

Example

Operation: Request LLACOPY to use the DCB you define when it performs a BLDL in your address space.

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

USERDCB  DCB  DDNAME=LLACOPY,MACRF=R,DSORG=PO
B_LIST   DS   CL76                            BLDL LIST
RETNCODE DS   F
RSONCODE DS   F
```

LLACOPY (List Form)

The list form of the LLACOPY macro constructs a nonexecutable control program parameter list.

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

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

,MF=(L,*cntl addr*) ,MF=(L,*cntl addr, parml attr*)	*cntl addr:* symbol. *parml attr:* 1- to 60-character string. Default: 0D

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

,MF=(L,*cntl addr*)
,MF=(L,*cntl addr, parml attr*)
specifies the list form of LLACOPY. *cntl addr* defines the area into which the system stores the parameter list.

parml attr, which is optional, defines a character string up to 60 characters long. It contains any special attributes for the parameter list. The default is 0D.

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

LLACOPY (Execute Form)

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

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

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

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

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

,MF=(E,*cntl addr*)
specifies the execute form of LLACOPY. *cntl addr* identifies the location of the parameter list.

LOAD — Bring a Load Module into Virtual Storage

The LOAD macro is used to bring the load module containing the specified entry name into virtual storage, if a usable copy is not available in virtual storage. Load services places the load module in storage above or below the 16 megabytes line depending on the RMODE of the module, which is specified in the directory entry for the module.

The responsibility count for the load module is increased by one. On output, the high-order byte of register 1 contains the authorization code of the loaded module and the low-order three bytes contain the module's length in doublewords. Control is *not* passed to the load module; instead, the virtual storage address and the addressing mode of the designated entry point is returned in register 0. The load module remains in virtual storage until the responsibility count is reduced to 0 through task terminations or until the effects of all outstanding LOAD requests for the module have been canceled (using the DELETE macro described in *Application Development Macro Reference*), *and* there is no other requirement for the module.

Load sets the high-order bit of the entry point address in register 0 to indicate the module's AMODE, which is obtained from the directory entry for the module. If the module's AMODE is 31-bit, it sets the indicator to 1; if the module's AMODE is 24-bit, it sets the indicator to 0; and if the module's AMODE is ANY, it sets the indicator to correspond to the caller's AMODE.

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

The entry name in the load module 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 entry name was previously specified in an IDENTIFY macro, no attempt is made to bring in an additional copy of the module. If the specified entry name cannot be located, the task is abnormally terminated.

The LOAD macro is written as follows:

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

EP=*entry name* EPLOC=*entry name addr*	*entry name:* symbol. **entry name addr:* RX-type address or register (2) - (12); A-type
DE=*list entry addr*	address or register (2) - (12). **list entry addr:* RX-type address, or register (2) - (12); A-type address or register (2) - (12).
,DCB=*dcb addr*	*dcb addr: RX-type address, or register (1) or (2) - (12); A-type address or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr:* RX-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default:** LSEARCH=NO
,ADDR=*load addr* ,ADRNAPF=*load addr*	*load addr:* A-type address or register (2) - (12).

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

* If you code any of the parameters: LSEARCH, ADDR, ADRNAPF, GLOBAL, EOM, or LOADPT, you will obtain a macro-generated parameter list. Therefore, except for the error routine address, all addresses must be specified as A-type addresses or registers (2) - (12).

The parameters are explained below:

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

specifies the entry name, the address of the name, or the address of the name field in a 60-byte list entry for the entry name that was constructed using the BLDL macro. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

Note: When you use the DE parameter with the LOAD macro, DE specifies the address of a list that was created by a BLDL macro. The LOAD and the BLDL must be issued from the same task. Otherwise, the system might terminate the program with an abend code of 106 and a return code of 15. Therefore, do not issue an ATTACH or DETACH between issuances of BLDL and LOAD.

,DCB=*dcb addr*

specifies the address of the data control block for the partitioned data set containing the entry name described above. This parameter must indicate the same DCB used in the BLDL mentioned above.

If the DCB parameter is omitted or if DCB=0 is specified when the LOAD macro is issued by the job step task, the data sets referred to by either the STEPLIB or JOBLIB DD statement are first searched for the entry name. If the entry name is not found, the link library is searched.

If the DCB parameter is omitted or if DCB=0 is specified when the LOAD macro is issued by a subtask, the data sets associated with one or more data control blocks referred to by the TASKLIB operand of previous ATTACH macros in the subtask chain are first searched for the entry name. If the entry name is not found, the search is continued as if the LOAD had been issued by the job step task.

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

,ERRET=*err rtn addr*

specifies the address of a routine to receive control when an error condition that would cause an abnormal termination of the task is detected. Register 1 contains the abend code that would have resulted had the task abended, and register 15 contains the reason code that is associated with the abend. The routine does not receive control when input parameter errors are detected.

,LSEARCH=NO
,LSEARCH=YES

specifies whether (YES) or not (NO) you want the library search limited to the job pack area and to the first library in the normal search sequence.

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

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

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

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

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

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

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

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

Note: A load request with the GLOBAL option does not cause the loaded module to be implicitly known to other address spaces. The loaded module can be accessed by other address spaces, however, only the requesting task is accountable for it (and may therefore delete it).

,EOM=YES
,EOM=NO

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

,LOADPT = *addr*

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

,RELATED = *value*

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

Example 1

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

```
LOAD  EP=PGMLKRUS
```

Example 2

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

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

LOAD (List Form)

The list form of the LOAD macro builds a non-executable parameter list that can be referred to by the execute form of the LOAD macro.

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

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* A-type address. *list entry addr:* A-type address.
,DCB=*dcb addr*	*dcb addr:* A-type address.
,LSEARCH=NO ,LSEARCH=YES	**Default:** LSEARCH=NO
,ADDR=*load addr* ,ADRNAPF=*load addr*	*load addr:* A-type address.
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default:** GLOBAL=NO If GLOBAL=YES is specified, the default GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default:** EOM=NO **Note:** GLOBAL must be specified with EOM=YES.
,LOADPT=*addr*	*addr:* A-type address. **Note:** ADDR and ADRNAPF cannot be specified with LOADPT.
,RELATED=*value*	*value:* any valid macro keyword specification.
,SF=L	

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

,SF=L
specifies the list form of the LOAD macro.

LOAD (Execute Form)

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

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

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* RX-type address or register (2) - (12). *list entry addr:* RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr:* RX-type address, or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr:* RX-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default:** LSEARCH=NO
,ADDR=*load addr* ,ADRNAPF=*load addr*	*load addr:* RX-type address or register (2) - (12). **Note:** For an RX-type address, the operand is treated as the address of a field that contains the actual load address.
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default:** GLOBAL=NO **Note:** If GLOBAL=YES is specified, the default is GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default:** EOM=NO **Note:** GLOBAL must also be specified with EOM=YES.
,LOADPT=*addr*	*addr:* RX-type address or register (2) - (12). **Note:** ADDR and ADRNAPF cannot be specified with LOADPT.
,RELATED=*value*	*value:* any valid macro keyword specification.
,SF=(E,*list addr*)	*list addr:* RX-type address or register (2) - (12) or (15).

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

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

LOCASCB — Locate ASCB

The LOCASCB macro is used to locate the ASCB address associated with a specified ASID or STOKEN. If the caller is concerned that the ASCB might terminate while being referenced, the caller should provide serialization to prevent ASCB termination by holding the CMS lock.

Programs executing in cross memory mode can invoke the LOCASCB macro. When using the ASID parameter, the program must be in primary or secondary ASC mode. For the STOKEN parameter, the program must be in primary or in AR ASC mode.

For callers in AR mode, the LOCASCB parameter list can be located in any addressable address space.

The LOCASCB macro uses registers 0, 1, 14, and 15, and is written as follows:

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

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

The parameter is explained as follows:

ASID = *asid addr*
specifies the RX-type address of a halfword that contains the ASID for which the ASCB is to be returned or the register that contains the ASID in bits 16-31. (Bits 0-15 of the register are ignored.) If the caller specifies (1), the ASID need not be copied into register 1 by the macro expansion.

When LOCASCB returns control, register 1 contains the results of the locate operation as follows:

- If register 1 is positive, it contains the ASCB address.
- If register 1 is negative or zero, the specified ASID is invalid.

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

When LOCASCB returns control, general purpose register 1 contains the results of the locate operation as follows:

- If the value is not zero, it is the ASCB address.
- If the value is zero, the specified STOKEN is invalid.

When LOCASCB returns control to an AR mode caller, the general purpose register/access register 1 pair contains the ASCB address.

Return Codes

If you specified the ASID parameter, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Macro was successful.
04	ASID was invalid.

If you specified the STOKEN parameter, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Macro was successful.
04	STOKEN did not map to a valid ASID. Causes might be that the STOKEN you specified identifies a data space; or the address space represented by the STOKEN has terminated.
08	STOKEN was invalid

LXFRE — Free a Linkage Index

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

The requestor must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed to this macro must also be addressable in primary mode when the macro is issued.

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

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

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

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

The parameters are explained as follows:

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

,FORCE=NO
,FORCE=YES
specifies whether (YES) or not (NO) the linkage index is to be freed even if entry tables are currently connected to it. Any connected entry tables are disconnected before the linkage index is freed. FORCE=NO is the default.

,RELATED = *value*

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

When LXFRE returns control, register 15 contains contains one of the following return codes:

Hexadecimal Code	Meaning
0	The specified linkage indexes were freed. No entry tables were connected.
4	The specified linkage indexes were freed. Entry tables were connected, but FORCE was specified and was successfully executed.
8	Some of the specified linkage indexes were freed. Entry tables were connected. FORCE was specified but one or more of the necessary disconnects failed. No action by the issuer of LXFRE is required in this situation.

LXFRE (List Form)

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

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

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

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

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

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

LXFRE (Execute Form)

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

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

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

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

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

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

LXRES — Reserve a Linkage Index

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

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

The requestor must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed to the LXRES macro must also be addressable in primary mode at the time the macro is issued.

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

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

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

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

The parameters are explained as follows:

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

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

,RELATED = *value*

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

On return, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The specified linkage indexes were successfully reserved.

LXRES (List Form)

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

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

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

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

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

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

LXRES (Execute Form)

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

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

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

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

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

,MF=(E,*cntl addr*)
specifies the execute form of the LXRES macro and *cntl addr* is the name or address of the list form of the macro.

MGCR — Internal START or REPLY Command

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

The caller must have PSW key 0-7, and must include mapping macro IEZMGCR.

The MGCR macro is written as follows:

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

The parameters are explained as follows:

command-buffer-address
specifies the address of a command buffer (mapped by the IEZMGCR macro) that contains the following information.

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

Notes:

1. Register 0 must contain zero.
2. The command buffer can be located in 24-bit or 31-bit addressable storage.
3. A program token is meaningful only with the START command.

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

Hexadecimal Code	Meaning
00	START command processed successfully. Register 0 contains the right-justified ASID of the started address space.
08	START command failed.

Example 1

Operation: Issue an internal START command for the catalogued procedure labeled PROG. In this example, security tokens are defined, which are assumed to be set elsewhere prior to issuing the MGCR.

```
         SR    R0,R0              Clear register 0
         MGCR  INPUT              Issue macro with parameter list
*                                 defined at label INPUT
         .
         .
         .
INPUT    DS    0H
SECYUSE  DC    X'80'              Security tokens present
LENGTH   DC    AL1(PTOKEN-INPUT)  Length of input command
SECFLGS  DC    X'0808'            Security token flags
CMD      DC    C'S PROG'          The actual input command
PTOKEN   DC    AL4(DATA)          Security token
UTOKEN   DC    CL80               UTOKEN
```

Example 2

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

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

MODESET — Change System Status

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

The form that generates inline code can execute in supervisor or problem program state. If a problem state caller's key is marked as authorized in the PSW-key mask the inline form can execute in problem state. The inline form can be used by programs executing in cross memory mode. If the key you specify is TCB, RBT1, or RBT234, you must also ensure that current addressability is to the home address space.

The form that generates an SVC is executable by users in supervisor state, under PSW key 0-7, or APF-authorized. The SVC form cannot be used in cross memory mode.

The macro does not generate any return codes.

Inline Code Generation

The standard form of the MODESET macro that generates inline code is written as follows:

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

EXTKEY=*key* KEYADDR=*new key addr* KEYREG=*new key reg*	*key:* one of the following: ZERO KEY2 TCB KEY3 RBT1 KEY4 RBT234 KEY7 *new key addr:* RX-type address or register (2). *new key reg:* register 1-15 without parentheses; may be symbolic.
,SAVEKEY=*old key addr*	*old key addr:* RX-type address or register (2). **Notes:** 1. If KEYADDR=(2) is specified above, then SAVEKEY=(2) cannot be specified. 2. The WORKREG parameter is required if SAVEKEY=A-type address is specified. 3. If WORKREG and SAVEKEY are specified with KEYREG, the KEYREG register should be different from the WORKREG register. Also, if SAVEKEY is specified with KEYREG, the KEYREG register should not be register 2.
,WORKREG=*work reg*	*work reg:* decimal digits 0-15 without parentheses. **Notes:** 1. WORKREG is required if the following are specified: EXTKEY=TCB EXTKEY=RBT234 EXTKEY=RBT1 KEYADDR=A-type address 2. The WORKREG parameter should be register 1-15 if one of these four parameters is specified because WORKREG is used as a base register on the SPKA instruction. WORKREG=0 sets the PSW key to zero.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

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

ZERO - Key of zero is to be set.

TCB - Key is to be obtained from the caller's TCB.

RBT1 - Key from the active RB of type 1 SVC routine issuing MODESET.

RBT234 - Key from the active RB preceding SVRB of type 2, 3, or 4 SVC routine issuing MODESET.

KEY2 - Key of 2 is to be set.

KEY3 - Key of 3 is to be set.

KEY4 - Key of 4 is to be set.

KEY7 - Key of 7 is to be set.

KEYADDR = *new key addr*
specifies a location 1 byte in length which contains the key in bit positions 0-3. If register (2) is specified, the key is contained in bit positions 24-27 (bits 28-31 are ignored). This parameter permits a previously saved key to be restored. If TCB, RBT1 or RBT234 is specified as the key address, the TCB mapping macro IKJTCB is required. The user is expected to establish addressability to the TCB with a USING statement.

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

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

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

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

SVC Generation

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

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

The parameters are explained as follows:

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

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

Example 1
Operation: Save the current PSW key, and change the key to that of the active TCB.

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

Example 2
Operation: Change to supervisor mode and key zero.

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

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

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

MODESET (List Form)

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

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

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

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

MODESET (Execute Form)

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

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

RELATED=*value*,	*value:* any valid macro keyword specification.
MF=(E,*list addr*)	*list addr:* RX-type address, or register (1).

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

MF=(E,*list addr)*
specifies the execute form of the MODESET macro, using a parameter list address.

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

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

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

The NIL macro is written as follows:

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

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

The parameters are explained as follows:

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

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

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

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

Example

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

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

NUCLKUP — Nucleus Map Lookup Service

The NUCLKUP macro can be used either to retrieve the address and AMODE of a nucleus CSECT or ENTRY or to retrieve the name and address of the nucleus CSECT, which is pointed to by a given address within the CSECT.

This macro runs in the key and state of the caller. On entry to this macro, register 13 must point to a 72-byte register save area. Users must include the CVT mapping macro.

The NUCLKUP macro is written as follows:

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

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

The parameters are explained as follows.

BYNAME
BYADDR
specifies the function to be performed. If BYNAME is specified, the user supplies the name of a CSECT or ENTRY and receives the address and AMODE of that CSECT or ENTRY. If BYADDR is specified, the user supplies an address within a CSECT and receives the name and address of the CSECT.

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

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

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

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

When control is returned, the registers contain the following information:

Register	Meaning
0	For a BYNAME request, the address and AMODE of the CSECT or ENTRY; for a BYADDR request, the 31-bit address of the CSECT
1	For a BYNAME request, the high-order byte is zero and the low-order three bytes contain the length from the entry point to the end of the CSECT; for a BYADDR request, unchanged
2-14	Unchanged
15	Return code

The return codes in register 15 are as follows:

Hexadecimal Code	Meaning
0	The request was satisfied.
4	The request was not satisfied. For a BYNAME request, the name was not found and the location containing the address was set to zero. For a BYADDR request, the address was not found in the nucleus and the location containing the name was set to zero.
8	The request was not satisfied because the type of request was not specified correctly. The locations containing the name and address were set to zero.

Example

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

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

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

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

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

The OIL macro is written as follows:

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

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

The parameters are explained as follows:

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

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

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

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

Example

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

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

OUTADD — Create Output Descriptor

Use the OUTADD macro to create an output descriptor for a system output (sysout) data set. For information about using the OUTADD macro, see "Dynamic Output" in *SPL: Application Development Guide*.

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

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

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

MF=L

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

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

MF=L
specifies the list form of the OUTADD macro.

Example

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

```
PARML    OUTADD  MF=L
```

OUTADD (Execute Form)

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

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

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

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

The parameters are explained as follows:

NAME=*descriptor name addr*
specifies the address of an eight-character field. This field contains the the name of the output descriptor that is to be added. It is mutually exclusive with SYSNAME. NAME or SYSNAME must be specified.

SYSNAME=*descriptor name addr*
specifies the address of an eight-character field that a system-generated output descriptor name is to be returned in. SYSNAME is mutually exclusive with NAME. SYSNAME or NAME must be specified.

,TEXTPTR=*tu pointer addr*
specifies the address of the text unit pointer list. It is a required parameter.

,ENQ=CONDITIONAL
,ENQ=UNCONDITIONAL
specifies whether the create request is to be conditional or unconditional.

,MF=(E,*list addr***)**
specifies the execute form of the OUTADD macro. *list addr* is the address of the parameter list.

Example

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

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

OUTDEL — Delete Output Descriptor

Use the OUTDEL macro to delete an output descriptor for a system output (sysout) data set. For information about using the OUTDEL macro, see "Dynamic Output" in *SPL: Application Development Guide*.

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

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

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

,MF=L

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

name
specifies the symbolic name to be associated with the storage declaration generated by the list form. It is a required parameter.

,MF=L
specifies the list form of the OUTDEL macro.

Example

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

```
PARML    OUTDEL  MF=L
```

OUTDEL (Execute Form)

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

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

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

NAME=*descriptor name addr*	*descriptor name addr*: Rx-type address or register (2)-(12).
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2)-(12).

The parameters are explained as follows:

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

,MF=(E,*list addr***)**
specifies the execute form of the OUTDEL macro. *list addr* is the address of the parameter list.

Example

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

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

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

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

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

Note: Do not issue PCLINK for a routine that receives control as the result of a stacking PC instruction.

STACK Option of PCLINK

To use PCLINK STACK you must be in primary mode and supervisor state. You must not change registers 13-4 between the time you get control and the time you issue PCLINK STACK.

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

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

STACK	
,INKEY=ZERO	
,OUTKEY=CALLER	**Default:** OUTKEY=CALLER
,OUTKEY=ZERO	
,OUTKEY=KEY*n*	*n:* Any valid PSW key value from 0-F.
,SAVE=YES	**Default:** SAVE=YES
,SAVE=NO	
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

STACK
saves some of the environment when a routine gets control as a result of a basic PC instruction. STACK is a required parameter.

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

,OUTKEY = CALLER
,OUTKEY = ZERO
,OUTKEY = KEY*n*

specifies the setting of the PSW key after the PCLINK macro has completed. Specifying CALLER causes the PSW key to be restored to the value it had on entry. Specifying ZERO sets the PSW key to zero. Specifying a key value indicates a specific value for the key. You may specify any key value from 0 to F.

,SAVE = YES
,SAVE = NO

specifies whether (YES) or not (NO) to preserve registers 8 - 12. The save area used is different from the area addressed by register 13. SAVE = YES is the default. Processing is more efficient if you code SAVE = NO.

,RELATED = *value*

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

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

R0, R1	Unchanged
R2	Bits 0-23 contain bits 8-31 from register 2 at the time the macro was issued. Bits 24-31 contain the PCLINK caller's PSW key.
R3, R4	Unchanged
R5	Linkage register to return from PCLINK STACK
R6, R7	Unchanged
R8-R12	Unchanged if SAVE = YES Unpredictable if SAVE = NO
R13	0, to ensure that the first save area created after the basic PC does not point to a previous save area.
R14	Stack token to uniquely identify the stack entry created. This token is required for the UNSTACK and EXTRACT forms of PCLINK.
R15	Unchanged

UNSTACK Option of PCLINK

To use PCLINK UNSTACK, you must be in supervisor state. In addition, if you specify PCLINK UNSTACK,THRU and the token contained in the specified register indicates the stack element most recently queued for that unit of work, you must be in primary mode and the PASID must be the same as when the stack element was created.

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

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

UNSTACK	
,THRU=*(reg)* ,TO=*(reg)* ,PURGE=YES	*reg:* Register (0) - (15).
,INKEY=ZERO	
,OUTKEY=STACK ,OUTKEY=ZERO	**Default:** OUTKEY=STACK
,SAVE=YES ,SAVE=NO	**Default:** SAVE=YES
,ERRET=*addr*	*addr:* RX-type address or register (0) - (13) or (15).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

UNSTACK
restores the environment before the routine issues a PT instruction to return control to the calling routine. UNSTACK is a required parameter.

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

Processing is more efficient if you issue a separate PCLINK UNSTACK,THRU for each stack element you want to dequeue rather than unstacking several elements at a time.

If the token you specify represents the most recently enqueued stack element, the PASID when UNSTACK,THRU is issued must be the same as the PASID when PCLINK STACK was issued for that element.

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

R0-R1	Unchanged
R2	Bits 24-27 contain the PSW key from the stack element identified by the token
R3	As saved by PCLINK STACK
R4-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R13,R14	As saved by PCLINK STACK
R15	Unchanged

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

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

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

R0,R1	Unpredictable
R2	Unchanged if INKEY=ZERO is specified and ERRET is not specified, otherwise, PSW key of PCLINK caller
R3-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R13	Unchanged
R14-R15	Unpredictable

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

The ERRET parameter cannot be used with PURGE.

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

R0,R1	Unpredictable
R2	Unchanged if INKEY=ZERO is specified, otherwise PSW key of PCLINK caller
R3-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R13	Unchanged
R14-R15	Unpredictable

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

,OUTKEY = STACK
,OUTKEY = ZERO

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

This parameter is valid only with PCLINK UNSTACK,THRU.

,SAVE = YES
,SAVE = NO

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

,ERRET = *addr*

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

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

R0,R1,R3,R13	Unpredictable
R2	PSW key of PCLINK caller
R4-R7	Unchanged
R8-R12	Unchanged if SAVE = YES is specified Unpredictable if SAVE = NO is specified
R14	The token passed as input
R15	4 - stack was empty 8 - input token is invalid 12 - an address on the queue is invalid 16 - an ASID on the queue is invalid 20 - Unknown error

,RELATED = *value*

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

EXTRACT Option of PCLINK

To use PCLINK EXTRACT, you must either be in PSW key 0, supervisor state, or have a PSW key mask authorized for key 0.

In addition, you must have addressability to the same address space as when PCLINK STACK was issued for the stack element from which you are extracting data.

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

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

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

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

The parameters are explained as follows:

EXTRACT
retrieves information from the saved environment. EXTRACT is a required parameter.

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

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

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

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

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

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

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

,LP = *(reg)*
specifies a register into which the latent parameter pointer is to be placed.

,ENTRY = *(reg)*
specifies a register into which the contents of register 5 as established by the PCLINK STACK macro are to be placed. Bit 0 of the register used by the ENTRY parameter specifies the addressing mode of the program call routine that issued the PCLINK macro.

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

PGANY — Page Anywhere

Note: The PGSER macro is the preferred programming interface.

Some fixed pages are assigned within the first 16 megabytes of storage. The system assumes that once a page has been fixed, it is likely to be fixed again. The next time that page is loaded, the system tries to put it in the first 16 megabytes in anticipation of a fix. Use the PGANY macro to indicate to the system that no further page fixes are planned for a particular page and that the next time the page is loaded, the system can put it anywhere.

Entry is by means of an SVC. The caller can be in either problem or supervisor state and must not hold any locks.

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.

On return, register contents are as follows:

R 0-1	Used as work registers by the macro
R 2-14	Unchanged
R 15	Return code

The PGANY macro is written as follows:

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

L,LA=*list addr*	*list addr:* RX-type address or register (1) or (2) - (12).
R,A=*start addr*	*start addr:* RX-type address or register (1), (2) - (12).
,EA=*end addr*	*end addr:* RX-type address or register (15), (2) - (12). **Note:** Cannot be specified unless R is specified. **Default:** EA=start addr + 1.

The parameters are explained as follows:

L

specifies that the virtual subarea list (VSL) is being supplied with this request. (See "Input to Page Services" in *SPL: Application Development Guide* for a description of the virtual subarea list.)

,LA = *list addr*

specifies the address of the virtual subarea list.

R

specifies that the necessary parameters will be passed in registers. A virtual subarea list is not being supplied.

,A = *start addr*

specifies the address of the start of the virtual area.

,EA = *end addr*

specifies the end + 1 byte of the virtual area. If this parameter is not coded, the default is the start address + 1.

Note: *start addr* and *end addr* must be located in 24-bit addressable storage.

Upon completion, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally.
04	Parameter error, X'171' abend, operation terminated because of invalid address in VSL entry.
10	Parameter error, X'171' abend, operation terminated abnormally because the VSL list was invalid.
14	Environmental error, X'028' abend.

For return codes 04 and 10, registers are loaded before the abend as follows:

R0	Unpredictable
R1	Abend code
R2-R10	Unpredictable
R11	Address of input VSL list or 0 for R-form
R12	0 (ECB address = 0)
R13-R14	Current VSL entry being processed
R15	Return code

PGFIX — Fix Virtual Storage Contents

Note: The PGSER macro is the preferred programming interface.

The PGFIX macro makes virtual storage areas, below 16 megabytes, resident in central (also called real) storage and ineligible for page-out while the requesting task's address space is swapped into central storage. PGFIX ignores requests to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by central storage below 16 megabytes. A subsequent PGFREE is effective only if issued by the same task. The PGFIX function is available only to authorized users.

PGFIX does not prevent pages from being paged out when an entire address space is swapped out of central storage. Consequently, when using the PGFIX macro, you can not assume a constant real address mapping for fixed pages that are susceptible to swapping.

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

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

R L	
,LA=*list addr*	*list addr:* A-type address, or register (1) or (2) - (12).
,A=*start addr*	*start addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,EA=*end addr*	*end addr:* A-type address, or register (2) - (12) or (15). **Default:** *start addr* + 1
,LONG=Y ,LONG=N	**Default:** LONG=Y
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N **Note:** RELEASE=Y may only be specified with EA above.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

R
specifies that no parameter list is being supplied with this request.

L
specifies that a parameter list is being supplied with this request.

,LA=*list addr*
specifies the address of the first entry of a virtual subarea list (VSL). See "Input to Page Services" in *SPL: Application Development Guide* for a description of the VSL.

,A=*start addr*
specifies the start address of the virtual area to be fixed.

Note: *start addr* must be located in 24-bit addressable storage.

,ECB = *ecb addr*
specifies the address of the ECB that is used to signal event completion. If the ECB address specified is zero, (ECB = 0 or ECB = (register) where the contents of the register specified is 0), the fix request is completely satisfied before control is returned.

Note: If the user intends to wait on the ECB as part of an ECB list, he must ensure that the list and associated ECBs are fixed in central storage before issuing the WAIT. The PGFIX service routine ensures that the specified ECB is fixed.

,EA = *end addr*
specifies the end address + 1 of the virtual area to be fixed.

Note: *end addr* must be located in 24-bit addressable storage.

,LONG = Y
,LONG = N
specifies that the relative real time duration anticipated for the fix is long (Y) or short (N).

,RELEASE = N
,RELEASE = Y
specifies that the contents of the virtual area is to remain intact (N) or be released (Y) before the fix is done.

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

Upon completion, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally; ECB posted complete.
04	Operation abnormally terminated with a X'171' abend. Operation incomplete because of invalid address virtual subarea list entry; ECB posted complete. See *Message Library: System Codes* for a complete description of the register contents after a X'171' abend.
08	Operation proceeding; ECB will be posted when all requested pages are fixed in central storage.
10	Operation abnormally terminated with a X'171' abend. Virtual subarea list entry or ECB address invalid; no ECB is posted. See *Message Library: System Codes* for a complete description of the register contents after a X'171' abend.

The ECB is unchanged if the request was initiated but not complete (return code 8), or if an ABEND was issued with return code 10. Otherwise, the ECB is posted complete with code:

0 - operation completed successfully.
4 - operation incomplete because of invalid address in VSL entry.

If the return code issued is 8, the ECB is posted asynchronously when paging I/O has completed, with code:

0 - operation completed successfully.
4 - operation incomplete because of paging error; requesting TCB will be abnormally terminated.

The ECB code is posted in the low-order 3 bytes of the ECB, and is right-justified.

Example 1

Operation: Fix a single byte of virtual storage addressed by register 3. Note that the full 4096-byte page containing the specified byte is actually fixed. The storage is long fixed.

```
PGFIX R,A=(R3),ECB=(R5)
```

Example 2

Operation: Fix virtual storage without using a virtual subarea list. Storage is long fixed.

```
PGFIX R,A=(R3),EA=(R4),ECB=ECB1
```

Example 3

Operation: Fix, but not long-fix, virtual storage, and ensure that the pages fully included in the address range are forfeited before fixing the area specified by registers 3 and 4.

```
PGFIX R,A=(R3),EA=(R4),ECB=(R5),LONG=N,RELEASE=Y
```

PGFIXA — Fix Virtual Storage Contents

Note: The PGSER macro is the preferred programming interface.

The PGFIXA macro makes virtual storage areas, below 16 megabytes, resident in central (also called real) storage and ineligible for page-out while the requesting task's address space is swapped into central storage. The PGFIXA function is available only to key zero and supervisor state users. The PGFIXA macro executes short-term, synchronous page fixes. The preferred area(s) of storage are intended for long term page fixes. A long term page fix in the V=R or non-preferred areas may delay V=R functions or CONFIG STORAGE commands. All fix processing is assumed to be short-term and is complete when control is returned to the issuer of the macro.

PGFIXA does not prevent pages from being paged out when an entire address space is swapped out of central storage. Consequently, when using the PGFIXA macro, you cannot assume a constant real address mapping for fixed pages that are susceptible to swapping.

Output

If the PGFIXA is successful, control is returned enabled to the user, all pages are fixed, and register 15 contains a return code of zero.

If the PGFIXA is unsuccessful, the user will be abended with a system completion code of X'171' or a system complete code of X'028'. For X'171' abends, all pages processed up to, but not including the page causing the error, will be fixed. Register 10 will contain the address of the pages in error when the abend is issued. No pages will be fixed in the event of a X'028' abend.

Restrictions

Use of the PGFIXA macro is subject to the following restrictions:

- Can be used only for short term synchronous fixes.
- The user must be in supervisor state with a protection key of zero.
- The user must not hold any spin locks.
- The program mask byte in the PSW is zero and interrupts are enabled upon return from the PGFIXA.
- The user is responsible for freeing any pages fixed via the PGFIXA. A corresponding PGFREEA macro should be issued. In addition, an FRR should be established during the period where fixes are outstanding. The FRR should free the frames in case there is an unexpected error.
- DSECTs for the IHAPSA, CVT, and IHAPVT must be provided.
- The user must ensure that the end address is greater than or equal to the start address.
- The SAVE keyword can only be used with TYPE=R.

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

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

,TYPE=L ,TYPE=R	**Default:** TYPE=R
,SAVE=YES ,SAVE=NO	**Default:** SAVE=YES

The parameters are explained as follows:

TYPE=L
TYPE=R
specifies the type of input. When L is specified, register 1 is to contain the address of a virtual subarea list (VSL) fixed in storage. (See the topic "Input to Page Services" in *SPL: Application Development Guide* for a description of the VSL.) By specifying TYPE=L, registers 1 through 13 are saved. If TYPE=R is specified, then register 1 contains the address of the first byte to be fixed in a contiguous range and register 2 contains the address of the last byte to be fixed (actual end address). When TYPE=R is specified, the registers saved depend upon what is specified on the SAVE parameter.

Note: All other users of the PGFIX, PGFIXA (TYPE=L), and PGFREEA macros must specify the actual end address plus one.

,SAVE=YES
,SAVE=NO
specifies the registers to be saved for TYPE=R. Registers 1 through 13 are saved if SAVE=YES is specified or if the default is taken. Registers 2 through 10 are saved if SAVE=NO is specified.

Example 1
Operation: Use PGFIXA to fix virtual storage without using a virtual subarea list. Registers 2 through 10 will be saved.

```
FIX1 PGFIXA TYPE=R,SAVE=NO
```

Example 2
Operation: Use PGFIXA to fix virtual storage using a virtual subarea list. Registers 1 through 13 will be saved.

```
FIX2 PGFIXA TYPE=L
```

PGFREE — Free Virtual Storage Contents

Note: The PGSER macro is the preferred programming interface.

The PGFREE macro makes virtual storage pages, below 16 megabytes, that were fixed via the PGFIX macro eligible for page-out. The PGFREE function is available only to authorized users. PGFREE must be issued by the same task that issued the PGFIX, otherwise PGFREE has no effect.

Note: A fixed page is not considered pageable until the number of PGFREEs issued for the page is equal to the number of PGFIXes previously issued for that page. That is, a page is not automatically made pageable as the result of issuing a PGFREE macro.

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

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

L	
,LA=*list addr*	*list addr:* A-type address, or register (1) or (2) - (12).
R	
,A=*start addr*	*start addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,EA=*end addr*	*end addr:* A-type address, or register (2) - (12) or (15). **Default:** *start addr* + 1
,ANYWHER=N ,ANYWHER=Y	**Default:** ANYWHER=N
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N **Note:** RELEASE=Y may only be specified with EA above.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

L
specifies that a parameter list is being supplied with this request.

,LA=*list addr*
specifies the address of the first entry of a virtual subarea list (VSL). See "Input to Page Services" in *SPL: Application Development Guide* for a description of the VSL.

R
specifies that no parameter list is being supplied with this request.

,A=*start addr*
specifies the start address of the virtual area to be freed.

Note: *start addr* must be located in 24-bit addressable storage.

,ECB = *ecb addr*
specifies the address of the ECB that was used in a prior PGFIX request. This parameter is used if there is any possibility that the ECB for the previously issued PGFIX was not posted complete.

,EA = *end addr*
specifies the end address + 1 of the virtual area to be freed.

Note: *end addr* must be located in 24-bit addressable storage.

,ANYWHER = N
,ANYWHER = Y
On subsequent page-ins, assign real frames below 16 megabytes in anticipation of a page fix (N) or on subsequent page-ins, assign real frames anywhere (Y). The ANYWHER option takes effect only when the page fix count goes to zero. The default is ANYWHER = N.

,RELEASE = N
,RELEASE = Y
specifies that the contents of the virtual area is to remain intact (N) or be released (Y).

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

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

Hexadecimal Code	Meaning
00	Operation completed normally.
04	Operation abnormally terminated. Operation incomplete because of invalid address in virtual subarea list entry.
10	Operation abnormally terminated. Virtual subarea list entry or ECB address invalid.

Example 1
Operation: Free the storage in Example 1 of standard-form PGFIX.

```
PGFREE R,A=(R3)
```

Example 2
Operation: Free the storage in Example 2 of standard-form PGFIX.

```
PGFREE R,A=(R3),EA=(R4)
```

Example 3
Operation: Free the storage in Example 3 of standard-form PGFIX, and forfeit the pages fully included in the address range.

```
PGFREE R,A=(R3),EA=(R4),ECB=(R5),RELEASE=Y
```

PGFREEA — Free Virtual Storage Contents

Note: The PGSER macro is the preferred programming interface.

The PGFREEA macro makes virtual storage areas, below 16 megabytes, that were fixed by the PGFIXA macro eligible for page-out.

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

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

No additional parameters are specified.

Restrictions

Use of the PGFREEA macro is subject to the following restrictions:

- The issuer of the PGFREEA must provide a fixed virtual subarea list (VSL) or chain of them, pointed to by register 1. For a description of the VSL, see *SPL: Application Development Guide*.
- The user must be in supervisor state, protection key 0.
- The user must provide DSECTs for IHAPSA, CVT, and IHAPVT.

Output

If the PGFREEA is successful, all pages will be freed and register 15 will contain a return code of zero. If unsuccessful, all pages up to, but not including the one that caused the abend will be freed. The user will be abended with a system completion code of X'171'.

PGSER — Page Services

The PGSER macro and its fast path version (see "PGSER — Fast Path Page Services" on page 365) perform the same paging services that PGANY, PGFIX, PGFIXA, PGFREE, PGFREEA, PGLOAD, PGOUT, and PGRLSE perform for addresses below 16 megabytes. PGSER performs these services for addresses either above or below 16 megabytes.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.

The services are:

- Page fix equivalent to PGFIX
- Fast path to fix virtual storage
- Page free equivalent to PGFREE
- Fast path to free virtual storage
- Page load equivalent to PGLOAD
- Page out equivalent to PGOUT
- Page release equivalent to PGRLSE
- Page anywhere equivalent to PGANY

This macro is also described in *Application Development Macro Reference* with the exception of the restricted parameters. The parameters FIX and FREE are restricted to APF-authorized, key zero, or supervisor state callers. The parameters BRANCH=SPECIAL (see "PGSER — Fast Path Page Services" on page 365) and BRANCH=Y are restricted to enabled, supervisor state, key zero callers; users of these options must provide the address of an 18-word save area in register 13. (See "Branch Entry to the PGSER Routine" in *SPL: Application Development Guide* for more information about branch entry.) Also, users of the BRANCH=SPECIAL and BRANCH=Y options must include the CVT and IHAPVT mapping macros. The RELEASE option of the macro is restricted to supervisor state key zero users if the common area is being released. Non-authorized users can release only the private area.

Regardless of the addressing mode, all addresses passed in registers are used as 31-bit addresses. All RX-type addresses are assumed to be in the addressing mode of the caller.

The syntax of the fast path version of PGSER is presented separately following this standard description. The standard form of the PGSER macro is written as follows:

---

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

R
L

,FIX
,FREE
,LOAD
,OUT
,RELEASE
,ANYWHER

,LA=*list addr*

list addr: RX-type address or register (1), (5) - (12) for branch entry; or register (1), (2) - (12) for SVC entry.
Note: This parameter is valid only with L.

,A=*start addr*	*start addr:* RX-type address or register (1), (5) - (12) for branch entry; or register (1), (2) - (12) for SVC entry. **Note:** This parameter is valid only with R.
,EA=*end addr*	**Default:** EA=*start addr* *end addr:* RX-type address or register (2), (5) - (12) for branch entry; or register (15), (2) - (12) for SVC entry. **Note:** This parameter is valid only with R.
,TCB=*tcb addr*	**Default:** TCB=0 *tcb addr:* RX-type address or register (4), (5) - (12). **Note:** This parameter can be specified only if FIX, FREE, LOAD, or OUT and BRANCH=Y are specified.
,ECB=*ecb addr*	**Default:** If FREE or LOAD is specified, ECB=0. **ecb addr:** RX-type address or register (0), (5) - (12) for branch entry; or register (0), (2) - (12) for SVC entry. **Note:** This parameter is required if FIX is specified; is optional if FREE or LOAD is specified; and is invalid for OUT, RELEASE, or ANYWHER. For synchronous page fix the ECB address must be 0.
,RELEASE=Y ,RELEASE=N	**Default:** RELEASE=N **Note:** This parameter may be specified only if FIX, FREE, or LOAD is specified.
,LONG=Y ,LONG=N	**Default:** LONG=Y **Note:** This parameter may be specified only if FIX is specified.
,BACKOUT=Y ,BACKOUT=N	**Default:** BACKOUT=Y **Note:** This parameter may be specified only if FIX is specified.
,KEEPREL=Y ,KEEPREL=N	**Default:** KEEPREL=N **Note:** This parameter may be specified only if OUT is specified.
,ANYWHER=Y ,ANYWHER=N	**Default:** ANYWHER=N **Note:** This parameter may be specified only if FREE is specified.
,BRANCH=Y ,BRANCH=N	**Default:** BRANCH=N
,RELATED=*value*	*value:* any valid macro keyword specification.

R
L

specifies the manner in which the input is supplied. If R is specified, the user supplies the starting and ending addresses of the virtual area for which the service needs to be performed. Before processing the request, page services puts these addresses in registers 1 and 15, respectively. If L is specified, the user supplies the address of the page services list (PSL), which specifies the virtual area for which the service is to be performed. Before processing the request, page services puts the address of the PSL in register 1. See the topic "Input to Page Services" in *SPL: Application Development Guide* for a description of the PSL.

,FIX
,FREE
,LOAD
,OUT
,RELEASE
,ANYWHER

indicates the function to be performed.

FIX specifies that the virtual storage areas are to reside in central (also called real) storage and are ineligible for page-out while the address space is swapped in. This

parameter does not prevent pages from being paged out when the entire address space is swapped out of central storage. FIX will ignore a request to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by central storage below 16 megabytes. Requests for disabled reference storage are invalid for the FIX parameter.

FREE specifies that the virtual storage areas that were previously fixed via the FIX option are eligible for page-out. A fixed page is not considered pageable until the number of FREE and FIX requests for the page are equal. Requests for disabled reference storage are invalid for the FREE parameter.

LOAD specifies that a page-in operation is to be initiated for the virtual storage area specified, in anticipation of future needs. Requests for disabled reference storage are invalid for the LOAD parameter.

OUT specifies that a page-out operation is to be initiated for the virtual storage area specified. Requests for disabled reference storage are invalid for the OUT parameter.

RELEASE specifies the release of all physical paging resources, including both processor storage and auxiliary storage. Functionally, RELEASE is equivalent to a FREEMAIN macro followed by a GETMAIN macro. That is, the virtual space is maintained, but the data is discarded. When a released page is next referred to, its contents are binary zeros. RELEASE is the only PGSER function that is valid for disabled reference storage.

ANYWHER applies to virtual storage areas that did not specify LOC=(BELOW,ANY) or LOC=(ANY,ANY) or LOC=ANY on a GETMAIN request, that have been previously fixed, and probably will not need to be fixed again. ANYWHER specifies that the virtual storage area specified can be placed either above or below 16 megabytes central on future page-ins.

,LA=*list addr*
specifies the address of the page services list (PSL) for L requests.

,A=*start addr*
specifies the address of the start of the virtual area for R requests.

,EA=*end addr*
specifies the address of the end of the virtual area for R requests.

,TCB=*tcb addr*
specifies either zero or the address of the TCB to be assigned ownership of fixes for a FIX request or fixes for a FREE request. If zero is specified, no TCB is assigned ownership of the request. Cross memory callers must specify zero.

For OUT and LOAD requests, the PGSER routine associates the request with a particular TCB so that the request can be purged if the task terminates before the request is complete. For SVC entry (BRANCH=N), the PGSER routine uses the current TCB.

Note: The TCB resides in storage below 16 megabytes.

,ECB=*ecb addr*
specifies the address of the ECB that is used to signal event completion for an asynchronous FIX or LOAD request. If the caller is in cross memory mode or if the caller requests a synchronous page fix (a FIX for which the caller is suspended until the entire FIX request is complete), the ECB must be zero (ECB=0 or ECB=(r), where (r) represents a register that contains zero).

For a FREE request, ECB specifies the address of the ECB that was used in a previous FIX request. If this parameter is specified, any pages in the previous FIX request that are not yet fixed, will not be fixed. If L is specified, the PSL chain must contain the addresses of the virtual pages in the same order in both the FREE and the previous FIX request. Also, the ECB for the FIX request will not be posted if it was not yet posted at the time of the FREE request.

If the ECB parameter is not specified on a FREE request, only the fix counts for the valid pages in storage at the time of the FREE request are decreased. This will not affect the paging activity and the posting of the ECB associated with the original FIX request.

If an ECB is supplied on a FIX or LOAD request, the caller must check the return code because the ECB will not be posted if the return code is zero. If an ECB is not supplied, it is not necessary to check the return code because control returns to the caller only if the request was successfully completed; if unsuccessful, page services abnormally terminates the caller.

For all callers that supply an ECB, page services verifies that the ECB address is in an area allocated via GETMAIN and if the caller is not in key 0, page services also verifies that the ECB is in the caller's protect key. Before posting the ECB, page services again verifies that the ECB is located in an allocated area and that the ECB is in the caller's protect key. (This is to check that the allocated area has not been freed via FREEMAIN and the protect key has not been changed.) It is the user's responsibility to ensure that the page containing the ECB is not freed and that the key is not altered. If either test fails, page services does not post the ECB.

,RELEASE=Y
,RELEASE=N
specifies that all the central and auxiliary storage associated with the virtual storage areas is to be released to the system (Y) or that all the central and auxiliary storage associated with the virtual storage areas is not to be released to the system (N).

,KEEPREL=Y
,KEEPREL=N
specifies that the virtual pages should be validated again after the page-out completes (Y); or that the virtual pages will be marked invalid and the real frames freed for reuse (N).

,LONG=Y
,LONG=N
specifies that the relative real time anticipated for the FIX is long (Y); or that the relative real time anticipated for the FIX is short (N). (In general, the duration of a fix is long if it can be measured in seconds.)

,BACKOUT=Y
,BACKOUT=N
specifies the procedure to follow when a non-allocated page is encountered during the processing of a FIX request. If BACKOUT=Y, all pages fixed as part of the request are freed before returning to the caller. If BACKOUT=N, the pages previously fixed as part of the request are not freed and no further processing is done before returning to the caller.

,ANYWHER=N
,ANYWHER=Y
specifies that on subsequent page-ins, page services is to assign real frames below 16 megabytes in anticipation of a page-fix (N); or on subsequent page-ins, page services is to assign real frames anywhere (Y). The ANYWHER option takes effect only when the page-fix count goes to zero.

,BRANCH=Y
,BRANCH=N
specifies whether or not this is a branch entry.

If BRANCH=Y is specified, it is a branch entry; and users of this option must provide the address of an 18-word save area in register 13. Register 2 contains the ending address.

If BRANCH=N is specified, it is an SVC entry. Register 15 contains the ending address.

Cross memory callers and callers in AR mode must use BRANCH=Y.

,RELATED=*value*
provides information to document the macro by relating the service performed to some corresponding function or service. The format can be any valid coding value that the user chooses.

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.

On return the register contents are as follows:

Register	Contents
0-4	Used as work registers by the macro.
5-13	Unchanged.
14	Used as work registers by the macro.
15	Return Code.

The return codes, given in register 15, along with the option used and the meaning follow:

Option	Code	Meaning
FIX	0	The operation completed normally and the ECB will not be posted.
FIX	8	The operation is proceeding, the ECB (if available) will be posted with X'00' when the requested pages are fixed.
FREE	0	The operation completed normally.
LOAD	0	The operation completed normally and the ECB will not be posted. If no ECB is supplied, the operation is completed or proceeding.
LOAD	8	The operation is proceeding. The ECB will be posted with X'00' when all page-ins are complete.
OUT	0 C	The operation completed normally. At least one page in the requested range was not paged out.
RELEASE	0	The operation completed normally.
ANYWHER	0	The operation completed normally.

If a error is found in one of the parameters, the requestor is abnormally terminated with a system abend code of X'18A' and one of the following hexadecimal reason codes is provided in register 15:

Hexadecimal Code	Meaning
4	A page-fix operation abnormally terminated cause of an invalid address in a PSL entry. The ECB will not be posted.
4	A page-release operation abnormally terminated because either a page release was attempted for permanently backed storage or a non-system key caller attempted to release storage in a different key.
10	A page-fix, page-free, or a page-load operation abnormally terminated because the PSL or ECB address was invalid.

Callers not authorized to use a specific service are abnormally terminated with a system abend code of X'28A' and a hexadecimal error code of X'10' in register 15. If an environmental error is encountered while processing the page-services request, the caller is abnormally terminated with a system abend code of X'028' and a hexadecimal error code of X'14' in register 15. A unique reason code is also provided in register 0 for these errors.

Example 1

Operation: Synchronously fix the page that starts at the address given in register 1 and ends at the address given in LOADWORD. Use branch entry. No particular TCB is associated with this request.

```
PGSER R,FIX,A=(1),ECB=0,EA=LOADWORD,TCB=0,BRANCH=Y
```

Example 2

Operation: Free the page specified in the PSL pointed to by register 2. The ECB address is given in register 8. Use branch entry. Release all central and auxiliary storage associated with this virtual area. Do not attempt to back the area below 16 megabytes on future page-ins.

```
PGSER L,FREE,LA=(2),ECB=(8),RELEASE=Y,ANYWHER=Y,BRANCH=Y
```

Example 3

Operation: Load the page specified in the PSL pointed to by register 1. Supply an ECB of zero.

```
PGSER L,LOAD,LA=(1),ECB=0
```

Example 4

Operation: Perform a page-out for the virtual area starting at the address given in register 1 and ending at the address given in register 5. The address of the TCB is given in register 8. Use branch entry.

```
PGSER R,OUT,A=(1),EA=(5),TCB=(8),BRANCH=Y
```

Example 5

Operation: Perform a page-out for the virtual area specified in the PSL located at LOADWORD. Use branch entry.

```
PGSER L,OUT,LA=LOADWORD
```

PGSER — Fast Path Page Services

The fast path PGSER macro performs FIX and FREE requests for users on performance paths. The following restrictions apply to this special fast path service:

- Short term fixes only
- No ECB
- No TCB
- No VIO window pages
- Key 0, supervisor state callers only
- Enabled
- Register 13 must point to an 18-word save area in non-pageable storage.
- If the list format of the macro is used, all user-defined short page service lists (SSLs) must be valid in non-pageable storage.
- Users of BRANCH=SPECIAL must include the CVT and IHAPVT mapping macros.

The fast path PGSER macro does not verify any of the restricted conditions. It is the user's responsibility to verify the restricted conditions and to provide recovery to purge FIX requests when the task terminates before a page service request is complete.

The fast path PGSER macro is written as follows:

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

R L	
,FIX ,FREE	
,LA=*list addr*	*list addr:* RX-type address or register (1), (5) - (12). **Note:** This parameter is valid only if L is specified.
,A=*start addr*	*start addr:* RX-type address or register (1), (5) - (12). **Note:** This parameter is valid only if R is specified.
,EA=*ending addr*	*ending addr:* RX-type address or register (2), (5) - (12). **Note:** This parameter is valid only if R is specified.
,BACKOUT=Y ,BACKOUT=N	**Default:** BACKOUT=Y **Note:** This parameter is valid only for FIX requests.
,ASCB=*ascb addr*	*ascb addr:* RX-type address or register (5) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,BRANCH=SPECIAL	

The parameters are explained as follows:

R
L

specify the manner in which the input is supplied. If R is specified, the user supplies the starting and ending addresses of the virtual storage area for which the service is to be performed. Before processing the request, page services puts these addresses in registers 1 and 2, respectively. If L is specified, the user supplies the address of the short page services list (SSL), which specifies the virtual storage area for which the service is to be performed. Before processing the request, page services puts the address of the SSL in register 1. See the topic "Input to Page Services" in *SPL: Application Development Guide* for a description of the SSL.

,FIX
,FREE

indicate the function to be performed.

FIX specifies that the virtual storage areas are to reside in central (also called real) storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of central storage. FIX will ignore a request to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by central storage below 16 megabytes.

FREE specifies that the virtual storage areas that were previously fixed via the FIX option are eligible for page-out. A fixed page is not considered pageable until the number of FREE and FIX requests for the page are equal.

,LA= *list addr*

specifies the address of the short page service list (SSL) for L requests.

,A= *start addr*

specifies the address of the start of the virtual area for R requests.

,EA= *end addr*

specifies the address of the end of the virtual area for R requests.

,BACKOUT=Y
,BACKOUT=N

specify the procedure to follow if an unallocated page is encountered during the processing of a fix request.

If BACKOUT=Y is specified, all pages fixed as part of the request will be freed before returning to the caller.

If BACKOUT=N is specified, the pages previously fixed as part of the request will not be freed before returning to the caller. In this situation, no further pages are processed once an unallocated page is encountered.

,ASCB= *ascb addr*

specifies the address of the ASCB for the currently addressable address space.

Note: The ASCB must reside in 24-bit addressable storage.

,RELATED= *value*

specifies information used to document the macro and to relate the service performed to some corresponding service or function. The format of the information specified can be any valid coding values that the user chooses.

,BRANCH=SPECIAL

specifies a branch entry call to the fast path FIX and FREE services. If BRANCH=SPECIAL is specified, users must provide an 18-word save area in non-pageable storage.

Example 1

Operation: Fix 4096 bytes of storage starting at the address BUFFER. The address of the ASCB is in register 6.

```
PGSER R,FIX,A=BUFFER,EA=BUFFER+4095,BRANCH=SPECIAL,ASCB=(6)
```

Example 2

Operation: Free the area specified in the SSL defined at LISTSSL. Use the ASCB in PSAAOLD.

```
L   5,PSAAOLD
PGSER L,FREE,LA=LISTSSL,ASCB=(5),BRANCH=SPECIAL
```

POST — Signal Event Completion

Use the POST macro to have the specified ECB (event control block) set to indicate the occurrence of an event. If this event satisfies the requirements of an outstanding WAIT or EVENTS macro, the waiting task is taken out of the wait state and dispatched according to its priority. The bits in the ECB are set as follows:

```
Bit 0 of the specified ECB is set to 0 (wait bit).
Bit 1 is set to 1 (complete bit).
Bits 2 through 31 are set to the specified completion code.
```

The description of the POST macro follows. The POST macro is also described in *Application Development Macro Reference*, with the exception of the ASCB, ERRET, ECBKEY, and MEMREL parameters. These parameters are restricted in use to programs that are authorized (supervisor state, APF-authorized, or PSW key 0-7) and, therefore, are only described here. The LINKAGE=BRANCH parameter is further restricted to to supervisor state and key 0 users.

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

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

ecb addr	*ecb addr:* RX-type address, or register (2) - (12), except (10).
,*comp code*	*comp code:* symbol, decimal or hexadecimal digit, or register (0), (2) - (9), (10), or (12). **Range of values:** 0 - 2^{30}-1 **Default:** 0
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr* ,ECBKEY=*key*	*addr:* RX-type address, or register (2) - (9), (12). *err addr:* RX-type address, or register (2) - (9), (12). *key:* symbol, decimal or hexadecimal digit, or register (2) - (9), (12). **Range of values:** 0 - 15 (decimal) **Default:** None. **Note:** If the register form is specified, bits 24-27 of the register must contain the key.
,LINKAGE=SVC ,LINKAGE=BRANCH ,LINKAGE=SYSTEM ,LINKAGE=SYSTEM,ERRET=*err addr*	**Default:** LINKAGE=SVC *err addr:* RX-type address or register (2) -(9), (12)
,BRANCH=NO ,BRANCH=YES	
,MEMREL=YES ,MEMREL=NO	**Default:** MEMREL=YES **Note:** MEMREL can be coded only if LINKAGE=BRANCH and the ASCB and ERRET parameters are coded.
,RELATED=*value*	*value:* any valid macro keyword specification.

The explanation of the parameters is as follows:

ecb addr
specifies the address of the fullword event control block representing the event.

,comp code
specifies the completion code to be placed in the event control block upon completion.

,ASCB = *addr* **,ERRET =** *err addr*
specifies the address of the ASCB of the address space containing the ECB being posted, and the address of the routine to be given control when a POST failure is detected. See the topic "Cross Memory Post" in *SPL: Application Development Guide* for information on the addressing mode in which the exit routine receives control.

Note: The ASCB must reside in 24-bit addressable storage

,ASCB = *addr*,**ERRET =** *err addr*,**ECBKEY =** *key*
specifies the address of the ASCB containing the ECB being posted, the address of the routine to be given control when an error condition resulting from a POST failure is detected, and the storage protection key of the ECB to be posted. If the ECB does not identify a current wait condition against it, the ECB is checked against the key before it is updated with the post completion code. Otherwise, the ECB is checked against the protection key of the waiting task. (See "Cross Memory Post" in *SPL: Application Development Guide* for information about the addressing mode in which the exit routine receives control.)

Note: The ASCB must reside in 24-bit addressable storage

,LINKAGE = SVC
,LINKAGE = BRANCH
,LINKAGE = SYSTEM
,LINKAGE = SYSTEM,ERRET = err addr
,BRANCH = YES
,BRANCH = NO
specifies the type of linkage from the caller to a system service routine that POST invokes. The default is LINKAGE = SVC.

For LINKAGE = SVC, the linkage is through an SVC instruction. This linkage is valid when the caller is in primary ASC mode, where primary, home, and secondary are the same address space. For SVC callers, registers 2-14 are preserved.

For LINKAGE = BRANCH, the linkage is through a branch entry. This linkage is valid when the caller is in primary or secondary ASC mode. The calling requirements and which registers are preserved depend on what other parameters are specified.

- If ASCB = is not specified, the caller must hold the local lock and be in non-cross memory mode. Registers 0-9, 12, and 13 are preserved.
- If ASCB = is specified, the MEMREL parameter and the LOCAL lock determine the calling requirements and registers saved.
  - If the LOCAL lock is held and MEMREL = YES is specified (or defaulted), then the current address space must be the home address space and registers 1-9 are preserved. If the ECBKEY parameter is not specified, register 0 is also preserved.
  - If the LOCAL lock is not held or MEMREL = NO is specified, then only register 9 is preserved. The current address space can be any address space.

For LINKAGE = SYSTEM, the linkage is through a stacking PC instruction. Callers must be enabled, unlocked, and in primary ASC mode. This linkage is valid for callers in cross memory mode. The ECB must be in the caller's primary address space. When you specify LINKAGE = SYSTEM and ASCB, you must also specify ECBKEY.

- ERRET = *err-addr* specifies the address of the routine that gets control when the system detects a POST failure. If the caller is not authorized, the error routine does not receive control. When you have specified LINKAGE = SYSTEM without ASCB =, ERRET is only needed when you are posting an extended SVC and the primary address space is different from the home address space.

- When you issue LINKAGE=SYSTEM, the POST macro service issues the following return codes:

Hexadecimal Code	Meaning
0	Indicates a synchronous POST was done, as requested.
4	Indicates an asynchronous POST was scheduled. If you specified ERRET, the error routine that you specified will receive control.
8	Indicates an asynchronous POST was scheduled. If you specified ERRET, the error routine that you specified will not receive control.

  LINKAGE=SYSTEM without the ASCB parameter is intended to be used by programs in cross memory mode.

,MEMREL=YES
,MEMREL=NO
specifies whether the error routine specified by the ERRET parameter runs in the caller's address space (YES) or in the master scheduler's address space (NO). The default is MEMREL=YES.

If the LOCAL lock is held and MEMREL=YES, the current address space must be the home address space.

If the LOCAL lock is not held or MEMREL=NO, the current address space can be any address space. Thus, when the cross memory mode and the lock status of the caller is unknown, specify MEMREL=NO.

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

Example 1
Operation: Post an event control block whose address is ECB, where the address space containing the ECB has an ASCB specified by register 5, and where ERRRTN is the routine to be given control on error conditions.

```
POST ECB,ASCB=(REG5),ERRET=ERRRTN
```

Example 2
Operation: Post the ECB from example 1 with a hexadecimal completion code of 3FF.

```
POST ECB,X'3FF',ASCB=(REG5),ERRET=ERRRTN
```

Example 3
Operation: Post the ECB from example 1 using a stacking PC for linkage. The address of the error routine is in register 3.

```
POST ECB,LINKAGE=SYSTEM,ECBKEY=0,ASCB=(REG5),ERRET=(REG3)
```

POST (List Form)

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

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

ecb addr	*ecb addr:* A-type address.
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr* ,ECBKEY=YES	*addr:* A-type address. *err addr:* A-type address.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the POST macro.

,ASCB=*addr***,ERRET=***err addr***,ECBKEY=YES**
specifies that the storage protection key of the ECB is defined in the execute form of the POST macro.

Note: The ASCB resides in 24-bit addressable storage.

POST (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede POST.
POST	
b	One or more blanks must follow POST.
ecb addr	*ecb addr:* RX-type address, or register (2) - (12).
,*comp code*	*comp code:* symbol, decimal or hexadecimal digit, or register (0) or (2) - (12). **Range of values:** 0 - 2^{30}-1
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr*, ,ECBKEY=*key*	*addr:* RX-type address, or register (2) - (12). *err addr:* RX-type address, or register (2) - (12). *key:* symbol, decimal or hexadecimal digit, or register (2) - (12). **Range of values:** 0 - 15 (decimal). **Default:** None. **Note:** If the register form is specified, bits 24-27 of the register must contain the key.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*prob addr*)	*prob addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*prob addr*)
specifies the execute form of the POST macro using a remote control program parameter list.

PTRACE — Processor Trace

The PTRACE macro creates a trace table entry and places it in the system trace table. The entry consists of an event identifier, the contents of a designated range of general registers or storage locations, and system supplied status information.

When using this macro, the user must provide the following information:

- The type of trace entry that is to be created
- The data to be recorded in the trace entry

The PTRACE macro can only be issued with DAT-ON. The caller must be in key 0 and supervisor state but can be in cross memory mode and in either 24 or 31-bit addressing mode. All addresses passed to the PTRACE routine are treated as 31-bit addresses. PTRACE users must include the IHAPSA and IHATRVT mapping macros and register 13 must point to a 72-byte save area that can be used by the PTRACE service.

The PTRACE macro is written as follows:

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

TYPE=USR*n*	*n*: hexadecimal digit 0 - F.
,REGS=(reg1,reg2) ,REGS=(1)	**Default**: REGS=(1) reg1: decimal digit 2 - 12. reg2: decimal digit 2 - 12.
,SAVEAREA=STANDARD	

The parameters are explained as follows:

TYPE=USR*n*
TYPE=USR*n* specifies a user-event explicit trace entry. The hexadecimal number, *n*, identifies the entry. Trace processing places this number in the trace entry for identification purposes.

,REGS=(*reg1*,*reg2*)
,REGS=(1)
defines the data to be placed in the user's trace entries. Multiple trace entries are created if more than 5 registers or 5 words of data are requested.

If REGS=(*reg1*,*reg2*) is specified, the data is located in a range of registers, where *reg1* specifies the first register in the range and *reg2* specifies the last register in the range. The register number, *reg2*, must always be greater than or equal to the register number, *reg1*. A maximum of 11 words of data can be indicated for tracing using REGS=(*reg1*,*reg2*).

If REGS=(1) is specified or used as the default, register 1 must contain the 31-bit address of a parameter list. The high order bit of this address must be set to 0. If REGS=(1) is specified, up to 1024 words of data can be recorded. The parameter list contains N+1 fullword entries. The first word contains the number of words of data (N) to be recorded. This is followed by the N words of data to be placed in the user's trace entries.

,SAVEAREA = STANDARD

SAVEAREA = STANDARD specifies that register 13 contains the address of a 72-byte save area that can be used by the PTRACE routine.

When control is returned, registers 2-13 are restored to their original values, but the original contents of registers 0, 1, 14, and 15 are destroyed. On exit, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The function completed successfully.

Example 1

Operation: Create a trace table entry for user event 4. Registers 5, 6, and 7 contain the user data to be recorded.

```
PTRACE TYPE=USR4,REGS=(5,7),SAVEAREA=STANDARD
```

Example 2

Operation: Create trace table entries for user event C. Register 1 contains the address of a parameter list containing the data to be recorded.

```
PTRACE TYPE=USRC,REGS=(1),SAVEAREA=STANDARD
```

PURGEDQ — Purge SRB Activity

The PURGEDQ macro allows a task to purge particular SRB activity. Because an SRB routine is dispatched asynchronously to the actual issuance of a SCHEDULE macro, the conditions that existed in the system at the time the SCHEDULE was issued may have totally changed by the time the routine is dispatched. If, in this time interval, the environment that the asynchronous routine requires to run successfully has been changed, the results are unpredictable. For this reason, the PURGEDQ macro is available to:

- Dequeue SRBs not yet dispatched.
- Allow processing for previously scheduled SRBs to complete.
- "Clean up" each dequeued SRB.

The following requirements apply to the caller:

- Must be in task mode
- Must be in primary ASC mode
- No locks may be held.

The parameters on PURGEDQ determine the target address space and limit the scope of the purge. When purging SRBs scheduled in the current address space, PURGEDQ waits for dispatched SRBs to terminate. PURGEDQ does not purge or wait for terminations of SRBs scheduled into address spaces other than the current address space once they have been dispatched. The issuer of PURGEDQ is not informed of SRBs that cannot be purged.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

See *SPL: Application Development Guide* for more information on using the PURGEDQ macro, especially the ASIDTCB parameter.

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

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

RMTR=*RMTR addr*	*RMTR addr:* RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12).
,ASIDTCB=*TCB addr*	*TCB addr:* RX-type address, or register (2) - (12).

The parameters are explained as follows:

RMTR=*RMTR addr*
Limits the purge to SRBs that contain the same address in field SRBRMTR.

,ASID=*ASID addr*
specifies the address of a half word that contains the ASID of the target address space into which the SRB was scheduled. If omitted, the system assumes the current address space.

,ASIDTCB=*TCB addr*
Provides a method of limiting the scope of the purge. ASIDTCB specifies the address of a double word that optionally defines a TCB address to match against the field SRBPTCB, or an ASID to match against the field SRBPASID, or both a TCB address and an ASID. When a non-zero value is specified, the system purges only SRBs that match

the specified value. When the TCB address is non-zero, the ASID field must also be non-zero. If the ASIDTCB parameter is omitted, SRBPASID and SRBPTCB are matched against the current address space ID and current TCB address.

The format of the double word is:

bytes 0-1 Reserved
bytes 2-3 ASID for match with SRBPASID or zero.
bytes 4-7 TCB address for match with SRBPTCB or zero.
If TCB address is specified, ASID must also be specified.

Note: The TCB resides in storage below 16 megabytes.

Example 1

Operation: Purge all SRBs scheduled to ASID '20'X with:

- SRBPTCB equal to the current TCB (that is, the TCB issuing the PURGEDQ)
- SRBPASID equal to the current ASID
- SRBRMTR equal to the address of RMTR routine RMTRA.

```
         PURGEDQ ASID=AS1,RMTR=RMTRA

AS1      DC   XL2'0020'
```

Example 2

Operation: Purge all SRBs scheduled to ASID '21'X, regardless of what is specified in SRBPASID and SRBPTCB, and that have SRBRMTR equal to the address of RMTR routine RMTRB.

```
         PURGEDQ ASID=AS2,ASIDTCB=PURGPRM1,RMTR=RMTRB

PURGPRM1 DC    XL4'00000000'
AS2      DC    XL2'0021'
```

Example 3

Operation: Purge all SRBs scheduled to the current address space (that is, the address space from which this PURGEDQ was issued) that have:

- SRBPASID of '12'X
- SRBPTCB equal to the the address of TCBX
- SRBRMTR equal to the address of RMTR routine RMTRC

```
         PURGEDQ ASIDTCB=PURGPRM2,RMTR=RMTRC

PURGPRM2 DS    0CL8
         DC    XL2'0000'
PURGASID DC    XL2'0012'
PURGTCB  DC    A(TCBX)
```

Example 4

Operation: Purge all SRBs scheduled into the current address space, related to the current (terminating) task, and associated with the resource manager termination routine located at RESCLEAN.

```
         PURGEDQ RMTR=RESCLEAN
```

PURGEDQ (List Form)

The list form of the PURGEDQ macro is used to construct a remote program parameter list.

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

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

RMTR=*RMTR addr*	*RMTR addr:* A-type address.
,ASID=*ASID addr*	*ASID addr:* A-type address.
,ASIDTCB=*TCB addr*	*TCB addr:* A-type address.
,MF=L	

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

,MF=L
specifies the list form of the PURGEDQ macro.

Example

Operation: Specify the resource manager termination routine located at RESCLEAN and produce the parameter list to be used by the execute form of the PURGEDQ macro.

```
STATPDQ PURGEDQ RMTR=RESCLEAN,MF=L
```

PURGEDQ (Execute Form)

The execute form of the PURGEDQ macro uses a remote control program parameter list. The parameter list is constructed using the list form of PURGEDQ.

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

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

RMTR=*RMTR addr*	*RMTR addr:* RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12).
,ASIDTCB=*TCB addr*	*TCB addr:* RX-type address, or register (2) - (12).
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl addr)*
specifies the execute form of the PURGEDQ macro, using a remote control program parameter list.

Example

Operation: Purge all SRBs scheduled into the address space given in register 6 and associated with the resource manager termination routine located at RESCLEAN. Indicate that the remote control program parameter list is located at STATPDQ.

```
PURGEDQ ASID=(6),RMTR=RESCLEAN,MF=(E,STATPDQ)
```

QEDIT — Command Input Buffer Manipulation

The QEDIT macro generates the required entry parameters and processes the command input buffer for the following uses:

- Dechaining and freeing of a command input buffer (CIB) from the CIB chain for a task.
- Setting a limit for the number of CIBs that may be simultaneously chained for a task.

The QEDIT macro is written as follows:

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

ORIGIN=*CIB addr ptr*	*CIB addr ptr:* RX-type address, or register (0),(2) - (12).
,BLOCK=*CIB addr* ,CIBCTR=*CIB nmbr*	*CIB addr:* RX-type address, or register (1), (2) - (12). *CIB nmbr:* decimal digit, with a maximum value of 255 or register (1), (2) - (12).

The parameters are explained as follows:

ORIGIN = *CIB addr ptr*
specifies the address of the pointer to the first CIB chain for the task. This address is obtained using the EXTRACT macro. If BLOCK and CIBCTR are omitted, the caller must be executing under PSW key 0-7; in this case, the entire CIB chain is freed. The system prevents problem state programs from freeing the entire CIB chain.

,BLOCK = *CIB addr*
specifies the address of the CIB to be freed from the CIB chain for a task.

,CIBCTR = *CIB nmbr*
specifies the limit for the number of CIBs to be chained at any time for a task.

Notes:

1. When using any address returned from the EXTRACT macro as input to the QEDIT macro, the user must use the IEZCOM mapping macro to establish addressability based on the address returned by EXTRACT.
2. The CIB must reside in 24-bit addressable storage.

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

Hexadecimal Code	Meaning
00	The required function was successfully completed.
04	The CIB to be deleted was not found on any CIB chain.
08	The limit for the number of CIBs to be chained was exceeded; an issuer who made a request to free all the CIBs on a chain was not in supervisor state and PSW key zero; or the user provided an invalid address for the pointer to the CIB chain, an invalid address for the CIB address, or an invalid CIB number as input to the macro.

Example 1

Operation: Free the entire CIB chain, where register 8 contains the address of the pointer to the CIB chain.

```
QEDIT ORGIN=(8)
```

Example 2

Operation: Free the CIB whose address is in register 5 from the CIB chain. Register 8 contains the address of the pointer to the CIB chain.

```
QEDIT ORIGIN=(8),BLOCK=(5)
```

RACDEF — Define a Resource to RACF (for RACF Release 1.8.1 or earlier)

Note: The RACROUTE macro is the preferred programming interface.

This macro description applies to RACF Release 1.8.1 or earlier. Your program can invoke the RACDEF macro directly; however, IBM recommends that you invoke the equivalent function through the RACROUTE macro, using the REQUEST = DEFINE parameter. See "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 437 for the applicable RACROUTE macro description.

If you have RACF Release 1.9 installed on your system, you can still invoke the RACDEF macro directly. However, if you are going to use the new Release 1.9 functions, see the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST = DEFINE:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 447**
- **"RACROUTE REQUEST = DEFINE — Define a Resource to RACF (for RACF Release 1.9)" on page 485.**

The RACDEF macro defines, modifies, or deletes resource profiles for RACF. You can also use it for special cases of authorization checking. RACF uses the resulting profiles to perform RACHECK authorization checking. The RACDEF caller must be authorized (APF-authorized, in system key 0-7, or in supervisor state).

A RACF user can change or add the RACDEF parameters, OWNER, LEVEL, UACC, or AUDIT by means of the RACDEF preprocessing and postprocessing exit routines.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode who want to use the RACDEF function can code the RACROUTE macro.

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

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

ENTITY=*profile name addr*	*profile name addr:* A-type address, or register (2) - (12).
,VOLSER=*vol addr*	*vol addr:* A-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used.
,TYPE=DEFINE ,TYPE=DEFINE,NEWNAME= *new dsn addr* ,TYPE=ADDVOL,OLDVOL= *old vol addr* ,TYPE=CHGVOL,OLDVOL= *old vol addr* ,TYPE=DELETE	*new dsn addr:* A-type address, or register (2) - (12). *old vol addr:* A-type address, or register (2) - (12). **Default:** TYPE=DEFINE
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** CLASS='DATASET'
,MENTITY=*entity addr*	*entity addr:* A-type address, or register (2) - (12).
,MCLASS='*classname*' ,MCLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** MCLASS='DATASET'
,MVOLSER=*volser addr*	*volser addr:* A-type address, or register (2) - (12). **Default:** MCLASS='DATASET'
,MGENER=ASIS ,MGENER=YES	**Default:** MGENER=ASIS
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12).
,UNIT=*unit addr*	*unit addr:* A-type address, or register (2) - (12).
,OWNER=*owner id addr*	*owner id addr:* A-type address, or register (2) - (12).
,LEVEL=*number* ,LEVEL=*reg*	**Default:** zero. *reg:* register (2) - (12).
,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=EXECUTE	

,UACC=NONE ,UACC=*reg*	*reg:* register (2) - (12).
,DATA=*data addr*	*data addr:* A-type address or register (2) - (12).
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value (access level),audit value(access level),. . .)* ,AUDIT=*(reg)*	**Note:** AUDIT is valid only if TYPE=DEFINE is specified. *audit value:* ALL, SUCCESS, or FAILURES *access level:* READ, UPDATE, CONTROL, or ALTER **Default:** READ *reg:* register (2) - (12).
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default:** CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default:** WARNING=NO **Note:** WARNING is valid only if TYPE=DEFINE is specified.
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999
,EXPDT=*expir date addr*	*expir date addr:* A-type address or register (2) - (12).
,EXPDTX=*extended expir date addr*	*extended expir date addr:* A-type address or register (2) - (12).
,RETPD=*retn period addr*	*retn period addr:* A-type address or register (2) - (12). **Default:** see description of parameter.
,ACCLVL=*(access value addr)* ,ACCLVL=(*access value addr, parm list addr)*	*access value addr:* A-type address or register (2) - (12). *parm list addr:* A-type address, or register (2) - (12).
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,SECLVL=*addr*	*addr:* A-type address, or register (2) - (12).
,ERASE=YES ,ERASE=NO	**Default:** ERASE=NO
,NOTIFY=*notify id addr*	*notify id addr:* A-type address or register (2) - (12).
,ENVIR=VERIFY	specifies that only verification is to be done. **Default:** Normal RACDEF processing.
,RESOWN=*resource owner addr*	*resource owner addr:* A-type address, or register (2) - (12).
,STORCLA=*storage class addr*	*storage class addr* A-type address, or register (2) - (12).
,MGMTCLA=*management class addr*	*management class addr:* A-type address, or register (2) - (12).

The parameters are explained as follows:

ENTITY =*profile name addr*
specifies the address of the name of the discrete or generic profile that is to be defined to, modified, or deleted from RACF. The profile name is a 44-byte DASD data set name for CLASS='DATASET' or a 6-byte volume serial name for CLASS='DASDVOL' or CLASS='TAPEVOL'. The lengths of all other profile names are determined by the class descriptor table. The name must be left justified in the field and padded with blanks.

,VOLSER =*vol addr*
specifies the address of the volume serial number.

- For TYPE=ADDVOL, it specifies the address of the new volume to be added to the definition of the data set.
- For TYPE=ADDVOL and CLASS='TAPEVOL', it specifies the address of the new volume being added to the tape volume set identified by ENTITY.
- For TYPE=DEFINE and CLASS='DATASET', it specifies the address of the catalog (for a VSAM data set), or of the volume on which the data set resides (for a non-VSAM data set).

The volume serial number is optional if you specify DSTYPE=M; it is ignored if the profile name is generic.

The field pointed to by the specified address contains the volume serial number (padded to the right with blanks, if necessary, to make six characters).

,TYPE =**DEFINE**
,TYPE =**DEFINE ,NEWNAME** =*new dsn addr*
,TYPE =**ADDVOL ,OLDVOL** =*old vol addr*
,TYPE =**CHGVOL ,OLDVOL** =*old vol addr*
,TYPE =**DELETE**
specifies the type of action to be taken.

- TYPE=DEFINE - The definition of the resource is added to the RACF data set, and the current user is established as the owner of the defined entity.
- TYPE=DEFINE,NEWNAME= - If you specify NEWNAME, the address points to a 44-byte field containing the new name for the data set that is to be renamed. NEWNAME is only valid with CLASS= 'DATASET'. NEWNAME is not valid with DSTYPE=T.
- TYPE=ADDVOL - The new volume is added to the definition of the specified resource. For the DATASET class, the OLDVOL address specifies a previous volume of a multi-volume data set. For the TAPEVOL class, the ENTITY address specifies a previous volume of a tape volume set. This parameter applies only to discrete profiles.
- TYPE=CHGVOL - The volume serial number in the definition of the specified resource is changed from the old volume serial number identified in OLDVOL to the new volume serial number identified in the VOLSER parameter. This parameter applies only to discrete profiles. TYPE=CHGVOL is not valid with DSTYPE=T.
- TYPE=DELETE - The definition of the resource is removed from the RACF data set. (For a multivolume data set or a tape volume set, only the specified volume is removed from the definition.)

,DSTYPE =**N**
,DSTYPE =**V**
,DSTYPE =**M**
,DSTYPE =**T**
specifies the type of data set associated with the request:

- N for non-VSAM
- V for VSAM
- M for model profile
- T for tape

If you specify DSTYPE=T and tape data set protection is not active, the processing will be the same as for RACDEF CLASS='TAPEVOL'. Specify DSTYPE only for CLASS='DATASET'.

,INSTLN =*parm list addr*
specifies the address of an area that is to contain parameter information meaningful to the RACDEF installation exit routines. This information is passed to the installation exit routines when they are given control from the RACDEF routine.

The INSTLN parameter can be used by an application program acting as a resource manager that needs to pass information to the RACDEF installation exit routines.

,CLASS ='*classname*'
,CLASS =*class name addr*
specifies that a profile is to be defined, modified, or deleted in the specified class. If you specify an address, the address must point to a one-byte length field followed by the class name (for example, DATASET or TAPEVOL). The class name should be no longer than eight characters.

,MENTITY =*entity addr*
specifies the address of the name of the discrete or generic profile that is to be used as a model in defining the ENTITY profile. The profile can belong to any class, as specified by the MCLASS parameter, and can be either a discrete or a generic profile. You can specify MENTITY with TYPE=DEFINE but not with TYPE=DEFINE,NEWNAME=*new dsn addr*. The name is contained in a 44-byte field pointed to by the specified address. The name is left justified in the field and padded with blanks.

,MCLASS= '*classname*'
,MCLASS= *class name addr*
specifies the class to which the profile defined by MENTITY= belongs. If you specify an address, the address must point to a one-byte length field followed by the class name. The class name should be no longer than eight characters. The default is MCLASS='DATASET'.

,MVOLSER =*volser addr*
specifies the address of the volume serial number of the volume associated with the profile in the MENTITY operand. The field to which the specified address points contains the volume serial number, padded to the right with blanks, if necessary, to make six characters.

If you specify MENTITY and CLASS='DATASET', you must specify MVOLSER with the name of the VOLSER or with blanks.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

If you specify it with blanks, the discrete MENTITY data set profile name must be unique, meaning it has no duplicates on the data base. In this case, RACF determines the correct MVOLSER.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,MGENER =ASIS
,MGENER =YES
specifies whether the profile name defined by MENTITY is to be treated as a generic name.

- If you specify MGENER=ASIS, the profile name is considered a generic only if it contains a generic character: an asterisk (*) or a percent sign (%).
- If MGENER=YES is specified, the profile name is considered a generic, even if it does not contain a generic character: an asterisk (*) or a percent sign (%).

MGENER is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*).

,ACEE = *acee addr*
specifies the address of the accessor environment element (ACEE) to be used during RACDEF processing. If you do not specify ACEE, RACF uses the TASK ACEE pointer (TCBSENV) in the extended TCB. If the TASK ACEE pointer is zero, RACF uses the main ACEE. The main ACEE's address is in the ASXBSENV field in the address space extension block.

,UNIT = *unit addr*
specifies the address of a field containing unit information. UNIT is valid only if you specify TYPE = CHGVOL or TYPE = DEFINE. If you specify a unit address, the unit information in the data set profile is replaced by the unit information pointed to by this unit address. The unit address must point to a field containing a one-byte length field (whose value can range from 4 through 8) followed by the actual unit information. If the value in the length field is 4, the unit information is assumed to contain a copy of the information in the UCBTYP field of the UCB. Otherwise the unit information is assumed to be in the generic form (for example, 3330-1). This parameter is ignored for generic names.

,OWNER = *owner id addr*
specifies the address of a field containing the profile owner's id. OWNER is valid if you specify TYPE = DEFINE. The owner's id must be a valid (RACF-defined) userid or group name. The address must point to an 8-byte field containing the owner's name, left-justified and padded with blanks.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Note: RACF does not check the validity of the owner's id if it has been added or modified by either the RACDEF preprocessing or postprocessing exit routines, or both.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,LEVEL = *number*
,LEVEL = *reg*
specifies a level value for the profile. LEVEL is valid only if you specify TYPE = DEFINE. The level number must be a valid decimal number in the range 0 to 99. If you specify a register, its low-order byte must contain the binary representation of the number.

Note: RACF does not check the validity of this number if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routines.

,UACC = ALTER
,UACC = CONTROL
,UACC = UPDATE
,UACC = READ
,UACC = EXECUTE
,UACC = NONE
,UACC = *reg*
specifies a universal access authority for the profile. UACC is valid only if you specify TYPE = DEFINE. UACC must contain a valid access authority (EXECUTE, ALTER, CONTROL, UPDATE, READ, or NONE).

To use the EXECUTE keyword, your system must have DFP Version 3 installed. EXECUTE authority means that the user has *only* the ability to execute the program; the user cannot READ the program. If your system is *not* running with DFP Version 3, an EXECUTE access attempt will be treated as NONE.

If you specify a register, the low-order byte must contain one of the following valid access authorities:

X'80' - ALTER
X'40' - CONTROL
X'20' - UPDATE
X'10' - READ
X'01' - NONE
X'08' - EXECUTE

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Note: RACF does not check the validity of the universal access authority if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routine.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,DATA =*data addr*
specifies the address of a field that contains up to 255 characters of installation-defined data to be placed in the profile. DATA is valid only if you specify TYPE=DEFINE. The data address must point to a field containing a one-byte length field (whose value can range from 0 to 255) followed by the actual installation-defined data.

,AUDIT =**NONE**
,AUDIT =*audit value*
,AUDIT =*(audit value(access level),audit value(access level),. . .)*
,AUDIT =*(reg)*
specifies the types of accesses and the access levels that are to be logged to the SMF data set. AUDIT is valid only if you specify TYPE=DEFINE.

For *audit value*, specify one of the following: ALL, SUCCESS, or FAILURES. You may optionally specify an *access level*(access authority) following each *audit value*.

Access Levels:

- EXECUTE is not audited. If you specify FAILURES (READ), EXECUTE logs the READ attempt as a failure, but allows EXECUTE access to the data set.
- READ the default access level value, logs access attempts at any level.
- UPDATE logs access attempts at the UPDATE, CONTROL, and ALTER levels.
- CONTROL logs access attempts at the CONTROL and ALTER levels.
- ALTER logs access attempts at the ALTER level only.

Note: For more information about specific audit values and access levels, please see the *RACF Command Language Reference*.

RACF resolves combinations of conflicting specifications by using the most encompassing specification. Thus, in the case of the following:

```
ALL(UPDATE),FAILURES(READ)
```

RACF assumes SUCCESS(UPDATE),FAILURES(READ).

For compatibility with previous releases, you can also specify register notation as AUDIT=*reg* if the register is not given as a symbolic name ALL, SUCCESS, or FAILURES.

Logging is controlled separately for SUCCESS and FAILURES, and can also be suppressed or requested via the RACHECK post-processing installation exit routine.

If you specify a register, its low-order byte must contain one of the following valid audit values:

Bit	Meaning
0	ALL
1	SUCCESS
2	FAILURES
3	NONE
4-5	Qualifier for SUCCESS
6-7	Qualifier for FAILURES

The qualifier codes are as follows:

00	READ
01	UPDATE
10	CONTROL
11	ALTER

Only one of bits 0-3 can be on. If you specify ALL, you can use the two qualifier fields to request different logging levels for successful and unsuccessful events.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Note: RACF does not check the validity of the audit type if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routine.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,RACFIND =YES
,RACFIND =NO
specifies whether or not a discrete profile is involved in RACDEF processing. When you specify TYPE=DEFINE, RACFIND=YES means that a discrete profile is to be created. When you specify TYPE=DELETE, DEFINE with NEWNAME, CHGVOL, or ADDVOL, RACFIND=YES means that a discrete profile already exists.

RACFIND=NO means (when TYPE=DEFINE) that no discrete profile is to be created, but some authorization checking is required. For other types of action, no discrete profile should exist.

Note: Use the RACFIND keyword only with the DATASET class.

,CHKAUTH =YES
,CHKAUTH =NO
specifies whether RACF will verify that the user is authorized to perform the operation.

CHKAUTH=YES is valid when you specify either TYPE=DEFINE,NEWNAME= or TYPE=DELETE.

For DSTYPE=T, specifies that RACF will verify that the user is authorized to define a data set (TYPE=DEFINE), delete a data set (TYPE=DELETE), or add a volume (TYPE=ADDVOL).

,RELEASE =1.6|1.7|1.8|1.8.1
specifies the RACF release level of the parameter list that this macro will generate.

To use the parameters associated with a release, specify the release number of that release or a later release. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 13.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACDEF macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,FILESEQ= *number*
,FILESEQ= *reg*

specifies the file sequence number of a tape data set on a tape volume or within a tape volume set. The *number* must be in the range 1 - 9999. If you specify a register, it must contain the file sequence number in the low-order half-word. If you do not specify CLASS='DATASET' and DSTYPE=T, FILESEQ is ignored.

,EXPDT= *expir date addr*
,EXPDTX= *extended expir date addr*
,RETPD= *retn period addr*

specifies the address containing information about the data set's expiration date or RACF security retention period.

EXPDT=*expir date addr* specifies the address of a three-byte field containing the data set's expiration date. The date is given in packed decimal form as YYDDDF, where YY is the year and DDD is the day number. The year must be in the range 01 through 99, and the day number must be in the range 1 through 366. All fields are right justified.

EXPDTX=*extended expir date addr* specifies the address of a 4-byte field that contains the address of the data set's expiration date. The date is given in packed decimal form as CCYYDDDF, where CC is the century change greater than 19, YY is the year, and DDD is the day number. The year must be in the range 01 through 99. The day must be in the range 1 through 366. All fields are right justified. When you want to represent 19 for the century, specify CC as 00; when you want to represent 20 for the century, specify CC as 01. To use this parameter, you must also specify RELEASE=1.8.

RETPD=*retn period addr* specifies the address of a two-byte binary field containing the number of days after which RACF protection for the data set expires. The value you specify must be in the range 1 through 65533. To indicate that there is no expiration date, specify 65534.

If you do not specify any of these parameters, a default RACF security retention period is obtained from the RETPD keyword specified on a prior RACF SETROPTS command.

These parameters are valid only if CLASS='DATASET' and DSTYPE=T.

,ACCLVL= *(access value addr)*
,ACCLVL= **(***access value addr,parm list addr***)**

specifies the tape label access level information for the MVS tape label functions. The address must point to a field containing a one-byte length field (with a value that can range from 0-8) followed by an eight-character string that will be passed to the RACDEF installation exit routines. The parameter list address points to a parameter list containing additional information to be passed to the RACDEF installation exit routines.

RACF does not check or modify this information.

TAPELBL =STD|BLP|NL

specifies the type of tape labelling to be done:

- STD - IBM or ANSI standard labels.
- BLP - bypass label processing.
- NL - non-labeled tapes.

For TAPELBL=BLP, the user must have the requested authority to the profile ICHBLP in the general resource class FACILITY. For TAPELBL=NL or BLP, the user will not be allowed to protect volumes with volume serial numbers in the format "Lnnnnn."

The TAPELBL parameter is passed to the RACDEF installation exits.

This parameter is primarily intended for use by data management routines to indicate the label type from the LABEL keyword on the JCL statement.

This parameter is only valid for CLASS='DATASET' and DSTYPE=T, or CLASS='TAPEVOL'.

,SECLVL = *addr*
specifies the address of a list of installation-defined security level identifiers. Each identifier is a half word, containing a value that corresponds to an installation-defined security level name.

The identifiers must be in the range 1 - 254. Only one identifier may be passed in the list.

The list must start with a full word containing the number of entries in the list (currently, only 0 or 1).

,ERASE = YES
,ERASE = NO
specifies whether the DASD data set, or the released space, is to be erased when it is deleted or part of its space is to be released for reuse.

- If you specify ERASE=YES, the data set will be erased when it is deleted, or released for reuse.
- If you specify ERASE=NO, the data set will not be erased, deleted, or released.

Note: This parameter may be overridden by the RACF SETROPTS command.

The default is ERASE=NO.

,NOTIFY = *notify id addr*
specifies the address of an eight-byte area containing the userid of the RACF-defined user who is to be notified when an unauthorized attempt to access the resource protected by this profile is detected.

,GENERIC =YES
,GENERIC =ASIS
specifies whether the resource name is treated as a generic profile name. If you specify GENERIC with CLASS=DEFINE, NEWNAME, then GENERIC applies to both the old and new names. GENERIC is ignored if you do not specify the GENCMD option on the RACF SETROPTS command for the class (see *RACF Command Language Reference*).

This keyword is designed primarily for use by RACF commands.

- If you specify GENERIC=YES, the resource name is considered a generic profile name, even if it does not contain a generic character: an asterisk (*) or a percent sign (%).
- If you specify GENERIC=ASIS, the resource name is considered a generic only if it contains a generic character: an asterisk (*) or a percent sign (%).

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,WARNING =YES
,WARNING =NO
WARNING is valid only if you specify TYPE=DEFINE. If you specify WARNING=YES, access is granted to the resource and the event is logged as a warning if either the SUCCESS and/or FAILURES logging is requested.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,ENVIR =VERIFY
specifies that only verification is to be done. If you specify ENVIR=VERIFY, you must also specify TYPE=DEFINE, RESOWN, the current RELEASE level, and either MGMTCLA or STORCLA, or both .

When you specify ENVIR=VERIFY, RACDEF calls RACHECK to verify that the user specified on the RESOWN keyword has the authority to create a data set in the specified resource class. To verify the authority of a non-RACF defined user to the specified resource, specify *NONE* for the RESOWNER keyword. In DFP support, the two resource classes are the MGMTCLA and the STORCLA.

Note: If you do not specify ENVIR=VERIFY, normal RACDEF processing occurs.

,RESOWN =*resource owner addr*
specifies the address of a field containing the resource owner's id. If you specify RESOWN, you must also specify TYPE=DEFINE and the current RELEASE parameter. The resource owner's id must be either a valid (RACF-defined) userid or group name, or *NONE*. If the resource owner's id is specified as *NONE*, then RACF performs third-party RACHECK using USERID=*NONE*. The address must point to a 2-byte field followed by the resource owner's name.

,STORCLA =*storage class addr*
specifies the address of the storage class to which the resource owner must have authority. The address must point to a 2-byte field followed by the management class name. If you specify STORCLA, you must also specify TYPE=DEFINE, RESOWN, the current RELEASE parameter and ENVIR=VERIFY.

,MGMTCLA =*management class addr*
specifies the address of of a management class to which the resource owner must have authority. The address must point to an 8-byte field that contains a management class name preceded by a halfword length. If you specify MGMTCLA, you must also specify TYPE=DEFINE, RESOWN, the current RELEASE parameter, and ENVIR=VERIFY.

Parameters for RELEASE = 1.6 and Later

The RELEASE values for which a parameter is valid are marked with an 'X'.

Figure 13. RACDEF Parameters for RELEASE = 1.6 and Later

Parameter	RELEASE = 1.6	RELEASE = 1.7	RELEASE = 1.8 or 1.8.1
ACEE =	X	X	X
ACCLVL =		X	X
AUDIT =	X	X	X
CHKAUTH =	X	X	X
CLASS =	X	X	X
DATA =	X	X	X
DSTYPE = N, V, or M	X	X	X
DSTYPE = T		X	X
ENTITY =	X	X	X
ENVIR =			X
ERASE =		X	X
EXPDT =		X	X
EXPDTX =			X
FILESEQ =		X	X
GENERIC =	X	X	X
INSTLN =	X	X	X
LEVEL =	X	X	X
MCLASS =		X	X
MENTITY =	X	X	X
MGENER =		X	X
MGMTCLA =			X
MVOLSER =	X	X	X
NOTIFY =		X	X
OWNER =	X	X	X
RACFIND =	X	X	X
RELEASE =	X	X	X
RESOWN =			X
RETPD =		X	X
SECLVL =		X	X
STORCLA =			X
TAPELBL =		X	X
TYPE =	X	X	X
UACC =	X	X	X
UNIT =	X	X	X
VOLSER =	X	X	X
WARNING =	X	X	X

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	RACDEF has completed successfully. Register 0 contains one of the following reason codes: 00 indicates a normal completion. 04 indicates RACFIND = NO was specified and no generic profile applying to the data set was found.
04	RACDEF has completed processing. Register 0 contains one of the following reason codes: 00 indicates the following: - For TYPE = DEFINE, the resource name was previously defined. - For TYPE = DEFINE,NEWNAME, the new resource name was previously defined. - For TYPE = DELETE, the resource name was not previously defined. 04 indicates for TYPE = DEFINE that the data set name was previously defined on a different volume and that the option disallowing duplicate data sets was specified at IPL.
08	RACDEF has completed processing. Register 0 contains one of the following reason codes: 00 indicates the following: - For TYPE = DEFINE, the check for authority to allocate a data set or create a profile with the specified name has been failed. - For TYPE = DELETE or TYPE = DEFINE,NEWNAME if CHKAUTH = YES is specified, the authorization check has been failed. - For TYPE = ADDVOL,OLDVOL the old value was not defined. 04 indicates for TYPE = DEFINE that no profile was found to protect the data set and that the RACF protect-all option is in effect. 08 indicates TYPE = DEFINE or TYPE = ADDVOL,OLDVOL and DSTYPE = T were specified. And the user is not authorized to define a data set on the specified volume. 0C indicates TYPE = DEFINE and DSTYPE = T were specified. And the user is not authorized to define a data set with the specified name. 10 indicates DSTYPE = T or CLASS = TAPEVOL was specified. and the user is not authorized to specify LABEL = (,BLP). 20 indicates the data set owner is not authorized to use the specified DFP storage class. 24 indicates the data set owner is not authorized to use the specified DFP management class.
0C	For TYPE = DEFINE,NEWNAME, the old data set name was not defined; or if the generation data group (GDG) modeling function is active, an attempt was made to rename a GDG name to a name that requires the creation of a new profile; or if generic profile checking is active, the old data set name was protected by a generic profile and there is no generic profile that will protect the new data set name. This last case refers only to an attempt to rename an existing profile, which cannot be found.
10	For TYPE = DEFINE with MENTITY, the model resource was not defined.

64 Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACDEF macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example 1

Operation: Invoke RACF to define a discrete profile for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name. All successful requests for update authority to the data set are to be audited, as well as all unsuccessful ones.

```
RACDEF  ENTITY=(R7),VOLSER=(R8),CLASS='DATASET',
        AUDIT=(SUCCESS(UPDATE),FAILURES),
        RACFIND=YES
```

Example 2

Operation: Use the standard form of the RACDEF macro to define a discrete data set profile for a non-VSAM DASD data set. The data set for which you are creating a profile is a non-VSAM DASD data set named DSNAME. It resides on a volume id named VOLID. You want to create a discrete profile by specifying the RACFIND keyword. In addition, you want to notify the user called USERNAME of any access attempts that have been rejected because they exceed the UACC of READ that you are allowing.

```
RACDEF ENTITY=DSNAME,VOLSER=VOLID,CLASS='DATASET',UACC=READ,      X

      RACFIND=YES,NOTIFY=USERNAME,RELEASE=1.7
```

Example 3

Operation: Use the standard form of the macro to check the authority of a user to define a discrete data set profile for a non-VSAM dasd data set, but do not actually build the profile. The name of the data set is DSNAME.

```
RACDEF ENTITY=DSNAME,VOLSER=VOLID,CLASS='DATASET',RACFIND=NO
```

Example 4

Operation: Use the standard form of the macro to define a generic data set profile named PROFNAME. Use the discrete profile named MDELPROF whose volser is in MDELVOL as a model for the new profile. Notify the user named USERNAME of any access attempts that have been rejected because they exceed the UACC of READ which you are allowing.

```
RACDEF ENTITY=PROFNAME,CLASS='DATASET',GENERIC=YES,MENTITY=MDELPROF,  X
      MVOLSER=MDELVOL,UACC=READ,NOTIFY=USERNAME,RELEASE=1.7
```

Example 5

Operation: Use the standard form of the macro to define a tape volume profile for a volume whose id is VOLID. Allow a universal access level of READ.

```
RACDEF ENTITY=VOLID,CLASS='TAPEVOL',UACC=READ
```

Example 6

Operation: Use the standard form of the macro to delete a discrete data set profile named DSNAME located on the volume named VOLID.

```
RACDEF TYPE=DELETE,ENTITY=DSNAME,VOLSER=VOLID,CLASS='DATASET'
```

RACDEF (List Form)

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

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

ENTITY=*profile name addr*	*profile name addr:* A-type address. **Note:** ENTITY must be specified on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* A-type address **Note:** VOLSER is required (on either LIST or EXECUTE) only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used.
,TYPE=DEFINE	
,TYPE=DEFINE,NEWNAME=*new dsn addr*	*new dsn addr:* A-type address
,TYPE=ADDVOL,OLDVOL=*old vol addr*	*old vol addr:* A-type address
,TYPE=CHGVOL,OLDVOL=*old vol addr*	
,TYPE=DELETE	**Default:** TYPE=DEFINE
,DSTYPE=N	**Default:** DSTYPE=N
,DSTYPE=V	
,DSTYPE=M	
,DSTYPE=T	
,INSTLN=*parm list addr*	*parm list addr:* A-type address
,CLASS='*classname*'	*'classname':* 1-8 character name.
,CLASS=*class name addr*	*class name addr:* A-type address **Default:** CLASS='DATASET'
,MENTITY=*entity addr*	*entity addr:* A-type address
,MCLASS='*classname*'	*'classname':* 1-8 character name.
,MCLASS=*class name addr*	*class name addr:* A-type address **Default:** MCLASS='DATASET'
,MVOLSER=*volser addr*	*volser addr:* A-type address
,MGENER=ASIS	**Default:** MGENER=ASIS
,MGENER=YES	
,ACEE=*acee addr*	*acee addr:* A-type address
,UNIT=*unit addr*	*unit addr:* A-type address
,OWNER=*owner id addr*	*owner id addr:* A-type address
,LEVEL=*number*	**Default:** zero.
,LEVEL=*reg*	*reg:* register (2) - (12).
,UACC=ALTER	
,UACC=CONTROL	
,UACC=UPDATE	
,UACC=READ	
,UACC=EXECUTE	
,UACC=NONE	

,UACC=*reg*	*reg:* register (2) - (12).
,DATA=*data addr*	*data addr:* A-type address
,AUDIT=NONE	
,AUDIT=*audit value*	*audit value:* ALL, SUCCESS, or FAILURES
,AUDIT=(*audit value (access level),audit value(access level)*)	*access level:* READ, UPDATE, CONTROL, or ALTER **Default:** READ
,AUDIT=*(reg)*	*reg:* register (2) - (12).
,RACFIND=YES	
,RACFIND=NO	
,CHKAUTH=YES	**Default:** CHKAUTH=NO
,CHKAUTH=NO	
,GENERIC=YES	**Default:** GENERIC=ASIS
,GENERIC=ASIS	
,WARNING=YES	**Default:** WARNING=NO
,WARNING=NO	**Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,FILESEQ=*reg*	*reg:* register (2) - (12).
,FILESEQ=*number*	*number:* 1-9999
,EXPDT=*expir date addr*	*expir date addr:* A-type address
,RETPD=*retn period addr*	*retn period addr:* A-type address
,EXPDTX=*ex expir date addr*	*extended expir date addr:* A-type address specifies that only verification is to be done.
,ENVIR=VERIFY	**Default:** Normal RACDEF processing.
,RESOWN=*resource owner addr*	*resource owner addr:* A-type address
,STORCLA=*storage class addr*	*storage class addr:* A-type address
,MGMTCLA=*management class addr*	*management class addr:* A-type address **Default:** see description of parameter.
,ACCLVL=*(access value addr)*	*access value addr:* A-type address
,ACCLVL=(*access value addr, parm list addr)*	*parm list addr:* A-type address
,TAPELBL=STD	**Default:** TAPELBL=STD
,TAPELBL=BLP	
,TAPELBL=NL	
,SECLVL=*addr*	*addr:* A-type address
,ERASE=YES	**Default:** ERASE=NO
,ERASE=NO	
,NOTIFY=*notify id addr*	*notify id addr:* A-type address
,MF=L	

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

,MF =L
specifies the list form of the RACDEF macro.

RACDEF (Execute Form)

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

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

ENTITY=*profile name addr*	*profile name addr:* RX-type address. **Note:** ENTITY must be specified on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* RX-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used.
,TYPE=DEFINE ,TYPE=DEFINE,NEWNAME=*new dsn addr* ,TYPE=ADDVOL,OLDVOL=*old vol addr* ,TYPE=CHGVOL,OLDVOL=*old vol addr* ,TYPE=DELETE	*new dsn addr:* RX-type address, or register (2) - (12). *old vol addr:* RX-type address, or register (2) - (12). **Default:** TYPE=DEFINE
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).
,CLASS=*class name addr*	*class name addr:* RX-type address, or register (2) - (12). **Default:** CLASS='DATASET'
,MENTITY=*entity addr*	*entity addr:* RX-type address, or register (2) - (12).
,MCLASS=*class name addr*	*class name addr:* RX-type address, or register (2) - (12). **Default:** MCLASS='DATASET'
,MVOLSER=*volser addr*	*volser addr:* RX-type address, or register (2) - (12).
,MGENER=ASIS ,MGENER=YES	**Default:** MGENER=ASIS
,ACEE=*acee addr*	*acee addr:* RX-type address, or register (2) - (12).
,UNIT=*unit addr*	*unit addr:* RX-type address, or register (2) - (12).
,OWNER=*owner id addr*	*owner id addr:* RX-type address, or register (2) - (12).
,LEVEL=*number* ,LEVEL=*reg*	**Default:** zero. *reg:* register (2) - (12).
,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=EXECUTE ,UACC=NONE ,UACC=*reg*	*reg:* register (2) - (12).

,DATA=*data addr*	*data addr:* RX-type address or register (2) - (12).
,AUDIT=NONE	
,AUDIT=*audit value*	*audit value:* ALL, SUCCESS, or FAILURES
,AUDIT=(*audit value (access level),audit value(access level)*)	*access level:* READ, UPDATE, CONTROL, or ALTER **Default:** READ
,AUDIT=*(reg)*	*reg:* register (2) - (12).
,RACFIND=YES	
,RACFIND=NO	
,CHKAUTH=YES	**Default:** CHKAUTH=NO
,CHKAUTH=NO	
,GENERIC=YES	**Default:** GENERIC=ASIS
,GENERIC=ASIS	
,WARNING=YES	**Default:** WARNING=NO
,WARNING=NO	**Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=*(number*,CHECK)	*number:* 1.8.1, 1.8, 1.7, or 1.6
,RELEASE=*number*	**Default:** RELEASE=1.6
,RELEASE=(,CHECK)	
,FILESEQ=*reg*	*reg:* register (2) - (12).
,FILESEQ=*number*	*number:* 1-9999
,EXPDT=*expir date addr*	*expir date addr:* RX-type address or register (2) - (12).
,RETPD=*retn period addr*	*retn period addr:* RX-type address or register (2) - (12).
,EXPDTX=*extended expir date addr*	*extended expir date addr:* RX-type address or register (2) - (12).
,ENVIR=VERIFY	specifies that only verification is to be done. **Default:** Normal RACDEF processing.
,RESOWN=*resource owner addr*	*resource owner addr:* RX-type address or Register (2) - (12).
,STORCLA=*storage class addr*	*storage class addr:* RX-type address or Register (2) - (12).
,MGMTCLA=*management class addr*	*management class addr:* RX-type address or Register (2) - (12).
,ACCLVL=*(access value addr)*	*access value addr:* RX-type address or register (2) - (12).
,ACCLVL=(*access value addr, parm list addr)*	*parm list addr:* RX-type address, or register (2) - (12).
,TAPELBL=STD	**Default:** TAPELBL=STD
,TAPELBL=BLP	
,TAPELBL=NL	
,SECLVL=*addr*	*addr:* RX-type address, or register (2) - (12).
,ERASE=YES	**Default:** ERASE=NO
,ERASE=NO	
,NOTIFY=*notify id addr*	*notify id addr:* RX-type address or register (2) - (12).
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF =(E,*ctrl addr*)
specifies the execute form of the RACDEF macro using a remote control program parameter list.

,RELEASE =(*number*,CHECK)
,RELEASE =1.6|1.7|1.8|1.8.1
,RELEASE= (,CHECK)

specifies the RACF release level of the parameter list to be generated by this macro.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 13.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACDEF macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code 'X64' will be generated.

RACHECK — Check RACF Authorization (for RACF Release 1.8.1 or earlier)

Note: The RACROUTE macro is the preferred programming interface.

This macro description applies to RACF Release 1.8.1 or earlier. Your program can invoke the RACHECK macro directly; however, IBM recommends that you invoke the equivalent function through the RACROUTE macro, using the REQUEST = AUTH parameter. See "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 437 for the applicable RACROUTE macro description.

If you have RACF Release 1.9 installed on your system, you can still invoke the RACHECK macro directly. However, if you are going to use the new Release 1.9 functions, see the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST = AUTH:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 447**
- **"RACROUTE REQUEST = AUTH — Check RACF Authorization (for RACF Release 1.9)" on page 467.**

The RACHECK macro provides authorization checking when a user requests access to a RACF-protected resource. The RACHECK macro is also described in the *MVS/ESA SPL: Application Development Guide*.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to use the RACHECK function, can code the RACROUTE macro.

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

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

PROFILE=*profile addr* ENTITY=(*resource name addr*) ENTITY=(*resource name addr,CSA*)	*profile addr:* A-type address, or register (2) - (12). *resource name addr:* A-type address, or register (2) - (12).
,VOLSER=*vol addr*	*vol addr:* A-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used and only when ENTITY is also coded.
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12).
,RELEASE=*number*	*number:* 1.8.1., 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* register (2) - (12). **Default:** ATTR=READ
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default:** LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr:* A-type address, or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*'applname'*: constant "Application name" *applname addr*: A-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12).
,OWNER=*owner ID addr*	*owner ID addr:* A-type address, or register (2) - (12).
,ACCLVL=(*access value addr*) ,ACCLVL=(*access value addr,* *parm list addr)*	*access value addr:* A-type address or register (2) - (12). *parm list addr:* A-type address or register (2) - (12).
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999

,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE	**Default:** STATUS=NONE
,USERID='*userid*' ,USERID=*userid addr*	*userid*: 1-8 character userid *userid addr*: A-type address or register (2) - (12)
,GROUPID='*groupid*' ,GROUPID=*groupid addr*	*groupid*: 1-8 character groupid *groupid addr:* A-type address, or register (2) - (12)

The parameters are explained as follows:

PROFILE =*profile addr*
ENTITY =*(resource name addr)*
ENTITY =*(resource name addr, CSA)*

PROFILE=*profile addr* specifies that RACF authorization checking is to be performed for the resource whose profile is pointed to by the specified address. This profile must be supplied by ENTITY=(xxx,CSA). A profile supplied by RACLIST is not acceptable.

ENTITY=*(resource name addr)* specifies that RACF authorization checking is to be performed for the resource whose name is pointed to by the specified address. The resource name is a 44-byte DASD data set name for CLASS='DATASET' or a 6-byte volume serial number for CLASS='DASDVOL' or CLASS='TAPEVOL'. The length of all other resource names is determined from the class descriptor tables. The name must be left justified in the field and padded with blanks.

ENTITY=*(resource name addr,CSA)* specifies that RACF authorization checking is to be performed for the indicated resource, and that a copy of the profile is to be maintained in main storage. The storage acquired for the profile is obtained from the common storage area (CSA), and is fetch-protected, key 0 storage. The issuer of RACHECK must free this storage when the profile is no longer needed. (The profile subpool number and length are part of the profile data returned.) If you specify CSA and the return code produced by the RACHECK macro is 00 or 08, the address of the profile is returned in register 1.

By establishing and maintaining a resource profile, the resource manager can reduce the I/O required to perform RACF authorization checks on highly-accessed resources.

,VOLSER =*vol addr*

specifies the volume serial number, as follows:

- For non-VSAM DASD data sets and tape data sets, this is the volume serial number of the volume on which the data set resides.
- For VSAM DASD data sets, this is the volume serial number of the catalog controlling the data set.

The volume serial number is optional if you specify DSTYPE=M; it is ignored if the profile name is generic.

The field to which the specified address points contains the volume serial number padded to the right with blanks, if necessary, to make six characters. VOLSER= is only valid and must be supplied with CLASS='DATASET', (unless you specify DSTYPE=M) and if you code ENTITY.

,CLASS ='*classname*'
,CLASS =*classname addr*

specifies that RACF authorization checking is to be performed for a resource of the specified class. If you specify an address, the address must point to a one-byte field indicating the length of the classname, followed by the class name.

,RELEASE =**1.6**|**1.7**|**1.8**|**1.8.1**

specifies the RACF release level of the parameter list that this macro will generate.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For instance, to use the STATUS parameter, you must be using RACF 1.7 or later on your system and specify RELEASE=1.7 or later. For the parameters that are valid for RELEASE=1.6 and later, see Figure 15.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACHECK macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,ATTR =READ
,ATTR =UPDATE
,ATTR =CONTROL
,ATTR =ALTER
,ATTR =*reg*

specifies the access authority of the user or group permitted access to the resource for which RACF authorization checking is to be performed:

READ - RACF user or group can open the resource only to read.

UPDATE - RACF user or group can open the resource to write or read.

CONTROL - For VSAM data sets, RACF user or group has authority equivalent to the VSAM control password. For non-VSAM data sets and other resources, RACF user or group has UPDATE authority.

ALTER - RACF user or group has total control over the resource.

If you specify a register, the register must contain one of the following codes in the low-order byte of the register:

X'02' - READ
X'04' - UPDATE
X'08' - CONTROL
X'80' - ALTER

,DSTYPE =N
,DSTYPE =V
,DSTYPE =M
,DSTYPE =T

specifies the type of data set associated with the request:

- N for non-VSAM
- V for VSAM
- M for model profile
- T for tape

If you specify DSTYPE=T and tape data set protection is not active, the processing will be the same as for RACHECK CLASS='TAPEVOL'.

Specify DSTYPE only for CLASS='DATASET'.

,INSTLN =*parm list addr*

specifies the address of an area that is to contain parameter information meaningful to the RACHECK installation exit routine. This information is passed to the installation exit routine when it is given control by RACHECK.

The INSTLN parameter can be used by an application program acting as a resource manager that needs to pass information to the RACHECK installation exit routine.

,LOG =ASIS
,LOG =NOFAIL
,LOG =NONE
,LOG =NOSTAT

specifies the types of access attempts to be recorded on the SMF data set:

ASIS - RACF records the event in the manner specified in the profile that protects the resource.

NOFAIL - If the authorization check fails, the attempt is not recorded. If the authorization check succeeds, the attempt is recorded as in ASIS.

NONE - The attempt is not to be recorded.

NOSTAT - The attempt is not to be recorded and no resource statistics are to be updated.

,OLDVOL =*old vol addr*

specifies a volume serial:

- For CLASS='DATASET', within the same multivolume data set specified by VOLSER=.
- For CLASS='TAPEVOL', within the same tape volume specified by ENTITY=.

RACF authorization checking will verify that the OLDVOL specified is part of the same multivolume data set or tape volume set.

The specified address points to the field that contains the volume serial number padded to the right with blanks, if necessary, to make six characters.

,APPL ='*applname*'
,APPL =*applname addr*

specifies the name of the application requesting authorization checking. The *applname* is not used for the authorization checking process but is made available to the installation exit routine(s) called by the RACHECK routine. If you specify the address, the address must point to an 8-byte field containing the application name left justified and padded with blanks.

,ACEE =*acee addr*

specifies the address of the accessor environment element (ACEE) to be used during RACHECK processing. If you do not specify ACEE, RACF uses the TASK ACEE pointer (TCBSENV) in the extended TCB. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE for the address space. The ASXBSENV field of the address space extension block points to the main ACEE.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,OWNER =*owner ID addr*

specifies a profile owner id that is compared with the profile owner id in the owner field of the RACF profile. If the owner names match, the access authority allowed for that userid is 'ALTER'. The address must point to an 8-byte field containing the owner name, left-justified and padded with blanks.

If you specify OWNER, any WARNING and OPERATIONS attribute processing is bypassed.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,ACCLVL =*(access value addr)*
,ACCLVL =*(access value addr,parm list addr)*

specifies the tape label access level information for the MVS tape label functions. The access value to which the specified address points is a one byte length field, containing the value (0-8) of the length of the following data, followed by an eight-character string that will be passed to the RACHECK installation exit routines. The optional parameter list to which the specified address points contains additional information to be passed

to the RACHECK installation exit routines. RACF does not inspect or modify this information.

,RACFIND = YES
,RACFIND = NO

indicates whether or not the resource is protected by a discrete profile. The RACF processing and the possible return codes are given in Figure 14.

Note: In all cases, a return code of X'0C' is also possible if the OLDVOL specified was not part of the multivolume data set defined by VOLSER, or it was not part of the same tape volume defined by ENTITY.

Figure 14. Types of Profile Checking Performed by RACHECK

Operand	Generic Profile Checking Inactive	Generic Profile Checking Active
RACFIND=YES	Look for discrete profile; if found, exit with return code 00 or 08. If no discrete profile is found, exit with return code 08.	Look for discrete profile; if found, exit with return code 00 or 08. Look for generic profile; if found, exit with return code 00 or 08. Exit with return code 08 if neither a discrete nor a generic profile is found.
RACFIND=NO	No checking. Exit with return code 04.	Look for generic profile; if found, exit with return code 00 or 08. If not found, exit with return code 04.
RACFIND not specified	Look for discrete profile; if found, exit with return code 00 or 08. If no discrete profile is found, exit with return code 04.	Look for discrete profile; if found, exit with return code 00 or 08. Look for generic profile; if found, exit with return code 00 or 08. Exit with return code 04 if neither a discrete nor a generic profile is found.

,GENERIC = YES
,GENERIC = ASIS

specifies whether the resource name is to be treated as a generic profile name. If you specify GENERIC with CLASS=DEFINE, NEWNAME, GENERIC applies to both the old and new names. GENERIC is ignored if you do not specify the GENCMD option on the RACF SETROPTS command for the class (see *RACF Command Language Reference*).

- If you specify GENERIC=YES, the resource name is considered a generic profile name, even if it does not contain either of the generic characters: an asterisk (*) or a percent sign (%).
- If you specify GENERIC=ASIS, the resource name is considered a generic only if it contains either of the generic characters: an asterisk (*) or a percent sign (%).

,FILESEQ= *number*
,FILESEQ= *reg*

specifies the file sequence number of a tape data set on a tape volume or within a tape volume set. The value must be in the range 1 - 9999. If you specify a register, it must contain the file sequence number in the low-order half-word. If you do not specify CLASS='DATASET' and DSTYPE=T, FILESEQ is ignored.

,TAPELBL = STD|BLP|NL

specifies the type of tape label processing to be done:

- STD - IBM or ANSI standard labels.
- BLP - bypass label processing.
- NL - non-labeled tapes.

For TAPELBL=BLP, you must have the requested authority to the profile ICHBLP in the general resource class FACILITY. For TAPELBL=NL or BLP, you will not be allowed to protect volumes with volume serial numbers in the format "Lnnnnn."

This parameter is primarily intended for use by data management routines to indicate the label type from the LABEL keyword on the JCL statement.

This parameter is valid only for CLASS='DATASET' and DSTYPE=T, or CLASS='TAPEVOL'.

,STATUS =NONE|ERASE
specifies whether or not RACHECK is to return the erase status of the given data set. This parameter is valid only for CLASS='DATASET' and a DSTYPE value other than T.

,USERID= *'userid'*
,USERID= *userid addr*
specifies the userid that RACF uses to perform third party RACHECK. If you specify USERID when the caller invokes RACHECK, RACF verifies that user's authority to the given entity; RACF disregards the userid associated with the ACEE of the caller. For third party RACHECK, RACF performs the following steps:

1. Checks to see if the USERID keyword is *NONE* and that you did not specify GROUPID. If you did, then RACF creates a default user (null) ACEE which it uses to perform the RACHECK.
2. If you did not, checks to see if an additional (third party) ACEE already exists, chained off the current caller's ACEE or the ACEE specified in the "ACEE=" keyword.
3. If so, checks to see if the userid in that ACEE matches the one specified on the USERID keyword. If so, RACHECK uses the existing ACEE and avoids RACINIT processing.
4. If you specify USERID and RACHECK does not find an additional (third party) ACEE, or the userid in the ACEE does not match the userid specified on the USERID keyword, then RACHECK creates a third party ACEE based on the USERID keyword.
5. If you specify the GROUPID keyword in addition to the USERID keyword, and a third party ACEE already exists, then the groupid of the existing third party ACEE must also match the groupid specified on the GROUPID keyword. If the groupid keywords do not match, RACHECK creates a third party ACEE based on the USERID keyword.

Note: If the calling program does not specify the GROUPID keyword, the internal RACINIT function will use the default group associated with the specified userid.

Only programs that are that are APF-authorized, system key 0-7, or in supervisor state, can use the USERID and GROUPID keywords.

Note: If the userid is *NONE* and you have not specified a GROUPID, a default user (null) ACEE is created and used to satisfy RACHECK processing.

,GROUPID= *'groupid'*
,GROUPID= *groupid addr*
specifies the groupid that RACF uses to perform third party RACHECK.

If the calling program wants a third party RACHECK performed on the GROUPID rather than the USERID, then the USERID keyword must be specified as *NONE*. Thus, when the caller invokes third party RACHECK, RACF verifies the authority of the groupid to the requested resource; RACF disregards the groupid associated with the ACEE of the caller. For third party RACHECK, RACF performs the following steps:

- Checks to see if an additional (third party) ACEE already exists, chains off the caller's ACEE, or the ACEE specified in the "ACEE=" keyword.
- If so, checks to see if the groupid matches that specified on the GROUPID keyword. If so, RACHECK uses that ACEE and avoids RACINIT processing.
- If you specify GROUPID and RACHECK does not find an additional (third party) ACEE, or the groupid in the ACEE does not match the groupid specified on the GROUPID keyword, then RACHECK creates a third party ACEE based on the GROUPID keyword.

Only programs that are that are APF-authorized, system key 0-7, or in supervisor state, can use the USERID and GROUPID keywords.

Parameters for RELEASE = 1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Figure 15. RACHECK Parameters for RELEASE = 1.6 and Later

Parameter	RELEASE = 1.6	RELEASE = 1.7	RELEASE = 1.8 or 1.8.1
ACEE =	X	X	X
ACCLVL =	X	X	X
APPL =	X	X	X
ATTR =	X	X	X
CLASS =	X	X	X
DSTYPE = N, V, or M	X	X	X
DSTYPE = T		X	X
ENTITY =	X	X	X
FILESEQ =		X	X
GENERIC =	X	X	X
GROUPID =			X
INSTLN =	X	X	X
LOG =	X	X	X
OLDVOL =	X	X	X
OWNER =	X	X	X
PROFILE =	X	X	X
RACFIND =	X	X	X
RELEASE =	X	X	X
STATUS =		X	X
TAPELBL =		X	X
USERID =			X
VOLSER =	X	X	X

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	The user is authorized by RACF to obtain use of a RACF-protected resource. Register 0 contains one of the following reason codes: 00 indicates a normal completion. 04 indicates STATUS=ERASE was specified and the data set is to be erased when scratched. PRODUCT-SENSITIVE PROGRAMMING INTERFACE Or the warning status of the resource was requested by the RACHECK issuer setting bit '10' at offset 12 decimal in the RACHECK parameter list and the resource is in warning mode. End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE 10 When CLASS=TAPEVOL, indicates the tapevol profile contains a TVTOC. 20 When CLASS=TAPEVOL, indicates that the tapevol profile can contain a TVTOC, but currently does not. (Scratch pool volume) 24 When CLASS=TAPEVOL, indicates that the tapevol profile does not contain a TVTOC.
04	The specified resource is not protected by RACF. Register 0 contains the following reason code: 00 indicates a normal completion.
08	The user is **not** authorized by RACF to obtain use of the specified RACF-protected resource. Register 0 contains the following reason code: 00 indicates a normal completion. 04 indicates STATUS=ERASE was specified and the data set is to be erased when scratched. 08 indicates DSTYPE=T or CLASS='TAPEVOL' was specified and the user is not authorized to use the specified volume. 0C indicates the user is not authorized to use the data set. 10 indicates DSTYPE=T or CLASS='TAPEVOL' was specified and the user is not authorized to specify LABEL=(,BLP). 1C User with EXECUTE authority to the data set profile specified ATTR=READ, and RACF failed the access attempt.
0C	The OLDVOL specified was not part of the multivolume data set defined by VOLSER, or it was not part of the same tape volume defined by ENTITY.
10	RACINIT issued by third party RACHECK failed. Register 0 contains the RACINIT return code.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACHECK macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example 1

Operation: Perform RACF authorization checking using the standard form, for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on. Logging and statistics updates are **not** to be done.

```
RACHECK  ENTITY=((R7)),VOLSER=(R8),CLASS='DATASET',                X
         ATTR=ALTER,RACFIND=YES,LOG=NOSTAT
```

Example 2

Operation: Perform RACF authorization checking using the standard form, for a non-VSAM data set controlled by the catalog pointed to by register 8. Register 7 points to the data set name, and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on.

```
RACHECK ENTITY=((R7)),VOLSER=(R8),CLASS='DATASET',                 X
               ATTR=ALTER,RACFIND=YES
```

Example 3

Operation: Perform RACF authorization checking using the standard form, for a VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name, and the RACF user is requesting the data set for read only. Register 4 points to an area containing additional parameter information.

```
RACHECK ENTITY=((R7)),VOLSER=(R8),CLASS='DATASET',                 X
               DSTYPE=V,INSTLN=(R4)
```

Example 4

Operation: Using the standard form, perform RACF authorization checking for a tape volume for read access only. The tape volume is pointed to by register 8 and the volume's access level is in register 5.

```
RACHECK ENTITY=((R8)),CLASS='TAPEVOL',ATTR=READ,                   X
               ACCLVL=((R5))
```

Example 5

Operation: Using the standard form, perform third party RACF authorization checking for a data set for READ access only for a user. Register 7 points to the data set name.

```
RACHECK ENTITY=((R7)),CLASS='DATASET',ATTR=READ                    X
               USERID='SOMEUSER'
```

Example 6

Operation: Using the standard form, perform third party RACF authorization checking for a data set for READ access only for a group. Register 7 points to the data set name.

```
RACHECK ENTITY=((R7)),CLASS='DATASET',ATTR=READ                    X
               USERID='*NONE*',GROUPID='SOMEGROUPID'
```

Example 7

Operation: Using the standard form, perform third party RACF authorization checking for a data set for a user connected to a group. Register 7 points to the data set name.

```
RACHECK ENTITY=((R7)),CLASS='DATASET',ATTR=READ                    X
               USERID='SOMEUSER',GROUPID='SOMEGROUPID'
```

RACHECK (List Form)

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

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

PROFILE=*profile addr* ENTITY=(*resource name addr*) ENTITY=(*resource name addr*,CSA)	*profile addr:* A-type address. *resource name addr:* A-type address. **Note:** PROFILE or ENTITY is required on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* A-type address. **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address.
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* register (2) - (12). **Default:** ATTR=READ
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* A-type address.
,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default:** LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr:* A-type address.
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address.
,ACEE=*acee addr*	*acee addr:* A-type address.
,OWNER=*owner ID addr*	*owner ID addr:* A-type address.
,ACCLVL=(*access value addr*) ,ACCLVL=(*access value addr*, *parm list addr*)	*access value addr:* A-type address *parm list addr:* A-type address
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS

,FILESEQ=*reg*	*reg:* register (2) - (12).
,FILESEQ=*number*	*number:* 1-9999
,TAPELBL=STD	**Default:** TAPELBL=STD
,TAPELBL=BLP	
,TAPELBL=NL	
,STATUS=NONE	**Default:** STATUS=NONE
,STATUS=ERASE	
,USERID='*userid*'	*userid*: constant RACF userid
,USERID=*userid addr*	*userid addr:* A-type address
,GROUPID='*groupid*'	*groupid*: constant RACF group id
,GROUPID=*groupid addr*	*groupid addr:* A-type address

,MF=L

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

,MF=L
specifies the list form of the RACHECK macro.

RACHECK (Execute Form)

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

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

PROFILE=*profile addr*	*profile addr:* RX-type address, or register (2) - (12).
,ENTITY=(*resource name addr*) ,ENTITY=(*resource name addr*,CSA)	*resource name addr:* RX-type address, or register (2) - (12). **Note:** PROFILE or ENTITY is required on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* RX-type address, or register (2) - (12). **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS=*class name addr*	*class name addr:* RX-type address, or register (2) - (12).
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* register (2) - (12). **Default:** ATTR=READ
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).
,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default:** LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr:* RX-type address, or register (2) - (12).
,APPL=*applname* ,APPL=*applname addr*	*applname addr:* RX-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* RX-type address, or register (2) - (12).
,OWNER=*owner ID addr*	*owner ID addr:* RX-type address, or register (2) - (12).
,ACCLVL=(*access value addr*) ,ACCLVL=(*access value addr)*	*access value addr:* RX-type address or register (2) - (12). *access value addr*: RX-type address or register (2) - (12).
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS

,FILESEQ=*reg*	*reg:* register (2) - (12).
,FILESEQ=*number*	*number:* 1-9999
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE	**Default:** STATUS=NONE
,USERID=*userid* ,USERID=*userid addr*	*userid*: RACF userid declared as a constant *userid addr:* RX-type address or register (2) - (12).
,GROUPID=*groupid* ,GROUPID=*groupid addr*	*groupid*: RACF group id declared as a constant *groupid addr:* RX-type address, or register (2) - (12).
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF =(E,*ctrl addr)*
specifies the execute form of the RACHECK macro.

,RELEASE =(*number*,**CHECK)**
,RELEASE =1.6|1.7|1.8|1.8.1
,RELEASE= (,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 15.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACHECK macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

RACINIT — Identify a RACF-Defined User (for RACF Release 1.8.1 or earlier)

Note: The RACROUTE macro is the preferred programming interface.

This macro description applies to RACF Release 1.8.1 or earlier. Your program can invoke the RACINIT macro directly; however, IBM recommends that you invoke the equivalent function through the RACROUTE macro, using the REQUEST = VERIFY parameter. See "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 437 for the applicable RACROUTE macro description.

If you have RACF Release 1.9 installed on your system, you can still invoke the RACINIT macro directly. However, if you are going to use the new Release 1.9 functions, see the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST = VERIFY:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 447**
- **"RACROUTE REQUEST = VERIFY — Identify a RACF-Defined User (for RACF Release 1.9)" on page 581.**

The RACINIT macro provides and verifies a Resource Access Control Facility (RACF) user. The macro identifies a user and verifies that the user is defined to RACF and has supplied a valid password and/or operator identification card (OIDCARD parameter).

To issue the RACINIT macro the calling module must be "authorized" which means

- APF-authorized, or
- in system key 0-7, or
- in supervisor state.

or you must omit the NEWPASS keyword and the calling module must:

- be in the RACF-authorized caller table *and*
- fetched from an authorized library *and*
- reentrant.

Note: It is recommended that if you run programs which issue the RACINIT macro, you run those programs AFP-authorized.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode who want to use the RACINIT function can code the RACROUTE macro.

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

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

USERID=*userid addr*	*userid addr:* A-type address, or register (2) - (12).
,PASSWRD=*password addr*	*password addr:* A-type address, or register (2) - (12).
,START=*procname addr*	*procname addr:* A-type address, or register (2) - (12).
,NEWPASS=*new password addr*	*new password addr:* A-type address, or register (2) - (12).
,GROUP=*group addr*	*group addr:* A-type address, or register (2) - (12). **Default:** zero.
,PGMNAME=*programmer name addr*	*programmer name addr:* A-type address, or register (2) - (12).

,ACTINFO=*account addr*	*account addr:* A-type address, or register (2) - (12).
,OIDCARD=*oid addr*	*oid addr:* A-type address, or register (2) - (12).
,TERMID=*terminal addr*	*terminal addr:* A-type address, or register (2) - (12).
,JOBNAME=*jobname addr*	*jobname addr:* A-type address, or register (2) - (12).
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default:** ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12).
,SUBPOOL=*subpool number*	*subpool number:* decimal digit 0-255.
,SMC=YES ,SMC=NO	**Default:** SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default:** PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default:** ENCRYPT=YES
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS

The parameters are explained as follows:

USERID =*userid addr*
identifies the user who has entered the system. The address points to a one-byte length field, followed by the userid.

,PASSWRD =*password addr*
specifies the currently defined password of the user who has entered the system. The address points to a one-byte length field, followed by the password.

,START =*procname addr*
specifies the PROC name of a started task. The address points to an eight-byte area containing the PROC name (left-justified and padded with blanks, if necessary). If you do not specify the USERID keyword, but do specify the START keyword, RACF searches the started procedure table to determine the userid.

,NEWPASS =*new password addr*
specifies the password which is to replace the user's currently defined password. The address points to a one-byte length field, followed by the password.

,GROUP =*group addr*
specifies the group specified by the user who has entered the system. The address points to a one-byte length field, followed by the group name.

,PGMNAME =*programmer name addr*
specifies the address of the name of the user who has entered the system. This twenty-byte area is passed to the RACINIT installation exit routine; the RACINIT routine does not use it.

,ACTINFO =*account addr*
specifies the address of a field containing accounting information. This 144 byte area is passed to the RACINIT installation exit routine; the RACINIT routine does not use it. The accounting field, if supplied, should have the following format:

- First byte of field contains the number (binary) of accounting fields.
- Following bytes contain accounting fields, where each entry for an accounting field contains a one-byte length field, followed by the field.

,OIDCARD =*oid addr*
specifies the address of the currently defined operator identification card of the user who has entered the system. The address points to a one-byte length field, followed by the operator ID card.

,TERMID =*terminal addr*
specifies the address of the identifier for the terminal through which the user is accessing the system. The address points to an eight byte area containing the terminal identifier. The area must reside in a non-task-related storage subpool.

,JOBNAME =*jobname addr*
specifies the address of the JOB name of a background job. The address points to an eight byte area containing the JOB name (left justified and padded with blanks, if necessary). RACINIT authorization checking does not use the JOBNAME parameter, but passes it to the installation exit routine.

,ENVIR =CREATE
,ENVIR =CHANGE
,ENVIR =DELETE
specifies the action that the user initialization component will perform regarding the accessor environment element (ACEE):

CREATE - Verify the user and create an ACEE.

CHANGE - Modify the ACEE according to other parameters specified on RACINIT.

DELETE - Delete the ACEE. Use this parameter only if a previous RACINIT has completed successfully.

,INSTLN =*parm list addr*
specifies the address of an area containing parameter information meaningful to the RACINIT installation exit routine. This area is passed to the installation exit when the exit routine is given control from the RACINIT routine.

An installation can use the INSTLN parameter if it has a user verification or a job initiation application, and it wants to pass information from one installation module to the RACINIT installation exit routine.

,APPL ='*applname*'
,APPL =*applname addr*
specifies the name of the application issuing the RACINIT. If you specify an address, the address must point to an 8-byte application name, left justified and padded with blanks, if necessary.

,ACEE =*acee addr*
specifies the address of the ACEE.

For ENVIR=DELETE: specifies the address of a fullword that contains the address of the accessor environment element (ACEE) to be deleted. If you do not specify ACEE=, and the TCBSENV field for the task using the RACINIT is non-zero, the ACEE to which

the TCBSENV points is deleted, and TCBSENV is set to zero. If the TCBSENV and ASXBSENV fields both point to the same ACEE, then ASXBSENV is also set to zero. If no ACEE address is passed, and TCBSENV is zero, the ACEE to which ASXBSENV points is deleted, and ASXBSENV is set to zero.

For ENVIR=CHANGE: specifies the address of a fullword that contains the address of the accessor environment element (ACEE) to be changed. If you do not specify ACEE=, and the TCBSENV field for the task using the RACINIT is non-zero, the ACEE to which the TCBSENV points is changed. If TCBSENV is 0, then the ACEE to which ASXBSENV points is changed.

For ENVIR=CREATE: specifies the address of a full word into which the RACINIT function will place the address of the ACEE created. If you do not specify an ACEE, the address of the newly created ACEE is stored in the TCBSENV field of the task control block. If the ASXBSENV field is set to binary zeros, the new ACEE address is also stored in the ASXBSENV field of the ASXB. If the ASXBSENV field is non-zero, it is not modified. The TCBSENV field is set unconditionally.

Notes:

1. If you omit USERID, GROUP, and PASSWRD and if you code ENVIR=CREATE or use ENVIR=CREATE as the default, you will receive a return code of X'00' and obtain an ACEE that contains an * (X'5C') in place of the USERID and group name.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

2. If you specify ACEE with ENVIR=CREATE, RACF suppresses the creation of a modeling table (MDEL) to reduce the amount of CSA and/or LSQA storage required by IMS/VS and CICS/VS installations.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,SUBPOOL =*subpool number*
specifies the storage subpool from which the ACEE and related storage are obtained.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,SMC =YES
,SMC =NO
specifies the use of the step-must-complete function of RACINIT processing. SMC=YES specifies that RACINIT processing should continue to place other tasks for the step non-dispatchable. SMC=NO specifies that the step-must-complete function is not used.

Note: Do not use SMC=NO if DADSM ALLOCATE/SCRATCH functions execute simultaneously in the same address space as the RACINIT function.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,PASSCHK =YES
,PASSCHK =NO
specifies whether or not the user's password is to be verified. PASSCHK=YES specifies that RACINIT verifies the user's password. PASSCHK=NO specifies that the user's password is not verified.

,ENCRYPT =YES
,ENCRYPT =NO
specifies whether or not RACINIT will encrypt the old password, the new password, and the OIDCARD data passed to it.

YES signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are not pre-encrypted. RACINIT encrypts the data before storing it in the user profile or using it to compare against stored data. ENCRYPT=YES is the default for this keyword.

NO signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are already encrypted. RACINIT bypasses the encryption of this data before storing it in, or comparing it against, the user profile.

Note: The exit routine ICHDEX01 can also perform the encryption.

,RELEASE =1.6|1.7|1.8|1.8.1
specifies the RACF release level of the parameter list that this macro will generate.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE = 1.6 and later, see Figure 16.

The default is RELEASE = 1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACINIT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,STAT =ASIS|NO
specifies whether the statistics controlled by the installation's options on the RACF SETROPTS command are to be maintained or ignored for this execution of RACINIT. This parameter also controls whether a message is to be issued when the logon is successful.

Note: Messages are always issued if the RACINIT processing is unsuccessful.

If you specify or default to STAT = ASIS, the installation's current options on the RACF SETROPTS command control the messages and statistics.

If you specify STAT = NO, the statistics are not updated. And, if the logon is successful, no message is issued.

The default is STAT = ASIS.

,LOG =ASIS|ALL
specifies when log records are to be generated.

If you specify or default to LOG = ASIS, only those attempts to create an ACEE that fail will generate RACF log records.

If you specify LOG = ALL, any request to create an ACEE, regardless of whether it succeeds or fails, will generate a RACF log record. The default is LOG = ASIS.

Parameters for RELEASE = 1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Figure 16. RACINIT Parameters for RELEASE = 1.6 and Later

Parameter	RELEASE = 1.6	RELEASE = 1.7	RELEASE = 1.8 or 1.8.1
ACEE =	X	X	X
ACCTINFO =	X	X	X
APPL =	X	X	X
ENCRYPT =	X	X	X
ENVIR =	X	X	X
GROUP =	X	X	X
INSTLN =	X	X	X
JOBNAME =	X	X	X
LOG =		X	X
NEWPASS =	X	X	X
OIDCARD =	X	X	X
PASSCHK =	X	X	X
PASSWRD =	X	X	X
PGMNAME =	X	X	X
RELEASE =	X	X	X
SMC =	X	X	X
START =	X	X	X
STAT =		X	X
SUBPOOL =	X	X	X
TERMID =	X	X	X
USERID =	X	X	X

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	RACINIT has completed successfully.
04	The user profile is not defined to RACF.
08	The password is not authorized.
0C	The password has expired.
10	The new password is invalid.
14	The user is not defined to the group.
18	RACINIT was failed by the installation exit routine.
1C	The user access has been revoked.
20	RACF is not active.
24	The user's access to the specified group has been revoked.
28	OIDCARD parameter is required but not supplied.
2C	OIDCARD parameter is invalid for specified user.
30	The user is not authorized to use the terminal. Register 0 contains one of the following reason codes: 00 indicates a normal completion. 04 indicates the user is not authorized to access the system on this day, or at this time of day. 08 indicates the terminal may not be used on this day, or at this time of day.
34	The user is not authorized to use the application.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACINIT macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example 1

Operation: Use the standard form of the macro to do the following:

- Create an ACEE for the userid and its default group
- Chain the ACEE off either the current TCB or ASXB, or both, by not specifying the ACEE keyword
- Verify that the user named USERNAME is a valid user
- Verify that the password called PASSWORD is valid

```
RACINIT ENVIR=CREATE,USERID=USERNAME,PASSWRD=PASSWORD
```

Example 2

Operation: Use the standard form to do the following:

- Verify that the user named USERNAME is a valid user
- Verify that the group named GROUPNAM is a valid group
- Verify that USERNAME is defined to the group
- Create an ACEE for the user and group and put its address in ACEEANCH
- Specify that the user's password is not required

```
RACINIT ENVIR=CREATE,USERID=USERNAME,GROUP=GROUPNAM,ACEE=ACEEANCH,     X
        PASSCHK=NO
```

Example 3

Operation: Use the standard form of the macro to delete the accessor environment (ACEE) of the current task or address space, or both.

```
RACINIT ENVIR=DELETE
```

RACINIT (List Form)

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

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

USERID=*userid addr*	*userid addr:* A-type address.
,PASSWRD=*password addr*	*password addr:* A-type address.
,START=*procname addr*	*procname addr:* A-type address.
,NEWPASS=*new password addr*	*new password addr:* A-type address.
,GROUP=*group addr*	*group addr:* A-type address.
,PGMNAME=*programmer name addr*	*programmer name addr:* A-type address.
,ACTINFO=*account addr*	*account addr:* A-type address.
,OIDCARD=*oid addr*	*oid addr:* A-type address
,TERMID=*terminal addr*	*terminal addr:* A-type address.
,JOBNAME=*jobname addr*	*jobname addr:* A-type address.
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default:** ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr:* A-type address.
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address.
,ACEE=*acee addr*	*acee addr:* A-type address.
,SUBPOOL=*subpool number*	*subpool number:* decimal digit 0-255.
,SMC=YES ,SMC=NO	**Default:** SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default:** PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default:** ENCRYPT=YES
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS

,LOG=ASIS
,LOG=ALL

Default: LOG=ASIS

,MF=L

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

,MF =L
specifies the list form of the RACINIT macro.

RACINIT (Execute Form)

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

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

USERID=*userid addr*	*userid addr:* RX-type address, or register (2) - (12).
,PASSWRD=*password addr*	*password addr:* RX-type address, or register (2) - (12).
,START=*procname addr*	*procname addr:* RX-type address, or register (2) - (12).
,NEWPASS=*new password addr*	*new password addr:* RX-type address, or register (2) - (12).
,GROUP=*group addr*	*group addr:* RX-type address, or register (2) - (12). **Default:** zero.
,PGMNAME=*programmer name addr*	*programmer name addr:* RX-type address, or register (2) - (12).
,ACTINFO=*account addr*	*account addr:* RX-type address, or register (2) - (12).
,OIDCARD=*oid addr*	*oid addr:* RX-type address, or register (2) - (12).
,TERMID=*terminal addr*	*terminal addr:* RX-type address. or register (2) - (12).
,JOBNAME=*jobname addr*	*jobname addr:* RX-type address, or register (2) - (12).
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default:** ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).
,APPL=*applname* ,APPL=*applname addr*	*applname addr:* RX-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* RX-type address, or register (2) - (12).
,SUBPOOL=*subpool number*	*subpool number:* decimal digit 0-255.
,SMC=YES ,SMC=NO	**Default:** SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default:** PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default:** ENCRYPT=YES
,RELEASE=*(number,*CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6

,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS
,MF=(E,*ctrl addr)*	*cntl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF =(E,*ctrl addr)*
specifies the execute form of the RACINIT macro using a remote control program parameter list.

,RELEASE =(*number*,**CHECK)**
,RELEASE =1.6|1.7|1.8|1.8.1
,RELEASE= (,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. If you specify a parameter with an incompatible release level, the parameter will not be accepted by macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 16.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACINIT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code 'X64' will be generated.

RACLIST — Build In-Storage Profiles (for RACF Release 1.8.1 or earlier)

Note: The RACROUTE macro is the preferred programming interface.

This macro description applies to RACF Release 1.8.1 or earlier. Your program can invoke the RACLIST macro directly; however, IBM recommends that you invoke the equivalent function through the RACROUTE macro, using the REQUEST=LIST parameter. See "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 437 for the applicable RACROUTE macro description.

If you have RACF Release 1.9 installed on your system, you can still invoke the RACLIST macro directly. However, if you are going to use the new Release 1.9 functions, see the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST=LIST:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 447**
- **"RACROUTE REQUEST=LIST — Build In-Storage Profiles (for RACF Release 1.9)" on page 543.**

RACLIST builds in-storage profiles for RACF defined resources. RACLIST processes only those resources described by class descriptors. The primary advantage of using the RACLIST macro is to use the resource grouping function and to improve resource authorization checking performance.

The module calling the RACLIST macro must either be authorized (APF-authorized, in system key 0-7, or in supervisor state) or re-entrant in the RACF-authorized caller table and fetched from an authorized library.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to use the RACLIST function, can code the RACROUTE macro.

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

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

CLASS=*'classname'* CLASS=*classname addr*	*classname addr:* A-type address or register (2) - (12).
,LIST=*list addr*	*list addr:* A-type address or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* A-type address or register (2) - (12).
,INSTLN=*parm list addr*	*parm list addr:* A-type address or register (2) - (12).
,APPL=*'applname'* ,APPL=*applname addr*	*applname addr:* A-type address or register (2) - (12).
,SUBPOOL=*(sub#1,sub#2)*	
,ENVIR=CREATE ,ENVIR=DELETE	**Default:**
,OWNER=YES ,OWNER=NO	**Default:** OWNER=NO
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6

The parameters are explained as follows:

CLASS = *'classname'*
CLASS = *classname addr*
specifies that RACLIST is to build an in-storage profile for the resources of the specified class. If you specify an address, the address must point to an 8-byte field containing the class name, left justified and padded with blanks, if necessary. A class descriptor must define the class name; if not, the class is not considered to be defined.

,LIST = *addr*
specifies the address of a list of resource names for which RACLIST is to build the in-storage profiles. The list consists of a 2-byte field containing the number of the names in the list, followed by one or more variable length names. Each name consists of a 1-byte length field, which is the length of the name, followed by the name. A zero in the 2-byte field causes the operand to be omitted. If LIST= is omitted, in-storage profiles are built for all the profiles defined to RACF in the given class as well as each member for a resource grouping associated with the specified class.

Note: You can specify this operand only with ENVIR=CREATE. If you specify ENVIR=DELETE, the RACLIST macro issues a return code of 18.

,ACEE = *acee addr*
specifies the address of the accessor control environment element (ACEE). The ACEE points to the in-storage profiles. If you do not specify an ACEE, RACF uses the TASK ACEE pointer in the extended TCB called the TCBSENV. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE to obtain the list of the in-storage profiles. The ASXBSENV field of the address space extension block points to the main ACEE. If you do not specify an ACEE and there is no main ACEE, the in-storage profiles are not constructed.

,INSTLN = *parm list addr*
specifies the address of an area that contains parameter information for the RACLIST installation exit. The address is passed to the installation exit when the RACLIST routine gives control to the exit. An application or an installation program can use the INSTLN parameter to pass information to the RACLIST installation exit.

,APPL = *'applname'*
,APPL = *applname addr*
specifies the name of the application requesting the authorization checking. This information is not used for the authorization checking process but is made available to the installation exit(s). If you specify an address, it should point to an 8-byte area containing the application name, left justified and padded with blanks, if necessary.

,SUBPOOL = *(sub#1,sub#2)*
specifies the subpool numbers of the storage into which the components of the in-storage profiles are to be built. Sub#1 represents the subpool of the profile index. Sub#2 represents the subpool of the profile proper. If you do not specify the subpools you default to subpool 255. You can use registers to specify sub#1 and sub#2.

,ENVIR = **CREATE**
,ENVIR = **DELETE**
specifies the action to be performed by the RACLIST macro.

CREATE - In-storage profiles for the specified class are to be built. The RACLIST function issues a return code of 18 if an in-storage list currently exists for the specified class.

DELETE - The in-storage profiles for the specified class are to be freed. If you do not specify class, the in-storage profiles for all classes are freed.

Note: It is the responsibility of the user issuing the RACLIST macro to assure that no multi-tasking that results in the issuing of a RACHECK, FRACHECK, RACINIT, or RACLIST macro occurs at the same time that the RACLIST occurs.

,OWNER =YES
,OWNER =NO
specifies that the resource owner is to be placed in the profile access list with the ALTER authority. If the OWNER= operand is omitted, the default is NO.

,RELEASE 1.6|1.7|1.8|1.8.1
specifies the RACF release level of the parameter list that this macro will generate.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time.

For the parameters that are valid for RELEASE=1.6 and later, see Figure 17.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACLIST macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

Parameters for RELEASE=1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Figure 17. RACLIST Parameters for RELEASE=1.6 and Later

Parameter	RELEASE=1.6	RELEASE=1.7	RELEASE=1.8 or 1.8.1
ACEE=	X	X	X
APPL=	X	X	X
CLASS=	X	X	X
ENVIR=	X	X	X
INSTLN=	X	X	X
LIST=	X	X	X
OWNER=	X	X	X
RELEASE=	X	X	X
SUBPOOL=	X	X	X

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	RACLIST function completed successfully.
04	Unable to perform the requested function. Register 0 contains additional codes as follows: 0 - Unable to establish an ESTAE environment. 1 - The function code (the third byte of the parameter list) does not represent a valid function. '01' represents the RACF manager; '02' represents the RACLIST macro.
08	The specified class is not defined to RACF.
0C	An error was encountered during RACLIST processing.
10	RACF and/or the resource class is not active.
14	RACLIST installation exit error occurred.
18	Parameter list error.
1C	RACF CVT does not exist (RACF is not installed) or an insufficient level of RACF is installed.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACLIST macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Note: If the resource class specified by the CLASS= operand is inactive, RACLIST does not build the in-storage profiles and a code of 0C is returned. If the resource group class is not active, RACLIST builds an in-storage profile but only from the individual resource profiles; resource group profiles are ignored.

Example 1

Operation: Use the standard form of the macro to build in-storage profiles for all the profiles in the class named CLASSNAM, and chain them off the ACEE whose address is pointed to by ACEEADDR.

```
RACLIST CLASS=CLASSNAM,ACEE=ACEEADDR,ENVIR=CREATE
```

Example 2

Operation: Use the standard form of the macro to build in-storage profiles for all the profiles whose names are in a list named PROFLIST and a class named CLASSNAM. Chain them from the task ACEE or address space ACEE.

```
RACLIST CLASS=CLASSNAM,LIST=PROFLIST,ENVIR=CREATE
```

Example 3

Operation: Use the standard form of the macro to delete the in-storage profiles for the CLASSNAM class.

```
RACLIST CLASS=CLASSNAM,ENVIR=DELETE
```

RACLIST (List Form)

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

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

CLASS='*classname*' CLASS=*classname addr*	*classname addr:* A-type address.
,LIST=*list addr*	*list addr:* A-type address.
,ACEE=*acee addr*	*acee addr:* A-type address.
,INSTLN=*parm list addr*	*parm list addr:* A-type address.
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address.
,SUBPOOL=*(sub#1,sub#2)*	**Default:** 255.
,ENVIR=CREATE ,ENVIR=DELETE	**Default:** ENVIR=CREATE
,OWNER=YES ,OWNER=NO	**Default:** OWNER=NO
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,MF=L	

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

,MF=L
specifies the list form of the RACLIST macro.

RACLIST (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACLIST.
RACLIST	
b	One or more blanks must follow RACLIST.
CLASS=*classname addr*	*classname addr:* RX-type address or register (2) - (12).
,LIST=*list addr*	*list addr:* RX-type address or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* RX-type address or register (2) - (12).
,INSTLN=*parm list addr*	*parm list addr:* RX-type address or register (2) - (12).
,APPL=*appIname addr*	*appIname addr:* RX-type address or register (2) - (12).
,SUBPOOL=*(sub#1,sub#2)*	
,ENVIR=CREATE ,ENVIR=DELETE	
,OWNER=YES ,OWNER=NO	
,RELEASE=*(number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,MF=(E,,*ctrl addr*)	*ctrl addr:* RX-type address or register (2) - (12).

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

,MF =(E,*ctrl addr)*
specifies the execute form of the RACLIST macro using a remote control program parameter list.

,RELEASE= (*number*,**CHECK)**
,RELEASE= *number*
,RELEASE= (,CHECK)
specifies the RACF release level of the parameter list that this macro will generate.

You can specify certain parameters only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 17.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACLIST macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code of 'X64' will be generated.

RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)

This macro description applies to RACF Release 1.8.1 or earlier. If you have RACF Release 1.9 installed on your system, and you are going to use the new Release 1.9 functions, see "RACROUTE — Router Interface (for RACF Release 1.9)" on page 445.

The RACROUTE macro invokes the System Authorization Facility (SAF) MVS router. Depending on how your installation has written the MVS router exit, and whether RACF is present, the MVS router directs control to the RACF router. If RACF is active, the RACF router then invokes RACF.

Note: Various RACF functions invoked by RACROUTE require that you specify the CLASS parameter, and that the specified CLASS be active. With few exceptions, for the IBM-supplied portion of the table, the class specified on the CLASS parameter *must* be active for the RACF router to invoke RACF. In the case of the installation-supplied portion of the table, there are no exceptions; the class specified on the CLASS parameter *must* be active for the RACF router to invoke RACF.

You can use RACROUTE to access the functions that the following RACF macros provide: RACDEF, RACINIT, RACXTRT, RACLIST, RACHECK, and FRACHECK. In coding the RACROUTE macro to access a particular RACF macro function, you must also use the necessary parameters from that macro on the RACROUTE macro. For example, if you code RACROUTE to access the RACHECK function, you must code REQUEST = AUTH and any other required parameters and any optional ones you need from the RACHECK macro. RACROUTE validates that only the parameters applicable to the RACHECK macro have been coded.

Notes:

1. For RACF Version 1 Release 6 and earlier, all parameters and parameter lists must reside below 16 megabytes.
2. For RACF Version 1 Release 7 and later:
   If a caller is executing in 24-bit addressing mode, all parameters and parameter lists are assumed to reside below 16 megabytes. If a caller, however, is executing in 31-bit addressing mode, and is calling RACF via the RACROUTE macro, RACF will assume that all parameters and parameter lists may reside above the 16 megabytes (that is, that all parameter addresses are true 31-bit addresses).

   All parameter lists generated by the RACROUTE macro are in a format that allows compiled code to be moved above 16 megabytes without recompilation.

   This 31-bit support is available only when RACF is called via RACROUTE, FRACHECK, or RACSTAT. Any caller that uses RACINIT, RACDEF, RACLIST or RACHECK must be in 24-bit addressing mode only. RACF does not support those callers in 31-bit mode.

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

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

REQUEST=AUTH
REQUEST=FASTAUTH
REQUEST=DEFINE
REQUEST=VERIFY
REQUEST=LIST
REQUEST=EXTRACT

,REQSTOR=*reqstor addr*	*reqstor addr:* A-type address or register (2) - (12). **Default:** zero. **Note:** If REQSTOR= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,SUBSYS=*subsys addr*	*subsys addr:* A-type address or register (2) - (12). **Note:** If SUBSYS= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,WORKA=*work area addr*	*work area addr:* A-type address or register (2) - (12).
,RELATED=*value*	*value:* Any valid macro keyword specified.
,ENVIR=VERIFY	**Note:** ENVIR can be coded only if REQUEST=VERIFY is coded.
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default:** See parameter description. **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,MSGRTRN=YES ,MSGRTRN=NO	**Default**=NO
,MSGSUPP=YES ,MSGSUPP=NO	**Default**=NO
,MSGSP=subpool number	**Decimal digit 0-255**
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6

Note: In addition to the parameters described above, all parameters valid on the RACDEF, RACLIST, RACINIT, RACXTRT, RACHECK, and FRACHECK macros are permitted on the RACROUTE macro. Depending on the parameter REQUEST=, some of these are required, some optional, and some are invalid.

The parameters are explained as follows:

REQUEST = AUTH
REQUEST = FASTAUTH
REQUEST = DEFINE
REQUEST = VERIFY
REQUEST = LIST
REQUEST = EXTRACT
specifies a code that determines the RACF parameter list to be issued internally as well as the RACF routine to receive control. The permissible codes and their associated RACF macros are as follows:

AUTH -- RACHECK
FASTAUTH -- FRACHECK
DEFINE -- RACDEF
VERIFY -- RACINIT
LIST -- RACLIST
EXTRACT -- RACXTRT

For RACROUTE to work correctly, once you have chosen a REQUEST code you must also code (on the RACROUTE macro) the parameters that belong to the associated macro. Please see the associated macro for the necessary parameters.

Notes:

1. Data areas that RACF returns to the caller will be either above or below 16-megabytes, depending upon the caller's addressing mode and the data area in question.
2. Unless the caller specifies the ACEE= parameter on a "RACROUTE REQUEST=VERIFY,ENVIR=CREATE" macro, the ACEE will always be placed below 16-megabytes.
3. If the caller specifies the ACEE= parameter, and is executing in 31-bit addressing mode and does not specify LOC=BELOW on the RACROUTE macro, the ACEE will be placed, if possible, above 16-megabytes.
4. If the ACEE is below 16-megabytes, any area, with the exception of generic profiles, chained off an ACEE (for example, RACLIST profiles, list-of-groups table) will be placed below 16-megabytes. Otherwise, the area will be placed above the line. However, a caller executing in 31-bit mode may issue a REQUEST=LIST with LOC=ABOVE, and the profiles will be placed above the line, if possible, even if the ACEE is below the line.
5. If the caller requests that RACF return an in-storage profile in CSA as part of a "RACROUTE REQUEST=AUTH," the profile will be returned in storage below 16-megabytes if the related ACEE is located below the line. Otherwise, the area will be located above 16-megabytes.
6. The area returned by a "RACROUTE REQUEST=EXTRACT or EXTRACTN" request will be located below 16-megabytes.

,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 you code this operand and RACF is installed, you must change the RACF router table to match the operand. If you do not update the table, the default to bypass RACF processing is taken.

If you omit this operand, a string of eight blanks is assumed.

,SUBSYS = *subsys addr*
specifies the address of an 8-byte character field containing the calling subsystem's name, version, and release level. If you code this operand and RACF is installed, you must change the RACF router table to match the operand. If you do not update the table, the default to bypass RACF processing is taken.

If you omit this operand, a string of eight blanks is assumed.

,WORKA = *work area addr*
specifies the address of a 512-byte work area for use by the MVS router and the RACF front end routine.

,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 is at the discretion of the user, and can be any valid coding value.

,ENVIR = **VERIFY**
specifies that only a user verification is to be made optionally combined with updating the user's password. The installation may handle this request through a System Authorization Facility (SAF) installation exit. If this is not done, the RACROUTE caller receives a return code of 4 with RACF return and reason codes of zero.

,LOC = **BELOW**
,LOC = **ANY**
,LOC = **ABOVE**
LOC can be coded only if REQUEST = VERIFY or REQUEST = LIST is coded.

For REQUEST = VERIFY:
specifies whether the ACEE and related data areas are to be allocated storage below 16 megabytes (LOC = BELOW), or anywhere (LOC = ANY).

If any of the following is true, LOC = BELOW is the default, and LOC = ANY is ignored if specified:

- The ACEE= parameter is not coded.
- The caller is executing in 24-bit mode.

In all other cases, the default is LOC = ANY.

Note: LOC = ABOVE is invalid for REQUEST = VERIFY.

For REQUEST = LIST:
specifies whether the RACLIST profiles are to reside where the ACEE is located, above or below 16 megabytes (LOC = ANY), or whether the RACLIST profiles are to reside above 16 megabytes (LOC = ABOVE), if possible, even if the ACEE is below 16 megabytes.

Notes:

1. LOC = BELOW is invalid for REQUEST = LIST.
2. LOC = ANY does not guarantee that storage is allocated above 16 megabytes. If any installation SAF or RACF exit routines execute in 24-bit mode, the storage will be below 16 megabytes.

,MSGRTRN = **NO**|**YES**
specifies whether you want to use message return processing. You can use this parameter in conjunction with the other RACROUTE MSGxxxx parameters to store and forward RACF messages. To use this parameter, you must also specify RELEASE = 1.8 or a later release number.

Note: This parameter only applies to RACHECK, RACDEF, and RACINIT.

,MSGSUPP = **NO**|**YES**
specifies whether you want to suppress write-to-operator (WTO) messages from within RACF processing. You can use this parameter in conjunction with the other RACROUTE MSGxxxx parameters to store and forward RACF messages. To use this parameter, you must also specify RELEASE = 1.8 or a later release number.

Note: This parameter only applies to RACHECK, RACDEF, and RACINIT.

,MSGSP = *subpool number*
specifies the number of the subpool into which you want RACF messages returned. You can use ,MSGSP in conjunction with the other RACROUTE MSGxxxx parameters to store and forward RACF messages.

To use this parameter, you must specify RELEASE = 1.8.

Notes:

1. This parameter only applies to RACHECK, RACDEF, and RACINIT.
2. Only ICH408I messages can be returned.
3. When control returns from RACROUTE, the RACROUTE parameter list field is mapped by SAFPMSAD in the ICHSAFP mapping macro. SAFPMSAD will be non-zero if messages have been returned. This field will contain the address of an area which consists of two full words followed by the message itself in WTO parameter list format. The first word is the length of the area including the two full word header; the second word points to the next message area, if there is one, or contains zero if no more messages areas exist. If is your responsibility to issue the FREEMAIN macro to release these message area(s).

,RELEASE =1.6|1.7|1.8|1.8.1

specifies the release number. With Release 1.8, you can specify message parameters to obtain messages from RACHECK, RACDEF, and RACINIT. To do so, you must also specify Release = 1.8 or a later release number. If you specify the message parameters without specifying Release = 1.8, you will receive an error message.

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	The requested security function has completed successfully. In addition, if the requested function was 'AUTH', the authorization request was accepted.
04	The requested function has not been processed. In addition, if the request was 'AUTH', the MVS router could neither accept nor fail the request. The following are some possible reasons for a request not being processed. - The MVS router is not active. - The RACF front end routine detected that a null action was requested for the specified request type, resource type, and subsystem ID. - The request/resource/subsystem combination could not be found in the router table. - RACF is not active on the system, and RACFIND = YES was not specified, and there is no RACROUTE installation exit routine (or an exit originated a return code of 4). - RACF is active on the system, but no profile exists for the specified resource.
08	The requested function was processed by RACF, the MVS router, or the router exit (ICHRTX00) and failed. If the requested function was 'AUTH', the authorization request has been failed. If RACF is inactive for an 'AUTH' request with RACFIND = YES, then the MVS router fails the request. The RACF or router exit return code and reason codes are returned in the first two words of the RACROUTE input parameter list.

Example 1

Operation: Invoke the MVS router to perform authorization checking using the standard form, for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on.

```
RACROUTE  REQUEST=AUTH,WORKA=RACWK,ENTITY=((R7)),          X
          VOLSER=(R8),CLASS='DATASET',ATTR=ALTER,          X
          RACFIND=YES
  .
  .
RACWK     DS  CL512
```

Example 2

Operation: Invoke the MVS router to perform authorization checking using the standard form, for an IMS/VS transaction pointed to by register 5. The user requests only read access. The request is issued on behalf of the IMS/VS subsystem.

```
RACROUTE  REQUEST=FASTAUTH,SUBSYS=SUBIMS,                  X
          WORKA=RACWK,ENTITY=(R5),                         X
          CLASS='TIMS',WKAREA=FRACWK,                      X
          ATTR=READ
  .
  .
  .
SUBIMS    DC  CL8'IMS'
FRACWK    DS  16F
RACWK     DS  CL512
```

RACROUTE (List Form)

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

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

REQUEST=AUTH REQUEST=FASTAUTH REQUEST=DEFINE REQUEST=VERIFY REQUEST=LIST REQUEST=EXTRACT	
,REQSTOR=*reqstor addr*	*reqstor addr:* A-type address. **Default:** zero. **Note:** If REQSTOR= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,SUBSYS=*subsys addr*	*subsys addr:* A-type address. **Note:** If SUBSYS= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,WORKA=*work area addr*	*work area addr:* A-type address
,RELATED=*value*	*value:* Any valid macro keyword specified.
,ENVIR=VERIFY	**Note:** ENVIR can be coded only if REQUEST=VERIFY is coded.
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default:** See parameter description. **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,MSGRTRN=YES ,MSGRTRN=NO	**Default:** NO
,MSGSUPP=YES ,MSGSUPP=NO	**Default:** NO
,MSGSP=*subpool number*	*subpool number*: decimal digit 0-255
,Release=1.6 ,Release=1.7 ,Release=1.8 ,Release=1.8.1	**Default:** 1.6
,MF=L	

Note: In addition to the parameters described above, all parameters valid on the RACDEF, RACLIST, RACINIT, RACXTRT, RACHECK, and FRACHECK macros are permitted on the RACROUTE macro. Depending on the parameter REQUEST=, some of these are required, some optional, and some are invalid.

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

,MF =L
specifies the list form of the RACROUTE macro.

RACROUTE (Execute Form)

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

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

REQUEST=AUTH REQUEST=FASTAUTH REQUEST=DEFINE REQUEST=VERIFY REQUEST=LIST REQUEST=EXTRACT	
,REQSTOR=*reqstor addr*	*reqstor addr:* RX-type address or register (2) - (12). **Default:** zero. **Note:** If REQSTOR= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,SUBSYS=*subsys addr*	*subsys addr:* RX-type address or register (2) - (12). **Note:** If SUBSYS= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,WORKA=*work area addr*	*work area addr:* RX-type address or register (2) - (12).
,RELATED=*value*	*value:* Any valid macro keyword specified.
,ENVIR=VERIFY	**Note:** ENVIR can be coded only if REQUEST=VERIFY is coded.
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default:** See parameter description. **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,MSGRTRN=YES ,MSGRTRN=NO	**Default**: NO
,MSGSUPP=YES ,MSGSUPP=NO	**Default**: NO
,MSGSP=*subpool number*	*subpool number*: decimal digit 0-255
,Release=(*number*, CHECK) ,Release=*number* ,Release=(,CHECK)	*number*1.8.1, 1.8, 1.7, or 1.6 **Default:** 1.6
,MF=(E,*ctrl addr)*	*ctrl addr:* RX-type address or register (1), (2) - (12).

Note: In addition to the parameters described above, all parameters valid on the RACDEF, RACLIST, RACINIT, RACXTRT, RACHECK, and FRACHECK macros are permitted on the RACROUTE macro. Depending on the parameter REQUEST=, some of these are required, some optional, and some are invalid.

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

,MF =(E, *ctrl addr)*
specifies the execute form of the RACROUTE macro where *ctrl addr* is the address of the associated parameter list.

,RELEASE =(*number*,CHECK)
,RELEASE =1.6|1.7|1.8|1.8.1
,RELEASE= (,CHECK)

specifies the RACF release level of the parameter list that this macro will generate.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE = 1.6 and later, see Figure 10.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the FRACHECK macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code of 'X64' will be generated.

RACROUTE — Router Interface (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435.

The RACROUTE macro is used to invoke the System Authorization Facility (SAF) MVS router. If RACF is present, the MVS router directs control to the RACF router. If RACF is active, the RACF router then invokes RACF.

You can use RACROUTE to perform the following functions:

REQUEST=AUDIT,
REQUEST=AUTH,
REQUEST=DEFINE,
REQUEST=DIRAUTH,
REQUEST=EXTRACT,
REQUEST=FASTAUTH,
REQUEST=LIST,
REQUEST=STAT,
REQUEST=TOKENBLD,
REQUEST=TOKENMAP,
REQUEST=TOKENXTR,
REQUEST=VERIFY,
REQUEST=VERIFYX

In coding the RACROUTE macro to perform a particular request, you must also use the necessary parameters from that request type on the RACROUTE macro. For example, if you code RACROUTE to access REQUEST=AUTH, you must code REQUEST=AUTH and any other required parameters and any optional ones you need from the RACROUTE REQUEST=AUTH macro. RACROUTE validates that only the parameters applicable to the RACROUTE REQUEST=AUTH macro have been coded.

Beginning with Release 1.9, when the function verifies the parameter list, the keywords with length fields of 0 are processed as if the keyword were not specified.

Note: Three RACF functions invoked by RACROUTE (REQUEST=AUTH, REQUEST=LIST, and REQUEST=EXTRACT) require that you specify the CLASS parameter, and that the specified CLASS be active. In addition, the class specified on the CLASS parameter in the class descriptor table (CDT) must be active for the RACF router to invoke RACF.

Notes:

1. If a caller is executing in 24-bit addressing mode, all parameters and parameter lists are assumed to reside below 16 megabytes. If a caller, however, is executing in 31-bit addressing mode, all parameters and parameter lists may reside above 16 megabytes (that is, that all parameter addresses are true 31-bit addresses).
2. All parameter lists generated by the RACROUTE macro are in a format that allows compiled code to be moved above 16 megabytes without recompilation.

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

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

ƀ                                One or more blanks must precede RACROUTE.

RACROUTE

ƀ                                One or more blanks must follow RACROUTE.
```

```
REQUEST=AUDIT
REQUEST=AUTH
REQUEST=DEFINE
REQUEST=DIRAUTH
REQUEST=EXTRACT
REQUEST=FASTAUTH
REQUEST=LIST
REQUEST=STAT
REQUEST=TOKENBLD
REQUEST=TOKENMAP
REQUEST=TOKENXTR
REQUEST=VERIFY
REQUEST=VERIFYX

   ,REQSTOR=reqstor addr         reqstor addr: A-type address or register (2) - (12)
                                 Default: REQSTOR=zero
                                 Note: If you specify REQSTOR and RACF is installed, you
                                 must either update the RACF router table to match the
                                 operand or specify DECOUPL=YES.

   ,SUBSYS=subsys addr           subsys addr: A-type address or register (2) - (12)
                                 Note: If you specify SUBSYS and RACF is installed, you must
                                 either update the RACF router table to match the operand or
                                 specify DECOUPL=YES.

,WORKA=work area addr            work area addr: A-type address or register (2) - (12)

   ,RELATED=value                value: Any valid macro keyword specified

   ,MSGRTRN=YES
   ,MSGRTRN=NO                   Default: MSGRTRN=NO

   ,MSGSUPP=YES
   ,MSGSUPP=NO                   Default: MSGSUPP=NO

   ,MSGSP=subpool number         subpool number: Decimal digit 0-255; Default is 0

   ,DECOUPL=YES
   ,DECOUPL=NO                   Default: DECOUPL=NO

,RELEASE=1.9
```

For RACROUTE to work correctly, once you have chosen a REQUEST, you must also specify the parameters that belong to that request. Please see the RACROUTE REQUEST= macros for the necessary parameters.

The parameters are explained as follows:

Notes:

1. This request requires a standard 18 word save area that is pointed to by Register 13.
2. Data areas returned by RACF to the caller will be either above or below 16-megabytes, depending upon the caller's addressing mode and the data area in question.

,REQSTOR = *reqstor addr*
specifies the address of an 8-byte character field containing the name of the piece of code that is making the request. (this address identifies a unique piece of code within a set of code that exists in a subsystem). If this operand is omitted, a string of eight blanks is assumed.

Prior to Release 1.9, if you specified this operand and RACF was installed, you had to update the RACF router table with a matching entry. If you did not update the table, RACF processing was bypassed. With Release 1.9 you do not have to put a matching entry in the router table if you specify the DECOUPL keyword. (See the DECOUPL keyword.)

,SUBSYS = *subsys addr*
specifies the address of an 8-byte character field containing the calling subsystem's name, version, and release level. If this operand is omitted, a string of eight blanks is assumed.

Prior to Release 1.9, if you specified this operand and RACF was installed, you had to update the RACF router table with a matching entry. If you did not update the table, RACF processing was bypassed. With Release 1.9 you do not have to put a matching entry in the router table if you specify the DECOUPL keyword. (See the DECOUPL keyword.)

,WORKA = *work area addr*
specifies the address of a 512-byte work area for use by the router and the RACF front end routine.

,RELATED = *value*
specifies information used to make notes to yourself to document macros by "relating" functions or services to corresponding functions or services. You can use any format and put in any length and type of data you want.

,MSGRTRN = YES
,MSGRTRN = NO
specifies whether you want to use message return processing. You can use this parameter in conjunction with the other RACROUTE MSGxxxx parameters to store and forward messages resulting from this service.

Note: This parameter only applies to REQUEST = AUTH, REQUEST = DEFINE, REQUEST = VERIFY, and REQUEST = VERIFYX.

,MSGSUPP = YES
,MSGSUPP = NO
specifies whether you want to suppress write-to-operator (WTO) messages from within RACF processing. You can use this parameter in conjunction with the other RACROUTE MSGxxxx parameters to store and forward messages resulting from this service.

Note: This parameter applies only to RACROUTE REQUEST = AUTH, RACROUTE REQUEST = DEFINE, RACROUTE REQUEST = VERIFY, and RACROUTE REQUEST = VERIFYX.

,MSGSP = *subpool number*
specifies the number of the subpool into which you want RACF messages returned. You can use ,MSGSP in conjunction with the other RACROUTE MSGxxxx parameters to store and forward RACF messages. If you do not specify a subpool, the default subpool is 0.

Notes:

1. This parameter only applies to REQUEST = AUTH, REQUEST = DEFINE, REQUEST = VERIFY, and REQUEST = VERIFYX.
2. For RACF, only IRR102I, IRR101I, ICH408I, ICH70004I, and ICH70005I messages can be returned.
3. When control returns from RACROUTE, the RACROUTE parameter list field is mapped by SAFPMSAD in the ICHSAFP mapping macro. SAFPMSAD will be non-zero if messages have been returned. This field will contain the address of an area which consists of two full words followed by the message itself in WTO parameter list format. The first word is the length of the area including the two full word header; the second word points to the next message area, if there is one, or contains zero if no more messages areas exist. If is your responsibility to issue the FREEMAIN macro to release these message area(s).

,DECOUPL = YES
,DECOUPL = NO

specifies whether you want to have the REQSTOR and SUBSYS parameters to have corresponding entries in the router table. Prior to 1.9, if you specified REQSTOR or SUBSYS, you had to have corresponding entries for the class in the router table. To use this keyword, you must specify RELEASE = 1.9.

,RELEASE = 1.9

specifies the release number. With Release 1.9, you can specify message parameters to obtain messages from RACROUTE REQUEST = AUTH, RACROUTE REQUEST = DEFINE, RACROUTE REQUEST = VERIFY, and RACROUTE REQUEST = VERIFYX.

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes. These return codes represent return codes from all invocations of the RACROUTE macro; for example, REQUEST = AUTH, REQUEST = VERIFY, etc. For specific information on the success or failure of the invocation in question, see the section of this book that describes that invocation.

When you execute the macro, space for return codes and their respective reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	The requested security function has completed successfully. In addition, if the requested function was 'AUTH', the authorization request was accepted.
04	The requested function has not been processed. In addition, if the request was 'AUTH', the MVS router could neither accept nor fail the request. The following are some possible reasons for a request not being processed. - The MVS router is not active. - The RACF front end routine detected that a null action was requested for the specified request type, resource type, and subsystem ID. - The request/resource/subsystem combination could not be found in the router table. - RACF is not active on the system, and RACFIND=YES was not specified, and there is no RACROUTE installation exit routine (or an exit originated a return code of 4). - RACF is active on the system, but no profile exists for the specified resource.
08	The requested function was processed by RACF, the MVS router, or the router exit (ICHRTX00) and failed. If the requested function was AUTH, the authorization request has been failed. If RACF is inactive for an 'AUTH' request with RACFIND=YES, then the MVS router fails the request. The RACF or router exit return code and reason codes are returned in the first two words of the RACROUTE input parameter list.

Example 1

Operation: Invoke the MVS router to perform authorization checking using the standard form, for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on.

```
RACROUTE  REQUEST=AUTH,WORKA=RACWK,ENTITY=((R7)),        X
          VOLSER=(R8),CLASS='DATASET',ATTR=ALTER,        X
          RACFIND=YES,RELEASE=1.9
  .
  .
RACWK     DS  CL512
```

Example 2

Operation: Invoke the MVS router to perform authorization checking using the standard form, for an IMS/VS transaction pointed to by register 5. The user requests only read access. The request is issued on behalf of the IMS/VS subsystem.

```
RACROUTE  REQUEST=FASTAUTH,SUBSYS=SUBIMS,              X
          WORKA=RACWK,ENTITY=(R5),                     X
          CLASS='TIMS',WKAREA=FRACWK,                  X
          ATTR=READ,RELEASE=1.9
  .
  .
  .
SUBIMS    DC  CL8'IMS'
FRACWK    DS  16F
RACWK     DS  CL512
```

RACROUTE (List Form)

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

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

REQUEST=AUDIT REQUEST=AUTH REQUEST=DEFINE REQUEST=DIRAUTH REQUEST=EXTRACT REQUEST=FASTAUTH REQUEST=LIST REQUEST=STAT REQUEST=TOKENBLD REQUEST=TOKENMAP REQUEST=TOKENXTR REQUEST=VERIFY REQUEST=VERIFYX	
,REQSTOR=*reqstor addr*	*reqstor addr*: A-type address **Default**: REQSTOR=zero **Note:** If you specify REQSTOR and RACF is installed, you must either update the RACF router table to match the operand or specify DECOUPL=YES.
,SUBSYS=*subsys addr*	*subsys addr*: A-type address **Note:** If you specify SUBSYS and RACF is installed, you must either update the RACF router table to match the operand or specify DECOUPL=YES.
,WORKA=*work area addr*	*work area addr*: A-type address
,RELATED=*value*	*value*: Any valid macro keyword specified
,MSGRTRN=YES ,MSGRTRN=NO	**Default**: MSGRTRN=NO
,MSGSUPP=YES ,MSGSUPP=NO	**Default**: MSGSUPP=NO
,MSGSP=*subpool number*	*subpool number*: Decimal digit 0-255
,DECOUPL=YES ,DECOUPL=NO	**Default**: DECOUPL=NO
,RELEASE=1.9	
,MF=L	

The REQUEST= parameters are explained under their respective invocations. The RACROUTE parameters are explained under the standard form of the RACROUTE macro with the following exception:

,MF=L
specifies the list form of the RACROUTE macro.

RACROUTE (Execute Form)

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

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

REQUEST=AUDIT REQUEST=AUTH REQUEST=DEFINE REQUEST=DIRAUTH REQUEST=EXTRACT REQUEST=FASTAUTH REQUEST=LIST REQUEST=STAT REQUEST=TOKENBLD REQUEST=TOKENMAP REQUEST=TOKENXTR REQUEST=VERIFY REQUEST=VERIFYX	
,REQSTOR=*reqstor addr*	*reqstor addr*: RX-type address or register (2) - (12) **Default:** REQSTORE=zero **Note:** If you specify REQSTOR and RACF is installed, you must either update the RACF router table to match the operand or specify DECOUPL=YES.
,SUBSYS=*subsys addr*	*subsys addr*: RX-type address or register (2) - (12) **Note:** If you specify SUBSYS and RACF is installed, you must either update the RACF router table to match the operand or specify DECOUPL=YES.
,WORKA=*work area addr*	*work area addr*: RX-type address or register (2) - (12)
,RELATED=*value*	*value*: Any valid macro keyword specified
,MSGRTRN=YES ,MSGRTRN=NO	**Default:** MSGRTRN=NO
,MSGSUPP=YES ,MSGSUPP=NO	**Default:** MSGSUPP=NO
,MSGSP=*subpool number*	*subpool number*: Decimal digit 0-255
,DECOUPL=YES ,DECOUPL=NO	**Default:** DECOUPL=NO
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9
,MF=(**E**,*ctrl addr*)	*ctrl addr*: RX-type address or register (1), (2) - (12)

The REQUEST= parameters are explained under their respective invocations. The RACROUTE parameters are explained under the standard form of the RACROUTE macro with the following exception:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACROUTE macro where *ctrl addr* is the address of the associated parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level 1.9 of the parameter list to be generated by the macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=FASTAUTH macro can be done by specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE (Modify Form)

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

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

REQUEST=AUDIT REQUEST=AUTH REQUEST=DEFINE REQUEST=DIRAUTH REQUEST=EXTRACT REQUEST=FASTAUTH REQUEST=LIST REQUEST=STAT REQUEST=TOKENBLD REQUEST=TOKENMAP REQUEST=TOKENXTR REQUEST=VERIFY REQUEST=VERIFYX	
,REQSTOR=*reqstor addr*	*reqstor addr*: RX-type address or register (2) - (12) **Default**: REQSTOR=zero. **Note:** If you specify REQSTOR and RACF is installed, you must either update the RACF router table to match the operand or specify DECOUPL=YES.
,SUBSYS=*subsys addr*	*subsys addr*: RX-type address or register (2) - (12) **Note:** If you specify SUBSYS and RACF is installed, you must either update the RACF router table to match the operand or specify DECOUPL=YES.
,WORKA=*work area addr*	*work area addr*: RX-type address or register (2) - (12)
,RELATED=*value*	*value*: Any valid macro keyword specified
,MSGRTRN=YES ,MSGRTRN=NO	**Default**: MSGRTRN=NO
,MSGSUPP=YES ,MSGSUPP=NO	**Default**: MSGSUPP=NO
,MSGSP=*subpool number*	*subpool number*: Decimal digit 0-255
,DECOUPL=YES ,DECOUPL=NO	**Default**: DECOUPL=NO
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9
,MF=(**M**,*ctrl addr*)	*ctrl addr*: RX-type address or register (1), (2) - (12)

The REQUEST= parameters are explained under their respective invocations. The RACROUTE parameters are explained under the standard form of the RACROUTE macro with the following exception:

,MF=(M,*ctrl addr*)
specifies the modify form of the RACROUTE macro where *ctrl addr* is the address of the associated parameter list. The macro updates the parameter list, but does not execute the macro.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level 1.9 of the parameter list to be generated by the macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=FASTAUTH macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code of X'64' will be generated.

RACROUTE REQUEST = AUDIT — General Purpose Security Audit Request

The RACROUTE REQUEST = AUDIT macro is a general purpose security audit request which records events in SMF type 80 records and issues messages to the network security administrator.

RACF searches profiles in the following order: the profiles chained off the supplied ACEE, the TCB (task control block) ACEE, the ASXB ACEE, and finally, any profiles that have been listed via the SETROPTS RACLIST command.

To use this service, you must specify RELEASE = 1.9.

REQUEST = AUDIT is an SRB compatible service. If the caller is not in SRB mode, then the caller must be APF-authorized. If the caller is not in supervisor state, RACROUTE issues a modeset to switch the caller to supervisor state.

The standard form of the RACROUTE REQUEST = AUDIT macro is written as follows:

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

REQUEST = AUDIT	
,ENTITYX = *extended resource name addr*	*extended resource name addr*: A-type address or register (2) - (12)
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: A-type address or register (2) - (12)
,EVENT = *'event name'* ,EVENT = *event name addr*	*event name*: 1-8 character name *event name addr*: A-type address or register (2) - (12)
,EVQUAL = *number* ,EVQUAL = *reg*	*number*: 0-99 *reg*: Register (2) - (12)
,ACEE = *acee addr*	*acee addr*: A-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9

The parameters are explained as follows:

,ENTITYX = *extended resource name addr*
specifies the address of a structure that consists of two 2-byte length fields, followed by the entity name.

- The first 2-byte field specifies a buffer length which can be from 0 to 255 bytes. This length field only refers to the length of the buffer that contains the entity name; it does not include the length of either length field.
- The second 2-byte field specifies the actual length of the entity name. This length field only includes the length of the actual name without any trailing blanks; it does not include the length of either length field.

These two length fields can be used in several different ways:

- If the length of the entity name is known, you can specify 0 in the first field and specify the length of the entity name in the second field. Note that when you specify the second field, each byte counts; this means the entity name specified must match exactly the entity name on the RACF data base.
- If you choose to use a buffer area in which to place the entity name, you can specify the first field to designate the length of the buffer. In regard to the second field, you can do one of two things:
  - If you know the length of the entity name, you would specify the length in the second field. (Note that the length of the first field can be from 0 to 255, but must be equal to or more than the length of the second field.)
  - If you do not know the length of the entity name, you would specify 0 in the second field, in which case RACF would be responsible for counting the number of characters in the entity name.

Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

,CLASS= *'class name'*
,CLASS= *class name addr*
specifies that you want RACF to perform authorization checking for a resource in this class. You can specify the class name or the class name address. If you specify a class name address, the address must point to an 8-byte field that contains the class name. The class name must be left justified and padded to the right with blanks, if necessary.

,RELEASE= *number*
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=AUDIT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,EVENT= *'event name'*
,EVENT= *event name addr*
specifies the name of the event that you want RACF to log. You can specify the event name or the event name address. If you specify the event name address, it must point to an 8-byte field that contains the event name. The event name must be left justified and padded to the right with blanks, if necessary. The only event that you can log with Release 1.9 is APPCSEC. RACF allows this keyword only when you specify REQUEST=AUDIT and RELEASE=1.9.

,EVQUAL= *number*
,EVQUAL= *reg*
specifies the event code qualifier for the event that you want logged. If you specify a register rather than a number, you must enter the event code qualifier in the low-order half-word of the register or the field the address in the register points to. The qualifier can be from 0-99. See the *RACF Security Administrator's Guide* for a description of the event code qualifiers for an event. RACF allows this keyword only when you specify REQUEST=AUDIT and RELEASE=1.9.

,ACEE= *acee addr*
specifies the address of an ACEE passed on a REQUEST=AUDIT. RACF searches local profiles chained off the ACEE that have been placed there via the RACLIST macro. For auditing purposes, RACF checks for LOGOPTIONS for the class first, then searches for a profile to match an entity name. If RACF does not find a profile, then it does not perform any auditing. An ACEE is required for REQUEST=AUDIT.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	The requested security function has completed successfully.
04	The class is not active.
08	The class was not specified.
0C	Indicates an internal error from RACXTRT.

Reason Code	Meaning
xxyy	xx - Return code from RACXTRT; yy - Reason code from RACXTRT

10 Parameter list error as described by the following hex reason codes

Reason Code	Meaning
0	Invalid event
4	Invalid event code qualifier
8	Invalid parameter list version
C	Invalid parameter list length
10	Invalid entity
14	LOGOPTIONS not set and no profile found. No auditing done.

Example 1

Operation: Invoke the RACROUTE REQUEST = AUDIT macro to search for a profile in the APPCLU class to match the entity specified in LULUPAIR. The profiles to be searched have been placed in storage using the RACROUTE REQUEST = RACLIST macro. Be aware that if SETROPTS LOGOPTIONS, other than MACRO, have been specified for the APPCLU class, then those auditing options will be the ones that RACF uses. If the SUBSYS and REQSTOR parameters are not represented in the router table, then you must specify DECOUPL = YES. Set the auditing options so that an SMF 80 event APPCLU event code qualifier 04 (partner session keys were not equal) is logged. A message will be sent to the security console, and message ICH70005I will be sent to the user.

Note: The message cannot be received by anyone other than the person to whom it was directed.

```
RACROUTE  REQUEST=AUDIT,CLASS='APPCLU',ENTITY=LULUPAIR, X
          ACEE=VTAMACEE,EVENT='APPCLU',EVQUAL=CODE04,   X
          WORKA=WORKADDR, REQSTOR=TPUTRECV,             X
          DECOUPL=YES,RTOKEN(8),                        X
          LOG=ASIS,RELEASE=1.9
  .
  .
RACWK     DS  CL512
```

Example 2

Operation: Invoke the RACROUTE REQUEST = AUDIT macro on behalf of the VTAM resource manager to perform mandatory access checking (MAC) in the "receiving" user's address space to ensure that the receiver's SECLABEL dominates that of the sender. Specify that RACF should not audit the event if RACF fails the mandatory access check. The SUBSYS and REQSTOR parameters are not represented in the router table; therefore, you must specify DECOUPL = YES.

Note: The message cannot be received by anyone other than the person to whom it was directed.

```
RACROUTE  REQUEST=AUDIT,WORKA=RACWK,SUBSYS=VTAM,        X
          REQSTOR=TPUTRECV,DECOUPL=YES,RTOKEN(8),       X
          LOG=ASIS,RELEASE=1.9
  .
  .
RACWK     DS  CL512
```

RACROUTE REQUEST = AUDIT (List Form)

The list form of the RACROUTE REQUEST = AUDIT macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = AUDIT	
,ENTITYX = *extended resource name addr*	*extended resource name addr*: A-type address
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: A-type address
,EVENT = *'event name'* ,EVENT = *event name addr*	*event name*: 1-8 character name *event name addr*: A-type address
,EVQUAL = *number*	*number*: 0-99
,ACEE = *acee addr*	*acee addr*: A-type address
,RELEASE = *number*	*number*: 1.9
,MF = **L**	

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

,MF = L
specifies the list form of the RACROUTE macro.

RACROUTE REQUEST = AUDIT (Execute Form)

The execute form of the RACROUTE REQUEST = AUDIT macro is written as follows:

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

REQUEST = AUDIT	
,ENTITYX = *extended resource name addr*	*extended resource name addr*: RX-type address or register (2) - (12)
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: RX-type address or register (2) - (12)
,EVENT = *'event name'* ,EVENT = *event name addr*	*event name*: 1-8 character name *event name addr*: RX-type address or register (2) - (12)
,EVQUAL = *number* ,EVQUAL = *reg*	*number*: 0-99 *reg*: Register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,MF = **E**	

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

,MF = E
specifies the execute form of the RACROUTE macro.

RACROUTE REQUEST = AUDIT (Modify Form)

The modify form of the RACROUTE REQUEST = AUDIT macro is written as follows:

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

REQUEST = AUDIT	
,ENTITYX = *extended resource name addr*	*extended resource name addr*: RX-type address or register (2) - (12)
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: RX-type address or register (2) - (12)
,EVENT = *'event name'* ,EVENT = *event name addr*	*event name*: 1-8 character name *event name addr*: RX-type address or register (2) - (12)
,EVQUAL = *number* ,EVQUAL = *reg*	*number*: 0-99 *reg*: Register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,MF = **M**	

The parameters are explained under the standard form of the RACROUTE REQUEST = AUDIT macro with the following exception:

,MF = M
specifies the modify form of the RACROUTE REQUEST = AUDIT macro.

RACROUTE REQUEST = AUTH — Check RACF Authorization (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see the following:

- **"RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435**
- **"RACHECK — Check RACF Authorization (for RACF Release 1.8.1 or earlier)" on page 403.** (IBM recommends that you use RACROUTE with the REQUEST = AUTH parameter rather than RACHECK.)

The RACROUTE REQUEST = AUTH macro is used to provide authorization checking when a user requests access to a RACF-protected resource.

The standard form of the RACROUTE REQUEST = AUTH macro is written as follows:

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

REQUEST = AUTH	
,PROFILE = *profile addr*	*profile addr*: A-type address or register (2) - (12)
,ENTITY = (*resource name addr*)	*resource name addr*: A-type address or register (2) - (12)
,ENTITY = (*resource name addr*,CSA)	
,ENTITY = (*resource name addr*,PRIVATE)	
,ENTITY = (*resource name addr*,NONE)	
,ENTITYX = *extended resource name addr*	*extended resource name addr*: A-type address or register (2) - (12)
,ENTITYX = (*extended resource name addr*,CSA)	
,ENTITYX = (*extended resource name addr*,PRIVATE)	
,ENTITYX = (*resource name addr*,NONE)	
,VOLSER = *vol addr*	*vol addr*: A-type address or register (2) - (12) **Note:** VOLSER is required only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used and only when ENTITY is also coded.
,CLASS = *'class name'*	*class name*: 1-8 character name
,CLASS = *class name addr*	*class name addr*: A-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9, 1 8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ATTR = READ	**Default**: ATTR = READ
,ATTR = UPDATE	
,ATTR = CONTROL	
,ATTR = ALTER	
,ATTR = *reg*	*reg*: register (2) - (12)
,DSTYPE = N	**Default**: DSTYPE = N
,DSTYPE = V	
,DSTYPE = M	
,DSTYPE = T	
,INSTLN = *parm list addr*	*parm list addr*: A-type address or register (2) - (12)

,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default**: LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr*: A-type address or register (2) - (12)
,APPL='*applname*' ,APPL=*applname addr*	*applname*: 1-8 character name *applname addr*: A-type address or register (2) - (12)
,ACEE=*acee addr*	*acee addr*: A-type address or register (2) - (12)
,OWNER=*owner id addr*	*owner id addr*: A-type address or register (2) - (12)
,ACCLVL=*access level addr* ,ACCLVL=(*access level addr*,*parm list addr*)	*access level addr*: A-type address or register (2) - (12) *parm list addr*: A-type address or register (2) - (12)
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE ,STATUS=EVERDOM	**Default**: STATUS=NONE
,USERID='*userid*' ,USERID=*userid addr*	*userid*: 1-8 character user ID *userid addr*: A-type address or register (2) - (12)
,GROUPID='*groupid*' ,GROUPID=*groupid addr*	*groupid*: 1-8 character group ID *groupid addr*: A-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: A-type address or register (2) - (12)
,UTOKEN=*token addr*	*token addr*: A-type address or register (2) - (12)
,RTOKEN=*rtoken addr*	*rtoken addr*: A-type address or register (2) - (12)
,RECVR=*recvr addr*	*recvr addr*: A-type address or register (2) - (12)

The parameters are explained as follows:

,PROFILE=*profile addr*
,ENTITY=(*resource name addr***)**
,ENTITY=(*resource name addr***,CSA)**
,ENTITY=(*resource name addr***,PRIVATE)**
,ENTITY=(*resource name addr***,NONE)**
,ENTITYX=*extended resource name addr*
,ENTITYX=(*extended resource name addr***,CSA)**
,ENTITYX=(*extended resource name addr***,PRIVATE)**
,ENTITYX=(*extended resource name addr***,NONE)**
specifies the resource address:

- PROFILE=*profile addr*
  specifies that RACF authorization checking is to be performed for the resource whose profile is pointed to by the specified address. This profile must be supplied

by a previously coded ENTITY=(xxx,CSA,PRIVATE). A profile supplied by RACLIST is not acceptable.

- ENTITY=(*resource name addr*)
  specifies that RACF authorization checking is to be performed for the resource whose name is pointed to by the specified address. The resource name is a 44-byte DASD data set name for CLASS=DATASET or a 6-byte volume serial number for CLASS=DASDVOL or CLASS=TAPEVOL. The length of all other resource names is determined from the class descriptor tables. The name must be left-justified in the field and padded with blanks.
- ENTITY=(*resource name addr*,**CSA**)
  specifies that RACF authorization checking is to be performed for the indicated resource and that a copy of the profile is to be maintained in central storage. The storage acquired for the profile is obtained from the common storage area (CSA), and is fetch-protected, key 0 storage.

  If CSA is specified and the return code produced by the RACROUTE REQUEST=AUTH macro instruction is 00 or 08, the address of the profile is returned in register 1.

  By establishing and maintaining a resource profile, the resource manager can reduce the I/O required to perform RACF authorization checks on highly-accessed resources.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.
- ENTITY=(*resource name addr*,**PRIVATE**)
  PRIVATE specifies the same as CSA except that RACROUTE returns the profile in the user private area rather than in common storage, and the name field contains the name of the returned profile instead of the name of the resource that was specified on the ENTITY keyword. The issuer of RACROUTE REQUEST=AUTH must free this storage when the profile is no longer needed. (The profile subpool number and length are returned as well as the profile data.)
- ENTITY=(*resource name addr*,**NONE**)
  specifies the same as ENTITY=resource name address. However, no profile is returned.
- ENTITYX=*extended resource address*
  specifies the address of a structure that consists of two 2-byte length fields, followed by the entity name.
  - The first 2-byte field specifies a buffer length which can be from 0 to 255 bytes. This length field refers to the length of the buffer that contains the entity name; it does not include the length of either length field.
  - The second 2-byte field specifies the actual length of the entity name. This length field includes the length of the actual name without any trailing blanks; it does not include the length of either length field.

  These two length fields can be used in several different ways:
  - If the length of the entity name is known, you can specify 0 in the first field and specify the length of the entity name in the second field. Note that when you specify the second field, each byte counts; this means the entity name specified must match exactly the entity name on the RACF data base.

- If you choose to use a buffer area in which to place the entity name, you can specify the first field to designate the length of the buffer. In regard to the second field, you can do one of two things:
  - If you know the length of the entity name, you would specify the length in the second field. (Note that the length of the first field can be from 0 to 255, but must be equal to or more than the length of the second field.)
  - If you do not know the length of the entity name, you would specify 0 in the second field, in which case RACF would be responsible for counting the number of characters in the entity name.

Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

To use this keyword, you must also specify RELEASE=1.9 or a later release number.

- ENTITYX=(*extended resource name addr*,**CSA**,)
  specifies that RACF authorization checking is to be performed for the indicated resource, and that a copy of the profile is to be maintained in main storage. The storage acquired for the profile is obtained from the common storage area (CSA), and is fetch-protected, key 0 storage.

  If CSA is specified and the return code produced by the RACROUTE REQUEST=AUTH macro instruction is 00 or 08, the address of the profile is returned in register 1.

  To use this keyword, you must also specify RELEASE=1.9 or a later release number.

- ENTITYX=(*extended resource name addr*,**PRIVATE**)
  PRIVATE specifies the same as CSA, except that RACROUTE returns the profile in the user private area rather than in common storage, and the name field contains the name of the returned profile instead of the name of the resource that was specified on the ENTITY keyword.

  The issuer of RACROUTE REQUEST=AUTH must free this storage when the profile is no longer needed. (The profile subpool number and length are part of the profile data returned.)

  To use this keyword, you must also specify RELEASE=1.9 or a later release number.

- ENTITYX=(*extended resource name addr*,**NONE**)
  specifies the same as ENTITYX=resource name address. However, no profile is returned.

  To use this keyword, you must also specify RELEASE=1.9 or a later release number.

Recommendation

IBM recommends that you use ENTITYX rather than ENTITY for two reasons:

- With ENTITYX, if you know the length of the entity name, ENTITYX allows you to pass that information to RACF. Doing so can result in slightly faster processing.
- With ENTITY, the entity name you pass to RACF must be in a buffer, the size of which is determined by the length in the CDT. If the MAXLNTH of a class increases in the future, you would have to modify your program to use a larger buffer. By using ENTITYX, you avoid this possible problem because you have removed the CDT dependency from your program.

,VOLSER = *vol addr*
specifies the volume serial number, as follows:

- For non-VSAM DASD data sets and tape data sets, this is the volume serial number of the volume on which the data set resides.
- For VSAM DASD data sets, this is the volume serial number of the catalog controlling the data set.

The volume serial number is optional if DSTYPE = M is specified; it is ignored if the profile name is generic.

The field pointed to by the specified address contains the volume serial number padded to the right with blanks, if necessary, to make six characters. VOLSER = is only valid and must be supplied with CLASS = DATASET, (unless DSTYPE = M is specified) and if ENTITY is also coded.

,CLASS = *'class name'*
,CLASS = *class name addr*
specifies that RACF authorization checking is to be performed for a resource of the specified class. The address must point to a 1-byte field indicating the length of the class name, followed by the class name.

,RELEASE = *number*
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST = AUTH macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,ATTR = READ
,ATTR = UPDATE
,ATTR = CONTROL
,ATTR = ALTER
,ATTR = *reg*
specifies the access authority of the user or group permitted access to the resource for which RACF authorization checking is to be performed:

READ - RACF user or group can open the resource only to read.

UPDATE - RACF user or group can open the resource to write or read.

CONTROL - For VSAM data sets, RACF user or group has authority equivalent to the VSAM control password. For non-VSAM data sets and other resources, RACF user or group has UPDATE authority.

ALTER - RACF user or group has total control over the resource.

If a register is specified, the register must contain one of the following codes in the low-order byte of the register:

X'02' - READ
X'04' - UPDATE
X'08' - CONTROL
X'80' - ALTER

,DSTYPE = N
,DSTYPE = V
,DSTYPE = M
,DSTYPE = T
specifies the type of data set associated with the request:

- N for non-VSAM
- V for VSAM
- M for model profile
- T for tape

If DSTYPE=T is specified and tape data set protection is not active, the processing will be the same as for RACROUTE REQUEST=AUTH CLASS=TAPEVOL.

DSTYPE should only be specified for CLASS=DATASET.

,INSTLN=*parm list addr*
specifies the address of an area that is to contain parameter information meaningful to the RACHECK installation exit routine. This information is passed to the installation exit routine when it is given control by RACHECK.

The INSTLN parameter can be used by an application program acting as a resource manager that needs to pass information to the RACHECK installation exit routine.

,LOG=ASIS
,LOG=NOFAIL
,LOG=NONE
,LOG=NOSTAT
specifies the types of access attempts to be recorded on the SMF data set:

ASIS - RACF records the event in the manner specified in the profile that protects the resource.

NOFAIL - If the authorization check fails, the attempt is not recorded. If the authorization check succeeds, the attempt is recorded as in ASIS.

NONE - The attempt is not to be recorded.

NOSTAT - The attempt is not to be recorded and no resource statistics are to be updated.

,OLDVOL=*old vol addr*
specifies a volume serial:

- For CLASS=DATASET, within the same multivolume data set specified by VOLSER=.
- For CLASS=TAPEVOL, within the same tape volume specified by ENTITY=.

RACF authorization checking will verify that the OLDVOL specified is part of the same multivolume data set or tape volume set.

The specified address points to the field that contains the volume serial number padded to the right with blanks, if necessary, to make six characters.

,APPL=*'applname'*
,APPL=*applname addr*
specifies the name of the application requesting authorization checking. The *applname* is not used for the authorization checking process but is made available to the installation exit routine(s) called by the RACHECK routine. If the address is specified, the address must point to an 8-byte field containing the application name left justified and padded with blanks.

,ACEE=*acee addr*
specifies the address of the accessor environment element (ACEE) to be used during RACHECK processing. If no ACEE is specified, RACF uses the TASK ACEE pointer (TCBSENV) in the extended TCB. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE for the address space. The main ACEE is pointed to by the ASXBSENV field of the address space extension block.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,OWNER = *owner ID addr*
specifies a profile owner id that is compared with the profile owner id in the owner field of the RACF profile. If the owner names match, the access authority allowed for that userid is ALTER. The address must point to an 8-byte field containing the owner name, left-justified and padded with blanks.

If OWNER is specified, any WARNING and OPERATIONS attribute processing is bypassed.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,ACCLVL = *access level addr*
,ACCLVL = (*access level addr,parm list addr*)
specifies the tape label access level information for the MVS tape label functions. The access level pointed to by the specified address is a 1-byte length field, containing the value (0-8) of the length of the following data, followed by an eight-character string that will be passed to the RACHECK installation exit routines. The optional parameter list pointed to by the specified address contains additional information to be passed to the RACHECK installation exit routines. RACF does not inspect or modify this information.

,RACFIND = YES
,RACFIND = NO
indicates whether or not the resource is protected by a discrete profile. The RACF processing and the possible return codes are given in Figure 18.

Note: In all cases, a return code of X'0C' is also possible if the OLDVOL specified was not part of the multivolume data set defined by VOLSER, or it was not part of the same tape volume defined by ENTITY.

Figure 18. Types of Profile Checking Performed by RACROUTE REQUEST = AUTH

Operand	Generic Profile Checking Inactive	Generic Profile Checking Active
RACFIND = YES	Look for discrete profile; if found, exit with return code 00 or 08. If no discrete profile is found, exit with return code 08.	Look for discrete profile; if found, exit with return code 00 or 08. Look for generic profile; if found, exit with return code 00 or 08. Exit with return code 08 if neither a discrete nor a generic profile is found.
RACFIND = NO	No checking. Exit with return code 04.	Look for generic profile; if found, exit with return code 00 or 08. If not found, exit with return code 04.
RACFIND not specified	Look for discrete profile; if found, exit with return code 00 or 08. If no discrete profile is found, exit with return code 04.	Look for discrete profile; if found, exit with return code 00 or 08. Look for generic profile; if found, exit with return code 00 or 08. Exit with return code 04 if neither a discrete nor a generic profile is found.

,GENERIC = YES
,GENERIC = ASIS
specifies whether the resource name is to be treated as a generic profile name. If GENERIC is specified with CLASS = DEFINE, NEWNAME, then GENERIC applies to both the old and new names. GENERIC is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*).

This keyword is designed primarily for use by RACF commands.

- If GENERIC=YES is specified, the resource name is considered a generic profile name, even if it does not contain either of the generic characters: an asterisk (*) or a percent sign (%).
- If GENERIC=ASIS is specified, the resource name is considered a generic only if it contains either of the generic characters: an asterisk (*) or a percent sign (%).

,FILESEQ=*number*
,FILESEQ=*reg*
specifies the file sequence number of a tape data set on a tape volume or within a tape volume set. The value must be in the range 1 - 9999. If a register is specified, it must contain the file sequence number in the low-order half-word. If CLASS=DATASET and DSTYPE=T are not specified, FILESEQ is ignored.

,TAPELBL=STD
,TAPELBL=BLP
,TAPELBL=NL
specifies the type of tape label processing to be done:

- STD - IBM or ANSI standard labels.
- BLP - bypass label processing.
- NL - non-labeled tapes.

For TAPELBL=BLP, the user must have the requested authority to the profile ICHBLP in the general resource class FACILITY. For TAPELBL=NL or BLP, the user will not be allowed to protect volumes with volume serial numbers in the format "Lnnnnn."

This parameter is primarily intended for use by data management routines to indicate the label type from the LABEL keyword on the JCL statement.

This parameter is valid only for CLASS=DATASET and DSTYPE=T, or CLASS=TAPEVOL.

,STATUS=NONE
,STATUS=ERASE
,STATUS=EVERDOM
specifies whether or not RACROUTE REQUEST=AUTH is to return the erase status of the given data set. This parameter is valid only for CLASS=DATASET and a DSTYPE value other than T.
specifies that mandatory (security label) access checking includes a check to see if the user has a SECLABEL, other than the SECLABEL of this job or logon session, which could *ever* dominate that of the current object. There are no restrictions on the CLASS parameter. Be aware that choosing this option increases processing time. The default is that mandatory access checking occurs only with the SECLABEL of the current job or logon session.

,USERID=*'userid'*
,USERID=*userid address*
specifies the userid that RACF uses to perform third-party RACROUTE REQUEST=AUTH. If USERID is specified when the caller invokes RACROUTE REQUEST=AUTH, RACF verifies that user's authority to the given entity; RACF disregards the user ID associated with the ACEE of the caller. This is an eight-character field that is left-justified and padded to the right with blanks. For third party RACROUTE REQUEST=AUTH, RACF performs the following steps:

1. Checks to see if the USERID keyword is *NONE* and GROUPID is not specified. If so, then RACF creates a default user (null) ACEE which it uses to perform the authorization checking.
2. If not, checks to see if an additional (third party) ACEE already exists, chained off the current caller's ACEE or the ACEE specified in the ACEE= keyword.
3. If so, checks to see if the userid in that ACEE matches the one specified on the USERID keyword. If so, RACROUTE REQUEST=AUTH uses the existing ACEE and avoids RACROUTE REQUEST=VERIFY processing.

4. If USERID is specified and RACROUTE REQUEST=VERIFY does not find an additional (third party) ACEE, or the user ID in the ACEE does not match the user ID specified on the USERID keyword, then RACROUTE REQUEST=AUTH creates a third party ACEE based on the USERID keyword.
5. If the GROUPID keyword is specified in addition to the USERID keyword, and a third party ACEE already exists, then the group ID of the existing third party ACEE must also match the group ID specified on the GROUPID keyword. If the group ID keywords do not match, RACROUTE REQUEST=AUTH creates a third party ACEE based on the USERID keyword.

Note: If the calling program does not specify the GROUPID keyword, the internal RACROUTE REQUEST=VERIFY function will use the default group associated with the specified user ID.

Only programs that are APF-authorized, system key 0-7, or in supervisor state, can use the USERID and GROUPID keywords.

,GROUPID= *'groupid'*
,GROUPID= *groupid address*
specifies the group ID that RACF uses to perform third party authorization checking. This is an eight-character field, left-justified, and padded to the right with blanks.

If the calling program wants a third party authorization check performed on the GROUPID rather than the USERID, then the USERID keyord must be specified as *NONE*. Thus, when the caller invokes third party authorization checking, RACF verifies the authority of the group ID to the requested resource; RACF disregards the group ID associated with the ACEE of the caller. For third party authorization checking, RACF performs the following steps:

- Checks to see if an additional (third party) ACEE already exists, chained off the caller's ACEE, or the ACEE specified in the ACEE= keyword.
- If so, checks to see if the group ID matches that specified on the GROUPID keyword. If so, RACROUTE REQUEST=AUTH uses that ACEE and avoids RACROUTE REQUEST=VERIFY processing.
- If GROUPID is specified and RACROUTE REQUEST=AUTH does not find an additional (third party) ACEE, or the group ID in the ACEE does not match the group ID specified on the GROUPID keyword, then RACROUTE REQUEST=AUTH creates a third party ACEE based on the GROUPID keyword.

Only programs that are APF-authorized, system key 0-7, or in supervisor state, can use the USERID and GROUPID keywords.

,LOGSTR= *logstr addr*
specifies a variable length data string consisting of a 1-byte binary length field followed by character data that is to be included in the RACF SMF PROCESS records. The character data can be 0-255 bytes long. The RACF Report Writer will include LOGSTR data on the PROCESS reports.

,UTOKEN= *token addr*
specifies the address of the UTOKEN of the user for which RACF will perform a RACROUTE REQUEST=AUTH. The first byte contains the length of the UTOKEN, and the second byte contains the version number.

,RTOKEN= *rtoken addr*
specifies the address of the RTOKEN of a unit of work. The first byte contains the length of the RTOKEN, followed by the UTOKEN of the creator of the resource. See UTOKEN explanation. For a mapping of the UTOKEN see ICHRUTKN in the Data Areas Chapter of the *SPL: RACF*.

,RECVR= *recvr addr*
specifies the address of the userid of the user who has the authority to access the resource if a resource profile does not exist to protect it. The field is 8 bytes, left justified and padded to the right with blanks.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	The user is authorized by RACF to obtain use of a RACF-protected resource. Register 0 contains one of the following reason codes: 00 Indicates a normal completion. 04 Indicates STATUS=ERASE was specified and the data set is to be erased when scratched. Or the warning status of the resource was requested by the RACROUTE REQUEST=AUTH issuer setting bit '10' at offset 12 decimal in the RACROUTE REQUEST=AUTH parameter list and the resource is in warning mode. 10 When CLASS=TAPEVOL, indicates the tapevol profile contains a TVTOC. 20 When CLASS=TAPEVOL, indicates that the tapevol profile can contain a TVTOC, but currently does not. (Scratch pool volume) 24 When CLASS=TAPEVOL, indicates that the tapevol profile does not contain a TVTOC.
04	The specified resource is not protected by RACF. Register 0 contains the following reason code: 00 Indicates a normal completion.
08	The user is **not** authorized by RACF to obtain use of the specified RACF-protected resource. Register 0 contains the following reason code: 00 Indicates a normal completion. 04 Indicates STATUS=ERASE was specified and the data set is to be erased when scratched. 08 Indicates DSTYPE=T or CLASS=TAPEVOL was specified and the user is not authorized to use the specified volume. 0C Indicates the user is not authorized to use the data set. 10 Indicates DSTYPE=T or CLASS=TAPEVOL was specified and the user is not authorized to specify LABEL=(,BLP). 14 User is not authorized to open a non-cataloged data set. 18 User is not authorized to issue RACROUTE REQUEST=AUTH when system is in tranquil state. 1C User with EXECUTE authority to the data set profile specified ATTR=READ, and RACF failed the access attempt. 20 User's SECLABEL does not dominate that of the resource. 24 User's SECLABEL can never dominate that of the resource. 28 Resource must have a SECLABEL, and does not have one.
0C	The OLDVOL specified was not part of the multivolume data set defined by VOLSER, or it was not part of the same tape volume defined by ENTITY.
10	RACROUTE REQUEST=VERIFY issued by third party RACROUTE REQUEST=AUTH failed. Register 0 contains the RACROUTE REQUEST=VERIFY return code.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=AUTH macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

CDT Default Return Codes and Reason Codes

Normally, if a resource profile is not found, the function returns a return code of 4. However, with Release 1.9, if a resource profile is not found, **but** a default return code keyword is specified in the CDT, then the function returns that specified return code. In some cases, the return codes may indicate that the RACLREQ keyword was specified in the CDT (meaning that a SETROPTS RACLIST must be issued for the class); however, it was not done.

To identify them as default return codes, each default return code will be accompanied by a reason code of X'200'.

Hexadecimal Code	Meaning
00	A default return code of 0 was specified in the CDT. This indicates that the user is authorized by RACF to obtain use of a **non-protected** resource. Register 0 contains the following reason code: 200 Indicates a normal completion
04	A default return code of 4 was specified in the CDT. This means the specified resource is not protected by RACF. Register 0 contains the following reason code: 200 Indicates a normal completion
08	A default return code of 8 was specified in the CDT. This means the user is **not** authorized by RACF to access this **non-protected** resource. Register 0 contains the following reason code: 200 Indicates a normal completion

Example 1

Operation: Perform RACF authorization checking using the standard form, for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on. Logging and statistics updates are **not** to be done.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),VOLSER=(R8),      X
         CLASS='DATASET',                             X
         ATTR=ALTER,RACFIND=YES,LOG=NOSTAT,
         RELEASE=1.9
```

Example 2

Operation: Perform RACF authorization checking using the standard form, for a non-VSAM data set controlled by the catalog pointed to by register 8. Register 7 points to the data set name, and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),VOLSER=(R8),      X
         CLASS='DATASET',                             X
         ATTR=ALTER,RACFIND=YES,RELEASE=1.9
```

Example 3

Operation: Perform RACF authorization checking using the standard form, for a VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name, and the RACF user is requesting the data set for read only. Register 4 points to an area containing additional parameter information.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),VOLSER=(R8),      X
         CLASS='DATASET',                             X
         DSTYPE=V,INSTLN=(R4),RELEASE=1.9
```

Example 4

Operation: Using the standard form, perform RACF authorization checking for a tape volume for read access only. The tape volume is pointed to by register 8 and the volume's access level is in register 5.

```
RACROUTE REQUEST=AUTH,ENTITY=((R8)),CLASS='TAPEVOL',  X
         ATTR=READ,                                   X
         ACCLVL=((R5)),RELEASE=1.9
```

Example 5

Operation: Using the standard form, perform third party RACF authorization checking for a data set for READ access only for a user. Register 7 points to the data set name, and A is an 8-byte declared field padded by zeros.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),                  X
         CLASS='DATASET',ATTR=READ,RELEASE=1.9,       X
         USERID=A
```

Example 6

Operation: Using the standard form, perform third party RACF authorization checking for a data set for READ access only for a group. Register 7 points to the data set name.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),                  X
         CLASS='DATASET',ATTR=READ,RELEASE=1.9,       X
         USERID='*NONE*',GROUPID='SOMEGROUPID'
```

Example 7

Operation: Using the standard form, perform third party RACF authorization checking for a data set for a user connected to a group. Register 7 points to the data set name.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),                  X
         CLASS='DATASET',ATTR=READ,RELEASE=1.9,       X
         USERID='SOMEUSER',GROUPID='SOMEGROUPID'
```

Example 8

Operation: Using the standard form, perform third party RACF authorization checking for a data set for READ access only for a user. Register 7 points to the data set name, and A is an 8-byte declared field padded with zeros.

```
RACROUTE REQUEST=AUTH,ENTITY=((R7)),                  X
         CLASS='DATASET',ATTR=READ,RELEASE=1.9,       X
         USERID=A
```

RACROUTE REQUEST = AUTH (List Form)

The list form of the RACROUTE REQUEST = AUTH macro is written as follows:

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

REQUEST = AUTH	
,PROFILE = *profile addr* ,ENTITY = *resource name addr* ,ENTITY = (*resource name addr* ,CSA,PRIVATE) ,ENTITYX = *extended resource name addr* ,ENTITYX = (*extended resource name addr*,CSA,PRIVATE)	*profile addr*: A-type address *resource name addr*: A-type address *extended resource name addr*: A-type address **Note:** PROFILE, ENTITY, or ENTITYX is required on either the list or the execute form of the macro.
,VOLSER = *vol addr*	*vol addr*: A-type address **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: A-type address
,RELEASE = *number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ATTR = READ ,ATTR = UPDATE ,ATTR = CONTROL ,ATTR = ALTER	**Default:** ATTR = READ
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default:** DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: A-type address
,LOG = ASIS ,LOG = NOFAIL ,LOG = NONE ,LOG = NOSTAT	**Default:** LOG = ASIS
,OLDVOL = *old vol addr*	*old vol addr*: A-type address
,APPL = *'applname'* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: A-type address
,ACEE = *acee addr*	*acee addr*: A-type address
,OWNER = *owner id addr*	*owner id addr*: A-type address
,ACCLVL = *access level addr*	*access level addr*: A-type address
,RACFIND = YES ,RACFIND = NO	

,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE ,STATUS=EVERDOM	**Default**: STATUS=NONE
,USERID=*'userid'* ,USERID=*userid addr*	*userid*: 1-8 character user ID *userid addr*: A-type address
,GROUPID=*'groupid'* ,GROUPID=*groupid addr*	*groupid*: 1-8 character group ID *groupid addr*: A-type address
,LOGSTR=*logstr addr*	*logstr addr*: A-type address
,UTOKEN=*token addr*	*token addr*: A-type address
,RTOKEN=*rtoken addr*	*rtoken addr*: A-type address
,RECVR=*recvr addr*	*recvr addr*: A-type address or register (2)- (12)

,MF=L

The parameters are explained under the standard form of the RACROUTE REQUEST=AUTH macro with the following exception:

,MF=L
specifies the list form of the RACROUTE REQUEST=AUTH macro.

RACROUTE REQUEST = AUTH (Execute Form)

The execute form of the RACROUTE REQUEST = AUTH macro is written as follows:

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

REQUEST = AUTH	
,PROFILE = *profile addr* ,ENTITY = *resource name addr* ,ENTITY = (*resource name addr* ,CSA,PRIVATE) ,ENTITYX = *extended resource name addr* ,ENTITYX = (*extended resource name addr*,CSA,PRIVATE)	*profile addr*: RX-type address or register (2) - (12) *resource name addr*: RX-type address or register (2) - (12) *extended resource name addr*: RX-type address or register (2) - (12) **Note:** PROFILE, ENTITY, or ENTITYX required on either the list or the execute form of the macro.
,VOLSER = *vol addr*	*vol addr*: RX-type address or register (2) - (12) **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ATTR = READ ,ATTR = UPDATE ,ATTR = CONTROL ,ATTR = ALTER ,ATTR = *reg*	**Default:** ATTR = READ *reg*: register (2) - (12)
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default:** DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,LOG = ASIS ,LOG = NOFAIL ,LOG = NONE ,LOG = NOSTAT	**Default:** LOG = ASIS
,OLDVOL = *old vol addr*	*old vol addr*: RX-type address or register (2) - (12)
,APPL = *applname* ,APPL = *applname addr*	*applname*: *applname addr*: RX-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,OWNER = *owner id addr*	*owner id addr*: RX-type address or register (2) - (12)
,ACCLVL = *access level addr*	*access level addr*: RX-type address or register (2) - (12)
,RACFIND = YES ,RACFIND = NO	

,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE ,STATUS=EVERDOM	**Default**: STATUS=NONE
,USERID=*userid addr*	*userid addr*: RX-type address or register (2) - (12)
,GROUPID=*groupid addr*	*groupid addr*: RX-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: RX-type address or register (2) - (12)
,UTOKEN=*token addr*	*token addr*: RX-type address or register (2) - (12)
,RTOKEN=*rtoken addr*	*rtoken addr*: RX-type address or register (2) - (12)
,RECVR=*recvr addr*	*recvr addr*: RX-type address or register (2) - (12)
,MF=(**E**,*ctrl addr*)	*ctrl addr*: RX-type address, or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=AUTH macro with the following exceptions:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST=AUTH macro.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=AUTH macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

RACROUTE REQUEST = AUTH (Modify Form)

The modify form of the RACROUTE REQUEST = AUTH macro is written as follows:

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

REQUEST = AUTH	
,PROFILE = *profile addr* ,ENTITY = *resource name addr* ,ENTITY = (*resource name addr* ,CSA,PRIVATE) ,ENTITYX = *extended resource name addr* ,ENTITYX = (*extended resource name addr*,CSA,PRIVATE)	*profile addr*: RX-type address or register (2) - (12) *resource name addr: RX-type address or register (2) - (12)* *extended resource name addr*: RX-type address or register (2) - (12) **Note:** PROFILE, ENTITY, or ENTITYX is required on either the list or the execute form of the macro.
,VOLSER = *vol addr*	*vol addr*: RX-type address or register (2) - (12) **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ATTR = READ ,ATTR = UPDATE ,ATTR = CONTROL ,ATTR = ALTER ,ATTR = reg	**Default**: ATTR = READ *reg*: register (2) - (12)
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default**: DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,LOG = ASIS ,LOG = NOFAIL ,LOG = NONE ,LOG = NOSTAT	**Default**: LOG = ASIS
,OLDVOL = *old vol addr*	*old vol addr*: RX-type address or register (2) - (12)
,APPL = *applname* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: RX-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,OWNER = *owner id addr*	*owner ID addr*: RX-type address or register (2) - (12)
,ACCLVL = *access level addr*	*access level addr*: RX-type address or register (2) - (12)
,RACFIND = YES ,RACFIND = NO	

,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE ,STATUS=EVERDOM	**Default**: STATUS=NONE
,USERID=*userid* ,USERID=*userid addr*	*userid*: 1-8 character user ID *userid address*: RX-type address or register (2) - (12)
,GROUPID=*groupid* ,GROUPID=*groupid addr*	*groupid*: 1-8 character goup ID *groupid addr*: RX-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: RX-type address or register (2) - (12)
,UTOKEN=*token addr*	*token addr*: RX-type address or register (2) - (12)
,RTOKEN=*rtoken addr*	*rtoken addr*: RX-type address or register (2) - (12)
,RECVR=*recvr addr*	*recvr addr*: RX-type address or register (2) - (12)
,MF=(M,*ctrl addr*)	*ctrl addr*: RX-type address, or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=AUTH macro with the following exceptions:

,MF=(M,*ctrl addr*)
specifies the modify form of the RACROUTE REQUEST=AUTH macro.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACROUTE REQUEST=AUTH macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the modify form of the macro, the modify form of the macro will not be done. Instead, a return code X'64' will be generated.

RACROUTE REQUEST = DEFINE — Define a Resource to RACF (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see the following:

- **"RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435**
- **"RACDEF — Define a Resource to RACF (for RACF Release 1.8.1 or earlier)" on page 383.** (IBM recommends that you use RACROUTE with the REQUEST = DEFINE parameter rather than RACDEF.)

The RACROUTE REQUEST = DEFINE macro is used to define, modify, or delete resource profiles for RACF. It can also be used for special cases of authorization checking. RACF uses the resulting profiles to perform RACHECK authorization checking. The RACROUTE REQUEST = DEFINE caller must be authorized (APF-authorized, in system key 0-7, or in supervisor state)

A RACF user can change or add the RACROUTE REQUEST = DEFINE parameters, OWNER, LEVEL, UACC, or AUDIT by means of the RACDEF preprocessing and postprocessing exit routines.

The standard form of the RACROUTE REQUEST = DEFINE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = DEFINE	
,ENTITY = *profile name addr* ,ENTITYX = *extended profile name addr*	*profile name addr*: A-type address or register (2) - (12) *extended profile name addr*: A-type address or register (2) - (12)
,VOLSER = *vol addr*	*vol addr*: A-type address or register (2) - (12) **Note:** VOLSER is required only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used.
,TYPE = DEFINE ,TYPE = DEFINE,NEWNAME = *new resource name addr* ,TYPE = DEFINE,NEWNAMX = *extended new resource name addr* ,TYPE = ADDVOL,OLDVOL = *old vol addr* ,TYPE = CHGVOL,OLDVOL = *old vol addr* ,TYPE = DELETE	**Default**: TYPE = DEFINE *new resource name addr*: A-type address or register (2) - (12) *extended new resource name addr*: A-type address or register (2) - (12) *old vol addr*: A-type address or register (2) - (12)
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default**: DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: A-type address or register (2) - (12)
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name. *class name addr*: A-type address or register (2) - (12) **Default**: CLASS = DATASET

,MENTITY=*entity addr* ,MENTX=*extended entity addr*	*entity addr*: A-type address or register (2) - (12) *extended entity addr*: A-type address or register (2) - (12)
,MCLASS='*class name*' ,MCLASS=*class name addr*	*class name*: 1-8 character name *class name addr*: A-type address or register (2) - (12) **Default**: MCLASS=DATASET
,MVOLSER=*volser addr*	*volser addr*: A-type address or register (2) - (12)
,MGENER=ASIS ,MGENER=YES	**Default**: MGENER=ASIS
,ACEE=*acee addr*	*acee addr*: A-type address or register (2) - (12)
,UNIT=*unit addr*	*unit addr*: A-type address or register (2) - (12)
,OWNER=*owner id addr*	*owner id addr*: A-type address or register (2) - (12)
,LEVEL=*number* ,LEVEL=*reg*	**Default**: LEVEL=zero *reg*: register (2) - (12)
,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=EXECUTE ,UACC=NONE ,UACC=*reg*	*reg*: register (2) - (12)
,DATA=*data addr*	*data addr*: A-type address or register (2) - (12)
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value*(*access level*),*audit value*(*access level*)) ,AUDIT=*reg*	**Note:** AUDIT is valid only if TYPE=DEFINE is specified. *audit value*: ALL, SUCCESS, or FAILURES *access level*: READ, UPDATE, CONTROL, or ALTER **Default**: AUDIT=READ *reg*: register (2) - (12)
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default**: CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default**: WARNING=NO **Note:** WARNING is valid only if TYPE=DEFINE is specified.
,RELEASE=*number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,EXPDT=*exper-date addr* ,EXPDTX=*extended expir-date addr* ,RETPD=*retn-period addr*	*exper-date addr*: A-type address or register (2) - (12) *extended expir-date addr*: A-type address or register (2) - (12) *retn-period addr*: A-type address or register (2) - (12) **Default**: see description of parameter
,ACCLVL=*access value addr* ,ACCLVL=(*access value addr*,*parm list addr*)	*access value addr*: A-type address or register (2) - (12) *parm list addr*: A-type address or register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD

,SECLVL=*addr*	*addr*: A-type address or register (2) - (12)
,ERASE=YES ,ERASE=NO	**Default**: ERASE=NO
,NOTIFY=*notify-id addr*	*notify-id addr*: A-type address or register (2) - (12)
,ENVIR=VERIFY	Specifies that only verification is to be done **Default**: Normal RACROUTE REQUEST=DEFINE processing
,RESOWN=*resource owner addr*	*resource owner addr*: A-type address or register (2) - (12)
,STORCLA=*storage class addr*	*storage class addr*: A-type address or register (2) - (12)
,MGMTCLA=*management type addr*	*management type addr*: A-type address or register (2) - (12)
,SECLABL=*addr*	*addr*: A-type address or register (2) - (12)

The parameters are explained as follows:

,ENTITY=*profile name addr*
,ENTITYX=*extended profile name addr*
specifies the address:

- ,ENTITY=*profile name addr* specifies the address of the name of the discrete or generic profile that is to be defined to, modified, or deleted from RACF. The profile name is a 44-byte DASD data set name for CLASS=DATASET or a 6-byte volume serial name for CLASS=DASDVOL or CLASS=TAPEVOL. The lengths of all other profile names are determined by the class descriptor table. The name must be left justified in the field and padded with blanks.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

- ,ENTITYX=*extended profile name addr* specifies the address of a structure that consists of two 2-byte length fields, followed by the entity name.
  - The first 2-byte field specifies a buffer length which can be from 0 to 255 bytes. This length field only refers to the length of the buffer that contains the entity name; it does not include the length of either length field.
  - The second 2-byte field specifies the actual length of the entity name. This length field only includes the length of the actual name without any trailing blanks; it does not include the length of either length field.

  These two length fields can be used in several different ways:

  - If the length of the entity name is known, you can specify 0 in the first field and specify the length of the entity name in the second field. Note that when you specify the second field, each byte counts; this means RACF processes the entity name with the exact length you specify.
  - If you choose to use a buffer area in which to place the entity name, specify the first field to designate the length of the buffer. In regard to the second field, you can do one of two things:
    - If you know the length of an entity name, you would specify the length in the second field. (Note that the length of the first field can be from 0 to 255, but must be equal to or more than the length of the second field.)
    - If you do not know the length of the entity name, you would specify 0 in the second field, in which case RACF would be responsible for counting the number of characters in the entity name.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39

characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

To use this keyword, you must also specify RELEASE=1.9 or a later release number.

Recommendation

IBM recommends that you use ENTITYX rather than ENTITY for two reasons:

- With ENTITYX, if you know the length of the entity name, ENTITYX allows you to pass that information to RACF. Doing so can result in slightly faster processing.
- With ENTITY, the entity name you pass to RACF must be in a buffer, the size of which is determined by the length in the CDT. If the MAXLNTH of a class increases in the future, you would have to modify your program to use a larger buffer. By using ENTITYX, you avoid this possible problem because you have removed the CDT-dependency from your program.

,VOLSER= *vol addr*
specifies the address of the volume serial number:

- For TYPE=ADDVOL, of the new volume to be added to the definition of the data set.
- For TYPE=ADDVOL and CLASS=TAPEVOL, of the new volume being added to the tape volume set identified by ENTITY or ENTITYX.
- For TYPE=DEFINE and CLASS=DATASET, of the catalog (for a VSAM data set), or of the volume on which the data set resides (for a non-VSAM data set)

The volume serial number is optional if DSTYPE=M is specified; it is ignored if the profile name is generic.

The field pointed to by the specified address contains the volume serial number (padded to the right with blanks, if necessary, to make six characters)

,TYPE=DEFINE
,TYPE=DEFINE,NEWNAME= *new resource name addr*
,TYPE=DEFINE,NEWNAMX= *extended new resource name addr*
,TYPE=ADDVOL,OLDVOL= *old vol addr*
,TYPE=CHGVOL,OLDVOL= *old vol addr*
,TYPE=DELETE
specifies the type of action to be taken:

- TYPE=DEFINE - The definition of the resource is added to the RACF data set, and the current user is established as the owner of the defined entity.
- TYPE=DEFINE,NEWNAME= or NEWNAMX=

  If NEWNAME is specified, the address points to a 44-byte field containing the new name for the resource that is to be renamed. NEWNAME is only valid with CLASS=DATASET, FILE, and DIRECTORY. NEWNAME is not valid with DSTYPE=T.

  If NEWNAMX is specified, the address points to a structure that consists of two 2-byte length fields, followed by the entity name.

  - The first 2-byte field specifies a buffer length which can be from 0 to 255 bytes. This length field only refers to the length of the buffer that contains the entity name; it does not include the length of either length field.
  - The second 2-byte field specifies the actual length of the entity name. This length field only includes the length of the actual name without any trailing blanks; it does not include the length of either length field.

  These two length fields can be used in several different ways:

  - If the length of the entity name is known, you can specify 0 in the first field and specify the length of the entity name in the second field. Note that when you specify the second field, each byte counts; this means the entity name specified will be added to the RACF data base using the specified length.

- If you choose to use a buffer area in which to place the entity name, specify the first field to designate the length of the buffer. In regard to the second field, you can can do one of two things:
  - If you know the length of an entity name, you would specify the length in the second field. (Note that the length of the first field can be from 0 to 255, but must be equal to or more than the length of the second field.)
  - If you do not know the length of the entity name, you would specify 0 in the second field, in which case RACF would be responsible for counting the number of characters in the entity name.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

Recommendation

IBM recommends that you use NEWNAMX rather than NEWNAME for two reasons:

- With NEWNAMX, if you know the length of the entity name, NEWNAMX allows you to pass that information to RACF. Doing so can result in slightly faster processing.
- With NEWNAME, the entity name you pass to RACF must be in a buffer, the size of which is determined by the length in the CDT. If the MAXLNTH of a class increases in the future, you would have to modify your program to use a larger buffer. By using NEWNAMX, you avoid this possible problem because you have removed the CDT-dependency from your program.

NEWNAMX is only valid with CLASS=DATASET, CLASS=FILE, or CLASS=DIRECTORY. NEWNAMX is not valid with DSTYPE=T.

The following parameters are ignored if you specify NEWNAME or NEWNAMX: INSTLN, MENTITY, MENTX, MCLASS, MVOLSER, MGENER, UNIT, OWNER, LEVEL, UACC, DATA, AUDIT, FILESEQ, EXPDT, RETPD, EXPDTX, ACCLVL, TAPELBL, CATEGORY, SECLVL, ERASE, NOTIFY, and WARNING.

- TYPE=ADDVOL - The new volume is added to the definition of the specified resource. For the DATASET class, the OLDVOL address specifies a previous volume of a multi-volume data set. For the TAPEVOL class, the ENTITY or ENTITYX address specifies a previous volume of a tape volume set. This parameter applies only to discrete profiles.
- TYPE=CHGVOL - The volume serial number in the definition of the specified resource is changed from the old volume serial number identified in OLDVOL to the new volume serial number identified in the VOLSER parameter. This parameter applies only to discrete profiles. TYPE=CHGVOL is not valid with DSTYPE=T.
- TYPE=DELETE - The definition of the resource is removed from the RACF data set. (For a multivolume data set or a tape volume set, only the specified volume is removed from the definition.)

,DSTYPE=N
,DSTYPE=V
,DSTYPE=M
,DSTYPE=T

specifies the type of data set associated with the request:

- N for non-VSAM
- V for VSAM
- M for model profile
- T for tape

If DSTYPE=T is specified and tape data set protection is not active, the processing will be the same as for RACROUTE REQUEST=DEFINE CLASS='TAPEVOL'. Specify DSTYPE only for CLASS=DATASET.

,INSTLN = *parm list addr*
specifies the address of an area that is to contain parameter information meaningful to the RACDEF installation exit routines. This information is passed to the installation exit routines when they are given control from the RACDEF routine.

The INSTLN parameter can be used by an application program acting as a resource manager that needs to pass information to the RACDEF installation exit routines.

,CLASS = *'class name'*
,CLASS = *class name addr*
specifies that a profile is to be defined, modified, or deleted in the specified class. If an address is specified, the address must point to a one-byte length field followed by the class name (for example, DATASET or TAPEVOL) The class name should be no longer than eight characters.

,MENTITY = *entity addr*
,MENTX = *extended entity address*
specifies the address of the name of the discrete or generic profile that is to be used:

- ,MENTITY = *entity addr* specifies the address of the name of the discrete or generic profile that is to be used as a model in defining the ENTITY or ENTITYX profile. The profile can belong to any class, as specified by the MCLASS parameter, and can be either a discrete or a generic profile. MENTITY can be specified with TYPE = DEFINE but not with TYPE = DEFINE,NEWNAME = *new resource name addr*. For data sets, the name is contained in a 44-byte field pointed to by the specified address. For general resource classes, the length of the field is determined by the CDT. The name is left justified in the field and padded with blanks.
- ,MENTX = *extended entity address* specifies the address of the name of the discrete or generic profile that is to be used as a model from which to define the ENTITY or ENTITYX profile. The structure consists of two 2-byte length fields, followed by the entity name.
  - The first 2-byte field specifies a buffer length which can be from 0 to 255 bytes. This length field only refers to the length of the buffer that contains the entity name from which you are modeling; it does not include the length of either length field.
  - The second 2-byte field specifies the actual length of the entity name from which you are modeling. This length field only includes the length of the actual name without any trailing blanks; it does not include the length of either length field.

  These two length fields can be used in several different ways:

  - If the length of the entity name from which you are modeling is known, you can specify 0 in the first field and specify the length of the entity name in the second field. Note that when you specify the second field, each byte counts; this means the entity name specified will be used as a model, using the specified length.
  - If you choose to use a buffer area in which to place the entity name, specify the first field to designate the length of the buffer. In regard to the second field, you can do one of two things:
    - If you know the length of the entity name which you are using as a model, you would specify the length in the second field. (Note that the length of the first field can be from 0 to 255, but must be equal to or more than the length of the second field.)
    - If you do not know the length of the entity name that you are using as a model, you would specify 0 in the second field, in which case RACF would be responsible for counting the number of characters in the entity name.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND = NO.

Recommendation

IBM recommends that you use MENTX rather than MENTITY for two reasons:

- With MENTX, if you know the length of the entity name, MENTX allows you to pass that information to RACF. Doing so can result in slightly faster processing.
- With MENTITY, the entity name you pass to RACF must be in a buffer, the size of which is determined by the length in the CDT. If the MAXLNTH of a class increases in the future, you would have to modify your program to use a larger buffer. By using MENTX, you avoid this possible problem because you have removed the CDT-dependency from your program.

The profile can belong to any class, as specified by the MCLASS parameter, and can be either a discrete or generic profile. MENTX can be specified with TYPE=DEFINE, but not with TYPE=DEFINE,NEWNAME= or TYPE=DEFINE,NEWNAMX=.

,MCLASS= *'class name'*
,MCLASS= *class name addr*
specifies the class to which the profile defined by MENTITY= belongs. If an address is specified, the address must point to a one-byte length field followed by the class name. The class name should be no longer than eight characters. The default is MCLASS=DATASET.

,MVOLSER= *volser addr*
specifies the address of the volume serial number of the volume associated with the profile in the MENTITY operand. The field pointed to by the specified address contains the volume serial number, padded to the right with blanks, if necessary, to make six characters.

If you specify MENTITY or MENTX and CLASS=DATASET, you must specify MVOLSER with the name of the VOLSER or with blanks.

If you specify with blanks, the discrete MENTITY or MENTX data set profile name must be unique, meaning it has no duplicates on the data base. In this case, RACF determines the correct MVOLSER.

,MGENER=ASIS
,MGENER=YES
specifies whether the profile name defined by MENTITY or MENTX is to be treated as a generic name.

- If MGENER=ASIS is specified, the profile name is considered a generic only if it contains a generic character: an asterisk (*) or a percent sign (%)
- If MGENER=YES is specified, the profile name is considered a generic, even if it does not contain a generic character: an asterisk (*) or a percent sign (%)

MGENER is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*)

,ACEE= *acee addr*
specifies the address of the accessor environment element (ACEE) to be used during RACDEF processing. If no ACEE is specified, RACF uses the TASK ACEE pointer (TCBSENV) in the extended TCB. If the TASK ACEE pointer is zero, RACF uses the main ACEE. The main ACEE's address is in the ASXBSENV field in the address space extension block.

,UNIT= *unit addr*
specifies the address of a field containing unit information. UNIT is valid only if TYPE=CHGVOL or TYPE=DEFINE is specified. If a unit address is specified, the unit information in the data set profile is replaced by the unit information pointed to by this unit address. The unit address must point to a field containing a one-byte length field (whose value can range from 4 through 8) followed by the actual unit information. If the value in the length field is 4, the unit information is assumed to contain a copy of the information in the UCBTYP field of the UCB. Otherwise the unit information is assumed to be in the generic form (for example, 3330-1) This parameter is ignored for generic names.

,OWNER = *owner id addr*
specifies the address of a field containing the profile owner's id. OWNER is valid if TYPE = DEFINE is specified. The owner's id must be a valid (RACF-defined) userid or group name. The address must point to an 8-byte field containing the owner's name, left-justified and padded with blanks.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Note: RACF does not check the validity of the owner's id if it has been added or modified by either the RACDEF preprocessing or postprocessing exit routines, or both.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,LEVEL = *number*
,LEVEL = *reg*
specifies a level value for the profile. LEVEL is valid only if TYPE = DEFINE is specified. The level number must be a valid decimal number in the range 0 to 99. If a register is specified, its low-order byte must contain the binary representation of the number.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Note: RACF does not check the validity of this number if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routines.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,UACC = ALTER
,UACC = CONTROL
,UACC = UPDATE
,UACC = READ
,UACC = EXECUTE
,UACC = NONE
,UACC = *reg*
specifies a universal access authority for the profile. UACC is valid only if TYPE = DEFINE is specified. UACC must contain a valid access authority (EXECUTE, ALTER, CONTROL, UPDATE, READ, or NONE).

To use the EXECUTE keyword, your system must be running on MVS/ESA with DFP 3.1.0 installed. EXECUTE authority means that the user has *only* the ability to execute the program; the user cannot READ the program. If your system is *not* running with DFP 3.1.0, an EXECUTE access attempt will be treated as NONE.

If a register is specified, the low-order byte must contain one of the following valid access authorities:

X'80' - ALTER
X'40' - CONTROL
X'20' - UPDATE
X'10' - READ
X'01' - NONE
X'08' - EXECUTE

Note: RACF does not check the validity of the universal access authority if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routine.

,DATA = *data addr*
specifies the address of a field that contains up to 255 characters of installation-defined data to be placed in the profile. DATA is valid only if TYPE = DEFINE is specified. The data address must point to a field containing a one-byte length field (whose value can range from 0 to 255) followed by the actual installation-defined data.

,AUDIT = NONE
,AUDIT = *audit value*
,AUDIT = (*audit value***(***access level***),***audit value***(***access level***),. . .)**
,AUDIT = *reg*

specifies the types of accesses and the access levels that are to be logged to the SMF data set. AUDIT is valid only if TYPE = DEFINE is specified.

For *audit value*, specify one of the following: ALL, SUCCESS, or FAILURES. You may optionally specify an *access level*(*access authority*) following each *audit value*.

Access Levels:

- EXECUTE, this access level is not audited. If you specify FAILURES (READ), RACF logs the READ attempt as a failure, but allows EXECUTE access to the data set.
- READ, the default access level value, logs access attempts at any level.
- UPDATE logs access attempts at the UPDATE, CONTROL, and ALTER levels.
- CONTROL logs access attempts at the CONTROL and ALTER levels.
- ALTER logs access attempts at the ALTER level only.

Note: For more information about specific audit values and access levels, please see the *RACF Command Language Reference*.

RACF resolves combinations of conflicting specifications by using the most encompassing specification. Thus, in the case of the following:

```
ALL(UPDATE),FAILURES(READ)
```

RACF assumes SUCCESS(UPDATE),FAILURES(READ)

For compatibility with previous releases, register notation can also be specified as AUDIT = *reg* if the register is not given as a symbolic name ALL, SUCCESS, or FAILURES.

Logging is controlled separately for SUCCESS and FAILURES, and can also be suppressed or requested via the RACHECK post-processing installation exit routine.

If a register is specified, its low-order byte must contain one of the following valid audit values:

Bit	Meaning
0	ALL
1	SUCCESS
2	FAILURES
3	NONE
4-5	Qualifier for SUCCESS
6-7	Qualifier for FAILURES

The qualifier codes are as follows:

00	READ
01	UPDATE
10	CONTROL
11	ALTER

Only one of bits 0-3 can be on. If ALL is specified, the two qualifier fields can be used to request different logging levels for successful and unsuccessful events.

Note: RACF does not check the validity of the audit type if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routine.

,RACFIND = YES
,RACFIND = NO

specifies whether or not a discrete profile is involved in RACDEF processing. When TYPE = DEFINE is specified, RACFIND = YES means that a discrete profile is to be created. When TYPE = DELETE, DEFINE with NEWNAME or NEWNAMX, CHGVOL, or ADDVOL is specified, RACFIND = YES means that a discrete profile already exists.

RACFIND=NO means (when TYPE=DEFINE) that no discrete profile is to be created, but some authorization checking is required. For other types of action, no discrete profile should exist.

Note: The RACFIND keyword should only be used with the DATASET class.

,CHKAUTH=YES
,CHKAUTH=NO
specifies whether or not an internal RACHECK ATTR=ALTER is to be done to verify that the user is authorized to perform the operation.

CHKAUTH=YES is valid when either TYPE=DEFINE,NEWNAME or NEWNAMX, or TYPE=DELETE is specified.

For DSTYPE=T, specifies that an internal RACHECK ATTR=UPDATE will be done to verify that the user is authorized to define a data set (TYPE=DEFINE), delete a data set (TYPE=DELETE), or add a volume (TYPE=ADDVOL)

,GENERIC=YES
,GENERIC=ASIS
specifies whether the resource name is treated as a generic profile name. If GENERIC is specified with CLASS=DEFINE, NEWNAME or NEWNAMX, then GENERIC applies to both the old and new names. GENERIC is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*)

- If GENERIC=YES is specified, the resource name is considered a generic profile name, even if it does not contain a generic character: an asterisk (*) or a percent sign (%)
- If GENERIC=ASIS is specified, the resource name is considered a generic only if it contains a generic character: an asterisk (*) or a percent sign (%)

,WARNING=YES
,WARNING=NO
WARNING is valid only if TYPE=DEFINE is specified. If WARNING=YES is specified, access is granted to the resource and the event is logged as a warning if either the SUCCESS and/or FAILURES logging is requested.

,RELEASE=*number*
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=DEFINE macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,FILESEQ=*number*
,FILESEQ=*reg*
specifies the file sequence number of a tape data set on a tape volume or within a tape volume set. The *number* must be in the range 1 - 9999. If a register is specified, it must contain the file sequence number in the low-order half-word. If CLASS=DATASET and DSTYPE=T are not specified, FILESEQ is ignored.

,EXPDT=*exper-date addr*
,EXPDTX=*extended expir-date addr*
,RETPD=*retn-period addr*
specifies the address containing information about the data set's expiration date or RACF security retention period.

- EXPDT=*exper-date addr* specifies the address of a three-byte field containing the data set's expiration date. The date is given in packed decimal form as YYDDDF, where YY is the year and DDD is the day number. The year must be in the range 01 through 99, and the day number must be in the range 1 through 366. All fields are right justified.

- EXPDTX=*extended expir-date addr* specifies the address of a 4-byte field that contains the address of the data set's expiration date. The date is given in packed decimal form as CCYYDDDF, where CC is the century change greater than 19, YY is the year, and DDD is the day number. The year must be in the range 01 through 99. The day must be in the range 1 through 366. All fields are right justified. When you want to represent 19 for the century, then you must specify CC as 00; when you want to represent 20 for the century, then you must specify CC as 01.
- RETPD=*retn-period addr*. specifies the address of a two-byte binary field containing the number of days after which RACF protection for the data set expires. The value specified must be in the range 1 through 65533. To indicate that there is no expiration date, specify 65534.

If you do not specify any of these parameters, a default RACF security retention period is obtained from the RETPD keyword specified on a prior RACF SETROPTS command.

These parameters are valid only if CLASS=DATASET and DSTYPE=T.

,ACCLVL=*access value addr*
,ACCLVL=(*access value addr,parm list addr***)**
specifies the tape label access level information for the MVS tape label functions. The address must point to a field containing a one-byte length field (with a value that can range from 0-8) followed by an eight-character string that will be passed to the RACDEF installation exit routines. The parameter list address points to a parameter list containing additional information to be passed to the RACDEF installation exit routines.

RACF does not check or modify this information.

,TAPELBL=STD
,TAPELBL=BLP
,TAPELBL=NL
specifies the type of tape labelling to be done:

- STD - IBM or ANSI standard labels.
- BLP - bypass label processing.
- NL - non-labeled tapes.

For TAPELBL=BLP, the user must have the requested authority to the profile ICHBLP in the general resource class FACILITY. For TAPELBL=NL or BLP, the user will not be allowed to protect volumes with volume serial numbers in the format "Lnnnnn."

The TAPELBL parameter is passed to the RACDEF installation exits.

This parameter is primarily intended for use by data management routines to indicate the label type from the LABEL keyword on the JCL statement.

This parameter is only valid for CLASS=DATASET and DSTYPE=T, or CLASS=TAPEVOL.

,SECLVL=*addr*
specifies the address of a list of installation-defined security level identifiers. Each identifier is a half word, containing a value that corresponds to an installation-defined security level name.

The identifiers must be in the range 1 - 254. Only one identifier may be passed in the list.

The list must start with a full word containing the number of entries in the list (currently, only 0 or 1)

,ERASE = YES
,ERASE = NO
specifies whether the DASD data set, or the released space, is to be erased when it is deleted or part of its space is to be released for reuse.

- If ERASE = YES is specified, the data set will be erased when it is deleted, or released for reuse.
- If ERASE = NO is specified, the data set will not be erased, deleted, or released.

Note: This parameter may be overridden by the RACF SETROPTS command.

The default is ERASE = NO.

,NOTIFY = *notify-id addr*
specifies the address of an eight-byte area containing the userid of the RACF-defined user who is to be notified when an unauthorized attempt to access the resource protected by this profile is detected.

,ENVIR = VERIFY
specifies that only verification is to be done. If you specify ENVIR = VERIFY, you must also specify TYPE = DEFINE, RESOWN, the current RELEASE level, and either MGMTCLA or STORCLA, or both .

When you specify ENVIR = VERIFY, RACROUTE REQUEST = DEFINE calls RACHECK to verify that the user specified on the RESOWN keyword has the authority to create a data set in the specified resource class. To verify the authority of a non-RACF defined user to the specified resource, specify *NONE* for the RESOWNER keyword. In DFP support, the two resource classes are the MGMTCLA and the STORCLA.

Note: If you do not specify ENVIR = VERIFY, normal RACROUTE REQUEST = DEFINE processing occurs.

,RESOWN = *resource owner addr*
specifies the address of a field containing the resource owner's id. If you specify RESOWN, you must also specify TYPE = DEFINE and the current RELEASE parameter. The resource owner's id must be either a valid (RACF-defined) userid or group name, or *NONE*. If the resource owner's id is specified as *NONE*, then RACF performs third-party RACHECK using USERID = *NONE*. The address must point to a 2-byte field followed by the resource owner's name.

,STORCLA = *storage class addr*
specifies the address of the storage class to which the resource owner must have authority. The address must point to a 2-byte field followed by the management class name. If you specify STORCLA, you must also specify TYPE = DEFINE, RESOWN, the current RELEASE parameter and ENVIR = VERIFY.

,MGMTCLA = *management type addr*
specifies the address of of a management class to which the resource owner must have authority. The address must point to an 8-byte field that contains a management class name preceded by a halfword length. If you specify MGMTCLA, you must also specify TYPE = DEFINE, RESOWN, the current RELEASE parameter, and ENVIR = VERIFY.

,SECLABL = *addr*
specifies the address of an 8-byte left-justified character field containing the SECLABEL.

An installation would use SECLABELs to establish an association between a specific RACF security level (SECLEVEL) and a set of (zero or more) RACF security categories (CATEGORY) In a B1 system, it is necessary to use SECLABELs to prevent the unauthorized movement of data from one level to another when multiple levels of data are in use on the system at the same time. See the *RACF Security Administrator's Guide* for further information.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	RACROUTE REQUEST = DEFINE has completed successfully. Register 0 contains one of the following reason codes: 00 Indicates a normal completion. 04 Indicates RACFIND = NO was specified and no generic profile applying to the data set was found. 08 Indicates that MODEL was specified, but the SECLABEL value has not been copied because of one of two reasons: • SETROPTS SECLABELCONTROL is on, but issuer is not RACF SPECIAL, or • SETROPTS MLSTABLE is on, but SETROPTS MLQUIET is not
04	RACROUTE REQUEST = DEFINE has completed processing. Register 0 contains one of the following reason codes: 00 Indicates the following: - For TYPE = DEFINE, the resource name was previously defined. - For TYPE = DEFINE,NEWNAME or NEWNAMX, the new resource name was previously defined. - For TYPE = DELETE, the resource name was not previously defined. 04 Indicates for TYPE = DEFINE that the data set name was previously defined on a different volume and that the option disallowing duplicate data sets was specified in ICHSECOP at IPL.
08	RACROUTE REQUEST = DEFINE has completed processing. Register 0 contains one of the following reason codes: 00 Indicates the following: - For TYPE = DEFINE, RACF has failed the check for authority to allocate a data set or create a profile with the specified name. - For TYPE = DELETE or TYPE = DEFINE,NEWNAME or NEWNAMX, if CHKAUTH = YES is specified, RACF has failed the authorization check. - For TYPE = ADDVOL,OLDVOL the old value was not defined. 04 Indicates for TYPE = DEFINE that no profile was found to protect the data set and that the RACF protect-all option is in effect. 08 Indicates TYPE = DEFINE or TYPE = ADDVOL,OLDVOL and DSTYPE = T were specified, and the user is not authorized to define a data set on the specified volume, or an ADDVOL was attempted to add a 43rd volume, but the maximum number of volumes that a data set can span is 42. 0C Indicates TYPE = DEFINE and DSTYPE = T were specified, and the user is not authorized to define a data set with the specified name. 10 Indicates DSTYPE = T or CLASS = TAPEVOL was specified, and the user is not authorized to specify LABEL = (,BLP) 1C Indicates that the user is not authorized to issue RACDEF when the system is in a tranquil state. 20 Indicates the data set owner is not authorized to use the specified DFP storage class.

24 Indicates the data set owner is not authorized to use the specified DFP management class.

0C For TYPE=DEFINE,NEWNAME or NEWNAMX, the old resource name was not defined; or for CLASS=DATASET, if the generation data group (GDG) modeling function is active, an attempt was made to rename a GDG name to a name that requires the creation of a new profile; or if generic profile checking is active, the old resource name was protected by a generic profile and there is no generic profile that will protect the new resource name. This last case refers only to an attempt to rename an existing profile, which cannot be found.

10 For TYPE=DEFINE with MENTITY or MENTX, the model resource was not defined.

64 Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=DEFINE macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example 1

Operation: Invoke RACF to define a discrete profile for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name. All successful requests for update authority to the data set are to be audited, as well as all unsuccessful ones.

```
RACROUTE REQUEST=DEFINE,ENTITY=(R7),VOLSER=(R8),CLASS='DATASET',  X
         AUDIT=(SUCCESS(UPDATE),FAILURES),                        X
         RACFIND=YES,RELEASE=1.9
```

Example 2

Operation: Use the standard form of the RACROUTE REQUEST=DEFINE macro to define a discrete data set profile for a non-VSAM DASD data set. The data set for which you are creating a profile is a non-VSAM DASD data set named DSNAME. It resides on a volume id named VOLID. You want to create a discrete profile by specifying the RACFIND keyword. In addition, you want to notify the user called USERNAME of any access attempts that have been rejected because they exceed the UACC of READ that you are allowing.

```
RACROUTE REQUEST=DEFINE,ENTITY=DSNAME,VOLSER=VOLID,    X
         CLASS='DATASET',UACC=READ,                     X
         RACFIND=YES,NOTIFY=USERNAME,RELEASE=1.9
```

Example 3

Operation: Use the standard form of the macro to check the authority of a user to define a discrete data set profile for a non-VSAM DASD data set, but do not actually build the profile. The name of the data set is DSNAME.

```
RACROUTE REQUEST=DEFINE,ENTITY=DSNAME,VOLSER=VOLID,  X
         CLASS='DATASET',RACFIND=NO,RELEASE=1.9
```

Example 4

Operation: Use the standard form of the macro to define a generic data set profile named PROFNAME. Use the discrete profile named MDELPROF whose volser is in MDELVOL as a model for the new profile. Notify the user named USERNAME of any access attempts that have been rejected because they exceed the UACC of READ which you are allowing.

```
RACROUTE REQUEST=DEFINE,ENTITY=PROFNAME,                  X
         CLASS='DATASET',GENERIC=YES,MENTITY=MDELPROF,    X
         MVOLSER=MDELVOL,UACC=READ,                       X
         NOTIFY=USERNAME,RELEASE=1.9
```

Example 5

Operation: Use the standard form of the macro to define a tape volume profile for a volume whose id is VOLID. Allow a universal access level of READ.

```
RACROUTE REQUEST=DEFINE,ENTITY=VOLID,CLASS='TAPEVOL',UACC=READ,
         RELEASE=1.9
```

Example 6

Operation: Use the standard form of the macro to delete a discrete data set profile named DSNAME located on the volume named VOLID.

```
RACROUTE REQUEST=DEFINE,TYPE=DELETE,ENTITY=DSNAME,       X
         VOLSER=VOLID,CLASS='DATASET',RELEASE=1.9
```

RACROUTE REQUEST = DEFINE (List Form)

The list form of the RACROUTE REQUEST = DEFINE macro is written as follows:

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

REQUEST = DEFINE	
,ENTITY = *profile name addr* ,ENTITYX = *extended profile name addr*	*profile name addr*: A-type address. *extended profile name addr*: A-type address **Note:** ENTITY or ENTITYX must be specified on either the list or the execute form of the macro.
,VOLSER = *vol addr*	*vol addr*: A-type address **Note:** VOLSER is required (on either LIST or EXECUTE) only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used.
,TYPE = DEFINE ,TYPE = DEFINE,NEWNAME = *new resource name addr* ,TYPE = DEFINE,NEWNAMX = *extended new resource name addr* ,TYPE = ADDVOL,OLDVOL = *old vol addr* ,TYPE = CHGVOL,OLDVOL = *old vol addr* ,TYPE = DELETE	**Default**: TYPE = DEFINE *new resource name addr*: A-type address *extended new resource name addr*: A-type address *old vol addr*: A-type address
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default**: DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: A-type address
,CLASS = *'class name'* ,CLASS = *class name addr*	*'class name'*: 1-8 character name. *class name addr*: A-type address **Default**: CLASS = DATASET
,MENTITY = *entity addr* ,MENTX = *extended entity addr*	*entity addr*: A-type address *extended entity addr*: A-type address
,MCLASS = *'class name'* ,MCLASS = *class name addr*	*'class name'*: 1-8 character name. *class name addr*: A-type address **Default**: MCLASS = DATASET
,MVOLSER = *volser addr*	*volser addr*: A-type address
,MGENER = ASIS ,MGENER = YES	**Default**: MGENER = ASIS
,ACEE = *acee addr*	*acee addr*: A-type address
,UNIT = *unit addr*	*unit addr*: A-type address
,OWNER = *owner id addr*	*owner id addr*: A-type address
,LEVEL = *number* ,LEVEL = *reg*	**Default**: LEVEL = zero. *reg*: register (2) - (12)

,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=EXECUTE ,UACC=NONE ,UACC=*reg*	*reg*: register (2) - (12)
,DATA=*data addr*	*data addr*: A-type address
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value(access level),audit value(access level)*)	*audit value*: ALL, SUCCESS, or FAILURES *access level*: READ, UPDATE, CONTROL, or ALTER **Default**: AUDIT=READ
,AUDIT=*reg*	*reg*: register (2) - (12)
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default**: CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default**: WARNING=NO **Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=*number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,EXPDT=*exper-date addr* ,EXPDTX=*extended expir-date addr* ,RETPD=*retn-period addr*	*exper-date addr*: A-type address *extended expir-date addr:* A-type address *retn-period addr*: A-type address
,ACCLVL=*access value addr* ,ACCLVL=(*access value addr,parm list addr*)	*access value addr*: A-type address *parm list addr*: A-type address
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,SECLVL=*addr*	*addr*: A-type address
,SECLABL=*addr*	*addr*: A-type address
,ERASE=YES ,ERASE=NO	**Default**: ERASE=NO
,ENVIR=VERIFY	Specifies that only verification is to be done. **Default**: Normal RACROUTE REQUEST=DEFINE processing.
,RESOWN=*resource owner addr*	*resource owner addr*: A-type address
,STORCLA=*storage class addr*	*storage class addr*: A-type address
,MGMTCLA=*management type addr*	*management type addr*: A-type address **Default**: See description of parameter.

,NOTIFY = *notify-id addr* *notify-id addr*: A-type address

,MF = L

The parameters are explained under the standard form of the RACROUTE REQUEST = DEFINE macro, with the following exception:

,MF = L
specifies the list form of the RACROUTE REQUEST = DEFINE macro.

RACROUTE REQUEST = DEFINE (Execute Form)

The execute form of the RACROUTE REQUEST = DEFINE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = DEFINE	
,ENTITY = *profile name addr* ,ENTITYX = *extended profile name addr*	*profile name addr*: RX-type address *extended profile name addr*: RX-type address or register (2) - (12) **Note:** ENTITY or ENTITYX must be specified on either the list or the execute form of the macro.
,VOLSER = *vol addr*	*vol addr*: RX-type address or register (2) - (12) **Note:** VOLSER is required only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used.
,TYPE = DEFINE ,TYPE = DEFINE,NEWNAME = *new resource name addr* ,TYPE = DEFINE,NEWNAMX = *extended new resource name addr* ,TYPE = ADDVOL,OLDVOL = *old vol addr* ,TYPE = CHGVOL,OLDVOL = *old vol addr* ,TYPE = DELETE	**Default**: TYPE = DEFINE *new resource name addr*: RX-type address or register (2) - (12) *extended new resource name addr*: RX-type address or register (2) - (12) *old vol addr*: RX-type address or register (2) - (12)
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default**: DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12) **Default**: CLASS = DATASET
,MENTITY = *entity addr* ,MENTX = *extended entity addr*	*entity addr*: RX-type address or register (2) - (12) *extended entity addr*: RX-type address or register (2) - (12)
,MCLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12) **Default**: MCLASS = DATASET
,MVOLSER = *volser addr*	*volser addr*: RX-type address or register (2) - (12)
,MGENER = ASIS ,MGENER = YES	**Default**: MGENER = ASIS
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,UNIT = *unit addr*	*unit addr*: RX-type address or register (2) - (12)
OWNER = *owner id addr*	*owner id addr*: RX-type address or register (2) - (12)
,LEVEL = *number* ,LEVEL = *reg*	**Default**: LEVEL = zero *reg*: register (2) - (12)
,UACC = ALTER	

,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=EXECUTE ,UACC=NONE ,UACC=*reg*	*reg*: register (2) - (12)
,DATA=*data addr*	*data addr*: RX-type address or register (2) - (12)
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value (access level),audit value(access level)*)	*audit value*: ALL, SUCCESS, or FAILURES *access level*: READ, UPDATE, CONTROL, or ALTER **Default**: AUDIT=READ
,AUDIT=*reg*	*reg*: register (2) - (12)
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default**: CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default**: WARNING=NO **Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,EXPDT=*exper-date addr* ,EXPDTX=*extended expir-date addr* ,RETPD=*retn-period addr*	*exper-date addr*: RX-type address or register (2) - (12) *extended expir-date addr*: RX-type address or register (2) - (12) *retn-period addr*: RX-type address or register (2) - (12)
,ACCLVL=*access value addr* ,ACCLVL=(*access value addr,parm list addr*)	*access value addr*: RX-type address or register (2) - (12) *parm list addr*: RX-type address or register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,SECLVL=*addr*	*addr*: RX-type address or register (2) - (12)
,SECLABL=*addr*	*addr*: RX-type address or register (2) - (12)
,ERASE=YES ,ERASE=NO	**Default**: ERASE=NO
,NOTIFY=*notify-id addr*	*notify-id addr*: RX-type address or register (2) - (12)
,ENVIR=VERIFY	Specifies that only verification is to be done. **Default**: Normal RACROUTE REQUEST=DEFINE processing
,RESOWN=*resource owner addr*	*resource owner addr*: RX-type address or Register (2) - (12)
,STORCLA=*storage class addr*	*storage class addr*: RX-type address or Register (2) - (12)
,MGMTCLA=*management type addr*	*management type addr*: RX-type address or Register (2) - (12)
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=DEFINE macro, with the following exception:

,MF=(E,*ctrl addr***)**
specifies the execute form of the RACROUTE REQUEST=DEFINE macro using a remote, control-program parameter list.

,RELEASE=(*number***,CHECK)**
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=DEFINE macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

RACROUTE REQUEST = DEFINE (Modify Form)

The modify form of the RACROUTE REQUEST = DEFINE macro is written as follows:

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

REQUEST = DEFINE	
,ENTITY = *profile name addr* ,ENTITYX = *extended profile name addr*	*profile name addr*: RX-type address *extended profile name addr*: RX-type address or register (2) - (12) **Note:** ENTITY or ENTITYX must be specified on either the list or the execute form of the macro.
,VOLSER = *vol addr*	*vol addr*: RX-type address or register (2) - (12) **Note:** VOLSER is required only for CLASS = DATASET and DSTYPE not equal to M when a discrete profile name is used.
,TYPE = DEFINE ,TYPE = DEFINE,NEWNAME = *new resource name addr* ,TYPE = DEFINE,NEWNAMX = *extended new resource name addr* ,TYPE = ADDVOL,OLDVOL = *old vol addr* ,TYPE = CHGVOL,OLDVOL = *old vol addr* ,TYPE = DELETE	**Default**: TYPE = DEFINE *new resource name addr*: RX-type address or register (2) - (12) *extended new resource name addr*: RX-type address or register (2) - (12) *old vol addr*: RX-type address or register (2) - (12)
,DSTYPE = N ,DSTYPE = V ,DSTYPE = M ,DSTYPE = T	**Default**: DSTYPE = N
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12) **Default**: CLASS = DATASET
,MENTITY = *entity addr* ,MENTX = *extended entity addr*	*entity addr*: RX-type address or register (2) - (12) *extended entity addr*: RX-type address or register (2) - (12)
,MCLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12) **Default**: MCLASS = DATASET
,MVOLSER = *volser addr*	*volser addr*: RX-type address or register (2) - (12)
,MGENER = ASIS ,MGENER = YES	**Default**: MGENER = ASIS
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,UNIT = *unit addr*	*unit addr*: RX-type address or register (2) - (12)
,OWNER = *owner id addr*	*owner id addr*: RX-type address or register (2) - (12)
,LEVEL = *number* ,LEVEL = *reg*	**Default**: LEVEL = zero *reg*: register (2) - (12)

,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=EXECUTE ,UACC=NONE ,UACC=*reg*	*reg*: register (2) - (12)
,DATA=*data addr*	*data addr*: RX-type address or register (2) - (12)
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value (access level),audit value(access level)*) ,AUDIT=*reg*	*audit value*: ALL, SUCCESS, or FAILURES *access level:* READ, UPDATE, CONTROL, or ALTER **Default**: AUDIT=READ *reg*: register (2) - (12)
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default**: CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default**: GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default**: WARNING=NO **Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,FILESEQ=*number* ,FILESEQ=*reg*	*number*: 1-9999 *reg*: register (2) - (12)
,EXPDT=*exper-date addr* ,EXPDTX=*extended expir-date addr* ,RETPD=*retn-period addr*	*exper-date addr*: RX-type address or register (2) - (12) *extended expir-date addr*: RX-type address or register (2) - (12) *retn-period addr*: RX-type address or register (2) - (12)
,ACCLVL=*access value addr* ,ACCLVL=(*access value addr,parm list addr*)	*access value addr*: RX-type address or register (2) - (12) *parm list addr*: RX-type address or register (2) - (12)
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default**: TAPELBL=STD
,SECLVL=*addr*	*addr*: RX-type address or register (2) - (12)
,SECLABL=*addr*	*addr*: RX-type address or register (2) - (12)
,ERASE=YES ,ERASE=NO	**Default**: ERASE=NO
,NOTIFY=*notify-id addr*	*notify-id addr*: RX-type address or register (2) - (12)
,ENVIR=VERIFY	Specifies that only verification is to be done. **Default**: Normal RACROUTE REQUEST=DEFINE processing.
,RESOWN=*resource owner addr*	*resource owner addr*: RX-type address or register (2) - (12)
,STORCLA=*storage class addr*	*storage class addr*: RX-type address or register (2) - (12)
,MGMTCLA=*management type addr*	*management type addr*: RX-type address or register (2) - (12)
,MF=(M,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=DEFINE macro, with the following exception:

,MF=(M,*ctrl addr***)**
specifies the modify form of the RACROUTE REQUEST=DEFINE macro using a remote, control-program parameter list.

,RELEASE=(*number***,CHECK)**
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACROUTE REQUEST=DEFINE macro can be done by your specifying the CHECK subparameter on the modify form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the modify form of the macro, the modify form of the macro will not be done. Instead, a return code X'64' will be generated.

RACROUTE REQUEST = DIRAUTH — Checks Messages (for RACF Release 1.9)

The RACROUTE REQUEST = DIRAUTH macro works on behalf of the message transmission managers (VTAM, TSO, and Session Manager) to ensure that the receiver of a message meets Mandatory Access Checking (MAC) requirements. That is, the SECLABEL of the receiver of the message must dominate (be equal to or higher) than the SECLABEL of the sender.

All parameter lists generated by the RACROUTE REQUEST = DIRAUTH macro are in a format that allows compiled code to be moved above 16 megabytes virtual storage without recompilation.

Note: This request is SRB compatible.

The standard form of the RACROUTE REQUEST = DIRAUTH macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = DIRAUTH	
,RTOKEN = *message token addr*	*message token addr*: A-type address or register (2) - (12)
,LOG = ASIS ,LOG = NOFAIL	**Default** = ASIS

The parameters are explained as follows:

,RTOKEN = *message token addr*
specifies the address of the token (RTOKEN) of a resource. The RTOKEN data contains the user token (UTOKEN) of the creator of the resource. If the first two bytes (length and version) are equal to 0, it is the same as not specifying the RTOKEN.

,LOG = ASIS
,LOG = NOFAIL
specifies the types of access attempts to the DIRAUTH resource class that RACF is to record on the SMF data set.

- ASIS - RACF records the event in the manner specified on the SETR LOGOPTIONS command for the DIRAUTH resource class.
- NOFAIL - If the authorization check fails, RACF does not record the attempt. If the authorization check succeeds, RACF records the attempt as it does in ASIS.

Return Codes and Reason Codes

When control is returned, the first word of the parameter list mapped by ICHSAFP contains the return code for the DIRAUTH function; the second word contains the reason code.

When you execute the macro, space for the return code and reason code is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping by loading the ICHSAFP pointer with the label that you specified on the execute form of the macro.

Hexadecimal Code	Meaning
00	Receiver is authorized to view the message. Register 0 contains one of the following reason codes: 00 Function completed successfully 04 RTOKEN passed belongs to an operator
04	DIRAUTH cannot make a decision. Register 0 contains one of the following reason codes: 00 DIRAUTH class not active or ACEE does not contain TOKEN information 08 The definition of the provided security label was not found 0C The translation of the security label to its defining security level and categories failed 10 The SECLABEL general resource class was not active/RACLISTed 14 No defining security level exists in the SECLABEL profile FF An unexpected error occured while checking mandatory access.
08	Receiver is not authorized to view the message. Register 0 contains one of the following reason codes: 00 The security label in the user's ACEE does not currently dominate that of the message; however, the user does possess a security label that can dominate that of the message. 04 The user's security label does not dominate that of the message, and the user does not possess a security label that ever will.
0C	Invalid parameters passed to DIRAUTH. Register 0 contains one of the following reason codes: 00 The resource token (RTOKEN) was not specified.

Example 1

Operation: Invoke the RACROUTE REQUEST=DIRAUTH macro on behalf of the VTAM resource manager to perform mandatory access checking (MAC) in the "receiving" user's address space to ensure that the receiver's SECLABEL dominates that of the sender. Specify that RACF should audit the event as the DIRAUTH CDT entry indicates. The SUSBYS and REQSTOR parameters are not represented in the router table; therefore, you must specify DECOUPL=YES.

Note: The message cannot be received by anyone other than the person to whom it was directed.

```
RACROUTE  REQUEST=DIRAUTH,WORKA=RACWK,SUBSYS=VTAM,      X
          REQSTOR=TPUTRECV,DECOUPL=YES,RTOKEN(8),       X
          LOG=ASIS,RELEASE=1.9
  .
  .
RACWK     DS  CL512
```

RACROUTE REQUEST = DIRAUTH (List Form)

The list form of the RACROUTE REQUEST = DIRAUTH macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = DIRAUTH	
,RTOKEN = *message token addr*	*message token addr*: A-type address
,LOG = ASIS ,LOG = NOFAIL	**Default**: LOG = ASIS
,MF = L	

The parameters are explained under the standard form of the RACROUTE REQUEST = DIRAUTH macro with the following exception:

,MF = L
specifies the list form of the RACROUTE REQUEST = DIRAUTH macro.

RACROUTE REQUEST = DIRAUTH (Execute Form)

The execute form of the RACROUTE REQUEST = DIRAUTH macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = DIRAUTH	
,RTOKEN = *message token addr*	*message token addr*: RX-type address or register (2) - (12)
,LOG = ASIS ,LOG = NOFAIL	**Default**: LOG = ASIS
,MF = (E,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST = DIRAUTH macro with the following exception:

,MF = (E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST = DIRAUTH macro.

RACROUTE REQUEST = DIRAUTH (Modify Form)

The modify form of the RACROUTE REQUEST = DIRAUTH macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = DIRAUTH	
,RTOKEN = *message token addr*	*message token addr*: RX-type address or register (2) - (12)
,LOG = ASIS ,LOG = NOFAIL	**Default**: LOG = ASIS
,MF = (M,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST = DIRAUTH macro with the following exception:

,MF = (M,*ctrl addr*)
specifies the modify form of the RACROUTE REQUEST = DIRAUTH macro.

RACROUTE REQUEST = EXTRACT — Replace or Retrieve Fields (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see the following:

- **"RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435**
- **"RACXTRT — Retrieve Fields from RACF User Profile (for RACF Release 1.8.1 or earlier)" on page 615.** (IBM recommends that you use RACROUTE with the REQUEST = EXTRACT parameter rather than RACXTRT.)

The RACROUTE REQUEST = EXTRACT macro is used to retrieve or replace certain specified fields from a RACF profile or to encrypt certain clear-text (readable) data.

This macro is only available to authorized callers (APF-authorized, in system key 0-7, or in supervisor state). If you specify BRANCH = YES the requested function is SRB compatible.

Notes:

1. Encryption, replacement, and extraction are mutually exclusive.
2. The area returned by a RACROUTE REQUEST = EXTRACT or EXTRACTN request will be located below 16-megabytes.

PROGRAMMING INTERFACES

ONLY the following REQUEST = EXTRACT functions are general-use programming interfaces:

- Retrieving or updating fields in the TSO segment in the user profile
- Retrieving or updating fields in the user and data set profiles
- Retrieving or updating the following installation-reserved fields:
  - USERDATA
  - USRCNT
  - USRDATA
  - USRFLG
  - USRNM

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

ONLY the following REQUEST = EXTRACT functions are product-sensitive programming interfaces:

- Retrieving or updating fields in the base segment

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

The standard form of the RACROUTE REQUEST=EXTRACT macro is written as follows:

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

REQUEST=EXTRACT

,TYPE=EXTRACT
,TYPE=EXTRACTN
,TYPE=REPLACE
,TYPE=ENCRYPT

,ENTITY=*profile name addr* ,ENTITYX=*extended profile name addr*	*profile name addr*: A-type address or register (2) - (12) *extended profile name addr*: A-type address or register (2) - (12)
,RELEASE=*number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,ACEE=*acee addr*	*acee addr*: A-type address or register (2) - (12)
,VOLSER=*vol addr*	*vol addr*: A-type address or register (2) - (12)
,GENERIC=ASIS ,GENERIC=YES	**Default**: GENERIC=ASIS
,FLDACC=YES ,FLDACC=NO	**Default**: FLDACC=NO

If TYPE=EXTRACT or EXTRACTN is specified:

,SUBPOOL=*subpool number*	*subpool number*: Decimal digit 0-255 **Default**: SUBPOOL=229
,DERIVE=YES	See explanation of keyword. **Default**: Normal processing
,CLASS='*class name*' ,CLASS=*class name addr*	*class name*: 1-8 character name **Default**: CLASS=USER *class name addr*: A-type address or register (2) - (12)
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*segment name*: 1-8 character name *segment name addr*: A-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr*: A-type address or register (2) - (12)
,MATCHGN=YES ,MATCHGN=NO	**Default**: MATCHGN=NO
,BRANCH=YES ,BRANCH=NO	**Default**: BRANCH=NO

If TYPE=REPLACE is specified:

,CLASS='*class name*'	*class name*: 1-8 character name **Default**: CLASS=USER
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*segment name*: 1-8 character name *segment name addr*: A-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr*: A-type address or register (2) - (12)

,SEGDATA=*segment data addr* *segment data addr*: A-type address or register (2) - (12)

If TYPE=ENCRYPT is specified:

,ENCRYPT=(*data addr*,DES)
,ENCRYPT=(*data addr*,HASH)
,ENCRYPT=(*data addr*,INST)
,ENCRYPT=(*data addr*,STDDES) *data addr*: A-type address or register (2) - (12)

,BRANCH=YES
,BRANCH=NO **Default**: BRANCH=NO

Note: If TYPE=ENCRYPT is specified, the only other allowable parameters are ENTITY, ENTITYX, RELEASE, ENCRYPT, and BRANCH with ENCRYPT being required.

The parameters are explained as follows:

,TYPE=EXTRACT
,TYPE=EXTRACTN
,TYPE=REPLACE
,TYPE=ENCRYPT
specifies the function to be performed by the extract function routine.

- ,TYPE=EXTRACT

  With Release 1.8 and later, RACROUTE REQUEST=EXTRACT can provide additional function: it can extract information from any field in any profile. The profile templates in Chapter 3 of the *SPL: RACF* define the type and name of each field in each profile. If you specify EXTRACT, the macro extracts information from the profile determined by the ENTITY or ENTITYX and CLASS keywords. Specifically, RACF extracts the fields specified in the FIELDS keyword from the segment specified by the SEGMENT keyword. If you do not specify ENTITY or ENTITYX, RACF retrieves the desired information from the current user's profile.

  To use TYPE=EXTRACT to extract field information from a profile, you must specify Release=1.8 or a later release number.

  Note: If you specify TYPE=EXTRACT, do not specify ENCRYPT.

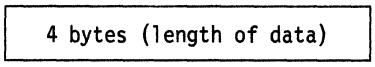
  Upon return, register 1 contains the address of a result area that begins with a fullword containing the area's subpool number and length. It is your responsibility to issue a FREEMAIN to release the area after you are through using it. See description of SUBPOOL keyword.

  The fields in the result area are in the order below:

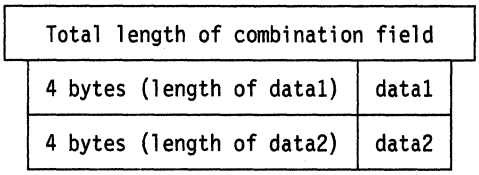
Offset (Dec)	Data	Length (Dec)
0	subpool of area	1
1	length of area	3
4	offset to start of optional field to contain segment data	2
6	reserved	18
24	specified or current user's userid, if CLASS=USER	8
32	specified user's default connect group or current user's current connect group, if CLASS=USER	8

In general, RACF returns field data in the order it was specified, with a four byte length field preceding each profile field. For example, if you were extracting the following:

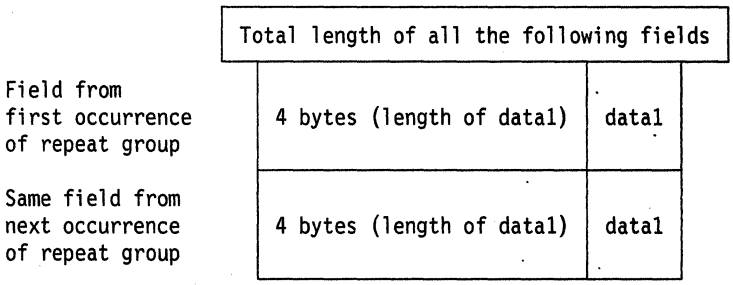
- Single field
  - A 4-byte length field that contains the length of the field that follows.
  - If the requested field is a variable length field, there is not an additional length byte.



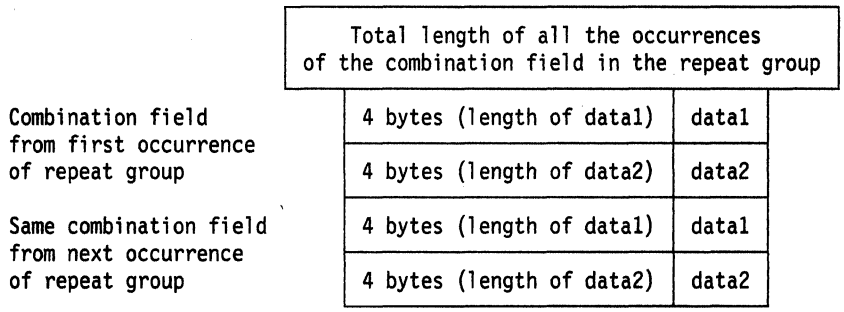
- Combination field (representing one or more fields) - You would receive:
  - A 4-byte length field that contains the combined length of all the fields that follow.
  - A combination field made up of 4-byte length fields followed by their respective individual data fields.



- Single field within a repeat group - You would receive:
  - A 4-byte length field that contains the combined length of all the fields that follow.
  - A 4-byte length field that indicates the length of the specified field in the first occurrence of the repeat group. This is followed by a 4-byte length field that indicates the length of the specified field in the second occurrence of the repeat group. And so on, until all the occurrences of the repeat group are accounted for.



- Combination field (representing one or more fields) within a repeat group - You would receive:
  - A 4-byte length field that contains the combined length of all the fields that follow.
  - A combination field consisting of a 4-byte length field indicating the length of the individual data field that follows it, followed by the next 4-byte length field indicating the length of the next individual data field. And so on, until all the individual fields that make up the combination field are accounted for. We then move on to the next occurrence of the repeat group and begin again.



  - Specifying the name of a repeat group count field retrieves only the 4-byte length followed by the 4-byte repeat group count.

    When a field to be extracted is empty, the following results:

    - For fixed length fields - RACF returns the default as specified by the template definitions. The default for flag fields is X'00'. The default for fixed length fields in the BASE segment of the profile is binary 1(s). The default for fixed length fields in other segments is binary zeros.
    - For variable length fields - RACF returns a length of zero and no data.

    If CLASS=USER, when you specify EXTRACT, the macro extracts the userid, connect group and, optionally, the encrypted password from the user profile.

- ,TYPE=EXTRACTN specifies the function to be performed by the EXTRACT function routine.

  Note: If you specify TYPE=EXTRACTN, do not specify ENCRYPT=.

  Upon return, register 1 contains the address of a result area that begins with a fullword containing the area's subpool number and length. To see the format of the result area, see the explanation of TYPE=EXTRACT above.

  If you specify EXTRACTN, the macro extracts information from the profile that follows the profile determined by the ENTITY or ENTITYX and CLASS keywords. From that next profile, RACF extracts the fields specified in the FIELDS keyword from the segment specified by the SEGMENT keyword. In addition, RACF returns the name of the profile from which it extracted the data.

- ,TYPE=REPLACE specifies the function to be performed by the EXTRACT function routine.

  Note: If you specify TYPE=REPLACE, do not specify ENCRYPT=.

  Use of the REPLACE option to update a profile requires a thorough knowledge of the inter-relationships of fields within a profile and the potential relationships between profiles. For instance, if you use RACROUTE REQUEST=EXTRACT to update a password, you should also update the password change date and password history information.

  If you specify TYPE=REPLACE, RACF takes the information in the fields specified in the FIELDS parameter and pointed to by SEGDATA, and places that information in the designated SEGMENT. (The SEGMENT is within the profile determined by the ENTITY or ENTITYX and CLASS keywords.) If you do not specify ENTITY or ENTITYX, RACF returns an error code. If you specify TYPE=REPLACE, you must specify FIELDS and SEGDATA=. If you want to replace a SEGMENT other than the BASE segment, you must specify the SEGMENT keyword with the segment you want. If you do not specify SEGMENT, the segment defaults to the BASE segment.

  if you want to create a TSO segment, you can do so by specifying the RACROUTE REQUEST=EXTRACT macro in the following way:

  TYPE=REPLACE SEGMENT=TSO

- ,TYPE=ENCRYPT specifies the function to be performed by the extract function routine.

  If TYPE=ENCRYPT is specified, the operation performed is data encryption. The ENCRYPT keyword specifies the data to be encrypted and the encryption method used. The first eight bytes of the area pointed to by the ENTITY or ENTITYX operand will be used by the DES (Data Encryption Standard) encryption routine. If ENTITY or ENTITYX is not specified, the userid from the current ACEE will be used instead. If TYPE=ENCRYPT is specified, no work area will be returned.

,ENTITY=*profile name addr*
,ENTITYX=*extended profile name addr*
specifies the address:

- ,ENTITY=*profile name addr*

  For Release 1.7 or earlier (limited to USER), specifies the address of an area eight bytes long that contains the resource name (USERID for CLASS=USER) for which profile data is to be extracted, or the userid to be used when encrypting. The name must be left-justified in the field and padded with blanks. If this parameter is not specified, a default value of zero will indicate that RACF should use the userid from the current ACEE.

  For Release 1.8 and later, specifies the address of a resource name for which profile data is to be extracted or replaced for TYPE=EXTRACT, EXTRACTN, or REPLACE, or the data to be encrypted when TYPE=ENCRYPT with DES is specified. The area is 8 bytes long for USER and GROUP; 17 bytes long for CLASS=CONNECT; and 44 bytes long for DATASET. The lengths of all other profile names are determined by the class descriptor table. The name must be left-justified in the field and padded with blanks. If this parameter is not specified, a default value of zero indicates to RACF to use the userid from the current ACEE.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39 characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

- ,ENTITYX=*extended profile name addr*
  specifies the address of a structure that consists of two, 2-byte length fields, followed by the entity name.
  - The first 2-byte field specifies a buffer length which can be from 0 to 255 bytes. This length field refers to the length of the buffer that contains the entity name; it does not include the length of either length field.
  - The second 2-byte field specifies the actual length of the entity name. This length field includes the length of the actual name without any trailing blanks; it does not include the length of either length field.

  These two length fields can be used in several different ways:
  - If the length of the entity name is known, you can specify 0 in the first field and specify the length of the entity name in the second field. Note that when you specify the second field, each byte counts; this means the entity name specified must match exactly the entity name on the RACF data base.
  - If you choose to use a buffer area in which to place the entity name, you can specify the first field to designate the length of the buffer. In regard to the second field, you can do one of two things:
    - If you know the length of the entity name, you would specify the length in the second field. (Note that the length of the first field can be from 0 to 255, but must be equal to or more than the length of the second field.)
    - If you do not know the length of the entity name, you would specify 0 in the second field, in which case RACF would be responsible for counting the number of characters in the entity name.

  Note: If your RACF installation is not using the restructured data base format, and the length of an entity name for a general resource class is longer than 39

characters, RACF uses generic profiles to match the name. This is similar to specifying RACFIND=NO.

Recommendation

IBM recommends that you use ENTITYX rather than ENTITY for two reasons:

- With ENTITYX, if you know the length of the entity name, ENTITYX allows you to pass that information to RACF. Doing so can result in slightly faster processing.
- With ENTITY, the entity name you pass to RACF must be in a buffer, the size of which is determined by the length in the CDT. If the MAXLNTH of a class increases in the future, you would have to modify your program to use a larger buffer. By using ENTITYX, you avoid this possible problem because you have removed the CDT dependency from your program.

,RELEASE=*number*

specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=EXTRACT macro can be done by specifying the CHECK subparameter on the execute form of the macro.

,ACEE=*acee addr*

specifies an alternate ACEE for RACF to use rather than the current ACEE. For example, if the ENTITY or ENTITYX parameter has not been specified, RACF refers to the ACEE during extract processing of user data.

,VOLSER=*volser addr* **(only valid with,CLASS=DATASET)**

specifies the volume serial as follows:

- For non-VSAM DASD data sets and tape data sets, specifies the volume serial number of the volume on which the data set resides.
- For VSAM DASD data sets and tape data sets, specifies the volume serial number of the catalog controlling the data set.

The field pointed to by the volser address contains the volume serial number. If necessary, you must pad it to the right with blanks so it contain six characters.

,GENERIC=ASIS
,GENERIC=YES

When CLASS is DATASET, specifies whether RACF is to treat the entity name as a generic profile name.

- If you specify GENERIC=YES, RACF considers the entity name a generic profile name, even if it does not contain any of the generic characters (an asterisk, percent sign, or ampersand).
- If you specify GENERIC=ASIS, RACF considers the entity name a generic only if it contains one or both of the generic characters.

,FLDACC=NO
,FLDACC=YES

Specifies whether field level access checking should be performed. If you specify FLDACC=YES, the RACF data base manager will check to see that the user running your program has the authority to extract or modify the fields that have been specified in the RACROUTE REQUEST=EXTRACT macro.

Notes:

1. For field level access checking to occur, you must specify RELEASE=1.8 or later when you code the macro. In addition, before the program executes, the security administrator must activate the FIELD class. If these conditions are not satisfied, the RACF manager behaves as though you had specified FLDACC=NO.
2. In addition, the security administrator must issue the RDEFINE and PERMIT commands to designate those users who will have the authority to access the fields designated in the RACROUTE REQUEST=EXTRACT macro.

3. If you specify FLDACC=NO or omit the parameter, the manager ignores field level access checking.

,SUBPOOL=*subpool number*
specifies the storage subpool from which the extract function routine obtains an area needed for the extraction. If this parameter is not specified, it defaults to 229. Note that all storage is obtained in the key of the caller. All the rules that apply to subpools and key and all their attributes are documented in the *MVS/ESA Application Development Guide*.

Note: Take some care in selecting a subpool. If you select a fetch-protected subpool or subpool 0, this may result in the program's not being able to access or free the retrieved data.

,DERIVE=YES
specifies that the desired field will be obtained from the DFP segment of the appropriate profile.

DERIVE requests are limited to the DFP segment of the DATASET and USER profiles. The following is an explanation of the DERIVE processing for both a DATASET and USER request.

- DATASET

  Specifying the DERIVE=YES keyword with CLASS=DATASET and FIELDS=RESOWNER causes RACF to perform additional processing other than simply extracting the data set resource owner from the data set profile.

  DFP uses this retrieved information for authority checking when allocating a new data set.

  To process the request, RACF first attempts to extract the RESOWNER field from the DATASET profile specified by the ENTITY or ENTITYX keyword. If the profile exists and the RESOWNER field contains data, RACF checks to see if that data is the userid of a USER or GROUP currently defined to RACF. If so, RACF returns that userid along with a reason code which indicates whether the userid is that of a USER or GROUP.

  If RACF does not find a profile that matches the DATASET name specified by the ENTITY or ENTITYX keyword, RACF attempts to locate the generic DATASET profile that protects that DATASET name.

  If it finds the generic profile, and the RESOWNER field contains data, RACF checks to see if that data is the userid of a USER or GROUP currently defined to RACF. If so, RACF returns that userid along with a reason code which indicates whether the userid is that of a USER or GROUP.

  If RACF does not find a generic profile or the retrieved data is neither a USER or GROUP, RACF returns the high-level qualifier from the name specified on the ENTITY or ENTITYX keyword along with a reason code which indicates if that high-level qualifier matches a defined USER or GROUP, or neither.

  You would specify a DERIVE request for RESOWNER as follows:

```
RACROUTE Request= EXTRACT,
ENTITY=data set name,
VOLSER=mydasd,
CLASS=DATASET,
FIELDS='RESOWNER', SEGMENT='DFP',
DERIVE=YES, RELEASE=1.9
```

  Note: You must specify all the keywords in the example for the DERIVE request to work.

- USER

  The purpose of specifying the DERIVE=YES keyword with CLASS=USER is to obtain the desired DFP field information (STORCLAS or MGMTCLAS) from the profile of the user. If the user's profile does not contain the desired DFP fields, RACF then goes to the user's default group and attempts to obtain the information

for the remaining fields from the GROUP profile (the remaining fields being those that did not contain information in the USER profile.)

You would specify a DERIVE request for information from a USER profile as follows:

```
RACROUTE Request= EXTRACT,
ENTITY=user name,
CLASS=USER,
FIELDS='STORCLAS', SEGMENT='DFP',
DERIVE=YES, RELEASE=1.9
```

RACF only processes the DERIVE keyword if it is specified with the DATASET or USER class. In addition, for DERIVE processing to occur, SEGMENT=DFP must also be specified.

,CLASS= *'class name'*
specifies the class the entity is in. The class name can be USER, GROUP, CONNECT, DATASET, or any general resource class defined in the class descriptor table.

,SEGMENT= *'segment name'*
,SEGMENT= *segment name add*
specifies the RACF profile segment that RACF is to update or from which it is to extract data. Segment-name-address is a fullword. If you specify SEGMENT, you must also specify the CLASS and FIELDS keywords. If you allow the SEGMENT parameter to default, RACF assumes that you want to extract information from the BASE segment.

,FIELDS= *field addr*
Specifies the address of a variable length list. The first field is a 4-byte field that contains the number of profile field names in the list that follows. Each profile field name is eight bytes long, left-justified, and padded to the right with blanks. The allowable field names for each type of profile are in the template listings in Chapter 3 of the *SPL: RACF*. To see how to specify the FIELDS keyword, see the TYPE=REPLACE example below.

The following options exist:

- The count field can contain numbers from 1 - 255.
- The field names can be any of the field names in the template listings.

If you specify TYPE=EXTRACT or EXTRACTN, RACF retrieves the contents of the named fields from the RACF profile indicated by the CLASS= and ENTITY= or ENTITYX= parameters, and returns the contents in the result area. (See the EXTRACT keyword for an explanation of the result area.)

You can specify TYPE=REPLACE. RACF replaces or creates the indicated fields in the profile specified on the CLASS and ENTITY or ENTITYX keywords with the data pointed to by the SEGDATA keyword.

Notes:

1. Do not replace a repeat group count field. Doing so will cause unpredictable results.
2. You cannot replace an entire repeat group, a single occurrence of a repeat group, or a single existing field in a repeat group. If you attempt to do so, RACF adds the data to the existing repeat group(s).

   The only thing you can do is retrieve all occurrences of specified fields within a repeat group or add a new occurrence of a repeat group.
3. If you add occurrences of a repeat group, RACF places those additions at the beginning (front) of the repeat group.

The following example of TYPE=REPLACE replaces fields in the BASE segment. It shows one way to code the macro and the declares necessary to make the macro work.

```
RACROUTE REQUEST=EXTRACT,TYPE=REPLACE,
         CLASS='USER',
         ENTITY=USERID,
         FIELDS=FLDLIST,
         SEGDATA=SEGDLIST,
         SEGMENT=BASE,
         RELEASE=1.9

.............................

USERID   DC  AL1(4), C'BILL'
FLDLIST  DC  A(3)
         DC  CL8'AUTHOR'
         DC  CL8'DFLTGRP'
         DC  CL8'NAME'
SEGDLIST DC  AL4(6),CL6'JSMITH'
         DC  AL4(8),CL8'SECURITY'
         DC  AL4(11),CL11'BILL THOMAS'
```

When the replacement action takes place, the following occurs:

- 'JSMITH' will be placed in the AUTHOR field in the profile.
- 'SECURITY' will be placed in the DFLTGRP field in the profile.
- 'BILL THOMAS' will be placed in the 'NAME' field in the profile.

In this example, RACROUTE REQUEST=EXTRACT retrieves the UACC from a fully qualified generic dataset profile. RACROUTE places the information in a workarea in SUBPOOL 1.

```
RACROUTE TYPE=EXTRACT
         CLASS='DATASET',
         ENTITY=DSN,
         FIELDS=FLDS,
         GENERIC=YES,
         SUBPOOL=1,
         RELEASE=1.9
......

DSN  DC CL44'SYS1.LINKLIB'
FLDS DC A(1),
     DC CL8 'UACC'
```

,MATCHGN=YES
,MATCHGN=NO

specifies that you want to extract data from a profile that matches the name specified on the ENTITY or ENTITYX keyword. If you specify MATCHGN=YES, RACF extracts data from the discrete profile, if one exists; if a discrete profile does not exist, RACF extracts data from the best-matching generic profile.

If you specify MATCHGN=NO, RACF only extracts data from a profile (discrete or generic) that *exactly* matches the name specified on the ENTITY or ENTITYX keyword.

,BRANCH=YES
,BRANCH=NO

specifies whether you want RACF to use a branch entry.

If you specify BRANCH=YES with TYPE=EXTRACT, the call is SRB compatible, (task mode callers can also use it) and RACF only searches profiles RACLISTed to a data space. This means the following:

- The only candidates for branch entry EXTRACT are general resource profiles because they are the only profiles that can be RACLISTed. If your installation has MVS 3.1.3 installed, you can use the SETROPTS RACLIST command to effect a global listing of profiles in a data space.

- RACF can only extract the following fields of the general resource profile: UACC, AUDIT, LEVEL, GAUDIT, OWNER, INSTDATA, APPLDATA, SECLEVEL, LOGDAYS, LOGTIME, LOGZONE, NOTIFY, ACLCNT, USERID, USERACS, NUMCTGY, CATEGORY, SECLABEL, SESSKEY, SLSFLAGS, SLSLOCK, KEYDATE, KEYINTVL.
- If an ACEE is passed on branch entry EXTRACT, then RACF searches RACLISTed profiles in the following order:
  - those off the ACEE
  - those off the TCB ACEE
  - those off the ASXB ACEE
  - if the profile is not found off any ACEE, then RACF searches globally RACLISTed profiles

You can only specify BRANCH=YES with TYPE=EXTRACT or ECTRACTN, or TYPE=ENCRYPT. Specifying BRANCH=YES with TYPE=ENCRYPT results in a call which is SRB compatible; there is no change in function.

,SEGDATA=*segment data addr*
specifies the address of a list of data items to be placed into the respective fields named by the FIELDS= parameter. You use the SEGDATA parameter when you specify TYPE=REPLACE. If you specify SEGDATA, you must also specify CLASS, FIELDS, and RELEASE=1.9. The stored data is paired in the following format:

- a 4-byte length field that contains the length of the data field that follows
- a data field of variable length

Each length field is followed immediately by a data field until you reach the end of the replacement data. The count field, which is pointed to by the first field in the FIELDS parameter, contains the total number of length-data pairs.

,ENCRYPT=(*data addr***,DES)**
,ENCRYPT=(*data addr***,HASH)**
,ENCRYPT=(*data addr***,INST)**
,ENCRYPT=(*data addr***,STDDES)**
specifies the data to be encrypted, and a method of encryption. The address points to a one-byte length field followed by from 1 to 255 bytes of clear-text data to be encrypted. The second subparameter specifies the encryption method: the DES algorithm, the RACF hashing algorithm, whatever scheme the installation uses (INST value), or the NBS standard DES algorithm (STDDES). Upon return to the macro issuer, the first subparameter will now contain the address of an area that contains a one-byte length followed by the encrypted version of the data. Neither the address itself nor the length is changed.

Note: When the DES or STDDES algorithm is used, RACF actually encrypts the data pointed to by the ENTITY or ENTITYX profile, (the userid if no ENTITY or ENTITYX s specified) using the data as the encryption key. Data is one-way encrypted, that is, no facility is provided to recover the data in readable form. If HASH is specified, then the RACF hashing algorithm is used and data is masked instead of encrypted.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	The extraction, replacement, or encryption completed successfully. **Reason code - For Derive requests** 0 - Some of the values are derived from the USER profile, and some may be derived from the GROUP profile. 4 - High-level qualifier returned as RESOWNER, and it matched a valid USER 8 - DFP data returned from an EXTRACT request from USER profile was actually from the user's default group. C - High-level qualifier returned as RESOWNER, and it matched a valid GROUP 24 - RESOWNER field matched a valid USER 28 - RESOWNER field matched a valid GROUP
04	An ESTAE environment was not able to be established, or if Register 0 contains a reason code of 1, neither EXTRACT, EXTRACTN, REPLACE, nor ENCRYPT was specified for TYPE=.
08	For TYPE=EXTRACT, TYPE=EXTRACTN, or TYPE=REPLACE the profile could not be found. The hexadecimal reason codes are: 0 - No profile found 4 - Field level access checking failed 8 - Segment not found (does not apply to TYPE=REPLACE) C - Class not RACLISTed (branch EXTRACT) 10 - Class not active (branch EXTRACT) 14 - Neither the RESOWNER field nor the high level qualifier matched a valid USER or GROUP
C	RACF is inactive.
10	The extract operation failed. Register 0 contains the RACF manager return code which caused termination. This return code is not used for the encrypt function. The manager return code and reason codes are returned in the low order and high order halfwords of R0.
14	An ACEE address was not found when required, or if found, was not for a defined user. The hexadecimal reason codes are: 0 - No ACEE exists 4 - ACEERACF bit is off

18 A parameter list error was encountered. The hexadecimal reason codes are:

- 4 - No fields for request type REPLACE
- 8 - Invalid type specified
- C - Invalid number of fields
- 10 - Invalid class name specified
- 14 - Invalid version in parameter list
- 18 - Invalid subpool specified
- 1C - Invalid parameter length
- 20 - No segdata specified
- 24 - Invalid entity name specified
- 2C - No encryption data
- 30 - Invalid encryption method
- 34 - No ENTITY specified with TYPE=REPLACE or TYPE=EXTRACTN
- 38 - Multiple profiles no volume specified
- 3C - Profile found wrong volser specified
- 40 - No encryption key supplied for DES or STDDES

64 Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=EXTRACT macro; however, the list form of the macro does not have the proper RELEASE parameter. It also indicates that the TYPE parameters specified on the list and execute forms may not be the same TYPE. Macro processing terminates.

Example 1

Operation: The following is an example of a RACROUTE REQUEST=EXTRACT which looks for an APPCLU profile that has been RACLISTed off the VTAMACEE to match the entity named in LULUPAIR. Specifying BRANCH=YES means that the function will not invoke an SVC. Thus, it is now SRB compatible. If the function finds a profile to match the entity name, it returns the data in the fields specified in FLDLIST.

```
RACROUTE REQUEST=EXTRACT,TYPE=EXTRACT,BRANCH=YES,CLASS='APPCLU', X
         ENTITY=LULUPAIR,ACEE=VTAMACEE,SEGMENT='SESSION',       X
         FIELDS=FLDLST,MATCHGN=YES,WORKA=WORKADDR,RELEASE=1.9
```

Example 2

Operation: The following is an example of a RACROUTE REQUEST=EXTRACT which will DES encrypt the data in RACDATA using the data in SESSNKEY as the encryption key. The function will overlay the data in RANDATA with the encrypted data. Specifying BRANCH=YES means that the function will be compatible with SRB mode and will not issue any SVCs.

```
RACROUTE REQUEST=EXTRACT,TYPE=ENCRYPT,BRANCH=YES,                X
         ENTITY=RANDATA,ENCRYPT=(SESSNKEY,STDDES),               X
         WORKA=WORKADDR,RELEASE=1.9
```

RACROUTE REQUEST = EXTRACT (List Form)

The list form of the RACROUTE REQUEST = EXTRACT macro is written as follows:

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

REQUEST = EXTRACT

,TYPE = EXTRACT
,TYPE = EXTRACTN
,TYPE = REPLACE
,TYPE = ENCRYPT

,ENTITY = *profile name addr*	*profile name addr*: A-type address.
,ENTITYX = *extended profile name addr*	*extended profile name addr*: A-type address
,RELEASE = *number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ACEE = *acee addr*	*acee addr*: A-type address or register (2) - (12)
,VOLSER = *vol addr*	*vol addr*: A-type address
,GENERIC = ASIS ,GENERIC = ONLY	**Default:** GENERIC = ASIS
,FLDACC = YES ,FLDACC = NO	**Default:** FLDACC = NO

,MF = L

If TYPE = EXTRACT or EXTRACTN is specified:

,SUBPOOL = *subpool number*	*subpool number*: Decimal digit 0-255 **Default:** SUBPOOL = 229
,DERIVE = **YES**	See explanation of keyword. **Default:** Normal processing
,CLASS = *'class name'*	*class name*: 1-8 character name **Default:** CLASS = USER
,CLASS = *class name addr*	*class name addr*: A-type address or register (2) - (12)
,SEGMENT = *'segment name'* ,SEGMENT = *segment name addr*	*segment name*: 1-8 character name *segment name addr*: A-type address or register (2) - (12)
,FIELDS = *field addr*	*field addr*: A-type address

If TYPE = REPLACE is specified:

,CLASS = *'class name'*	*class name*: 1-8 character name **Default:** CLASS = USER
,SEGMENT = *segment name addr*	*segment name addr*: A-type address or register (2) - (12)
,FIELDS = *field addr*	*field addr*: A-type address or register (2) - (12)

,MATCHGN=YES ,MATCHGN=NO	**Default**: MATCHGN=NO
,BRANCH=YES ,BRANCH=NO	**Default**: BRANCH=NO
,SEGDATA=*segment data addr*	*segment data addr*: A-type address or register (2) - (12)

If TYPE=ENCRYPT is specified:

,ENCRYPT=(*data addr*,DES) ,ENCRYPT=(*data addr*,HASH) ,ENCRYPT=(*data addr*,INST) ,ENCRYPT=(*data addr*,STDDES)	*data addr*: A-type address
,BRANCH=YES ,BRANCH=NO	**Default**: BRANCH=NO

The parameters are explained under the standard form of the RACROUTE REQUEST=EXTRACT macro with the following exception:

,MF=L
specifies the list form of the RACROUTE REQUEST=EXTRACT macro.

RACROUTE REQUEST = EXTRACT (Execute Form)

The execute form of the RACROUTE REQUEST = EXTRACT macro is written as follows:

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

REQUEST = EXTRACT	
,TYPE = EXTRACT ,TYPE = EXTRACTN ,TYPE = REPLACE ,TYPE = ENCRYPT	
,ENTITY = *profile name addr*	*profile name addr*: RX-type address or register (2) - (12)
,ENTITYX = *extended profile name addr*	*extended profile name addr*: RX-type address or register (2) - (12)
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,VOLSER = *vol addr*	*vol addr*: RX-type address or register (2) - (12)
,GENERIC = ASIS ,GENERIC = ONLY	**Default**: GENERIC = ASIS
,FLDACC = YES ,FLDACC = NO	**Default**: FLDACC = NO
,MF = (E,*ctrl addr*)	*ctrl addr*: RX-type address register (1), or register (2) - (12)
If TYPE = EXTRACT or EXTRACTN is specified:	
,SUBPOOL = *subpool number*	*subpool number*: Decimal digit 0-255 **Default**: SUBPOOL = 229
,DERIVE = YES	See explanation of keyword. **Default**: Normal processing
,CLASS = *'class name'*	*class name*: 1-8 character name **Default**: CLASS = USER
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,SEGMENT = *segment name addr*	*segment name addr*: RX-type address or register (2) - (12)
,FIELDS = *field addr*	*field addr*: RX-type address or register (2) - (12)
,MATCHGN = YES ,MATCHGN = NO	**Default**: MATCHGN = NO
,BRANCH = YES ,BRANCH = NO	**Default**: BRANCH = NO
If TYPE = REPLACE is specified:	
,CLASS = *class name addr*	*class name addr*: 1-8 character name **Default**: CLASS = USER

,SEGMENT=*'segment name'*	*segment name*: 1-8 character name
,SEGMENT=*'segment name addr'*	*segment name addr*: RX-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr*: RX-type address or register (2) - (12)
,SEGDATA=*segment data addr*	*segment data addr*: RX-type address or register (2) - (12)

If TYPE=ENCRYPT is specified:

,ENCRYPT=(*data addr*,DES) ,ENCRYPT=(*data addr*,HASH) ,ENCRYPT=(*data addr*,INST) ,ENCRYPT=(*data addr*,STDDES)	*data addr*: RX-type address or register (2) - (12)
,BRANCH=YES ,BRANCH=NO	**Default**: BRANCH=NO

The parameters are explained under the standard form of the RACROUTE REQUEST=EXTRACT macro with the following exception:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST=EXTRACT macro using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=EXTRACT macro can be done by specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE REQUEST = EXTRACT (Modify Form)

The modify form of the RACROUTE REQUEST = EXTRACT macro is written as follows:

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

REQUEST = EXTRACT	
,TYPE = EXTRACT ,TYPE = EXTRACTN ,TYPE = REPLACE ,TYPE = ENCRYPT	
,ENTITY = *profile name addr*	*profile name addr*: RX-type address or register (2) - (12)
,ENTITYX = *extended profile name addr*	*extended profile name addr*: RX-type address or register (2) - (12)
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,VOLSER = *vol addr*	*vol addr*: RX-type address or register (2) - (12)
,GENERIC = ASIS ,GENERIC = ONLY	**Default:** GENERIC = ASIS
,FLDACC = YES ,FLDACC = NO	**Default:** FLDACC = NO
,MF = (M,*ctrl addr*)	*ctrl addr*: RX-type address register (1), or register (2) - (12)
If TYPE = EXTRACT or EXTRACTN is specified:	
,SUBPOOL = *subpool number*	*subpool number*: Decimal digit 0-255 **Default:** SUBPOOL = 229
,DERIVE = YES	See explanation of keyword. **Default:** Normal processing
,CLASS = '*class name*'	*class name*: 1-8 character name **Default:** CLASS = USER
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,SEGMENT = *segment name addr* ,FIELDS = *field addr*	*segment name addr*: RX-type address or register (2) - (12) *field addr*: RX-type address or register (2) - (12)
,MATCHGN = YES ,MATCHGN = NO	**Default:** MATCHGN = NO
,BRANCH = YES ,BRANCH = NO	**Default:** BRANCH = NO
If TYPE = REPLACE is specified:	
,CLASS = '*class name*'	*class name*: 1-8 character name **Default:** CLASS = USER

,SEGMENT='*segment name*'	*segment name*: 1-8 character name
,SEGMENT=*segment name addr*	*segment name addr*: RX-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr*: RX-type address or register (2) - (12)
,SEGDATA=*segment data addr*	*segment data addr*: RX-type address or register (2) - (12)

If TYPE=ENCRYPT is specified:

,ENCRYPT=(*data addr*,DES) ,ENCRYPT=(*data addr*,HASH) ,ENCRYPT=(*data addr*,INST) ,ENCRYPT=(*data addr*,STDDES)	*data addr*: RX-type address or register (2) - (12)
,BRANCH=YES ,BRANCH=NO	**Default**: BRANCH=NO

The parameters are explained under the standard form of the RACROUTE REQUEST=EXTRACT macro with the following exception:

,MF=(M,*ctrl addr*)
specifies the modify form of the RACROUTE REQUEST=EXTRACT macro using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACROUTE REQUEST=EXTRACT macro can be done by specifying the CHECK subparameter on the modify form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the modify form of the macro, the modify form of the macro will not be done.

RACROUTE REQUEST = FASTAUTH — Verifies Access to Resources (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see the following:

- **"RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435**
- **"FRACHECK - Check User's Authorization (for RACF Release 1.8.1 or earlier)" on page 243.** (IBM recommends that you use RACROUTE with the REQUEST = FASTAUTH parameter rather than FRACHECK.)

The RACROUTE REQUEST = FASTAUTH macro is used to check a user's authorization for access to a resource. RACROUTE REQUEST = FASTAUTH verifies access to those resources whose RACF profiles have been brought into main storage by the RACROUTE REQUEST = RACLIST facility. RACROUTE REQUEST = FASTAUTH is a branch entered service that does not save registers upon entry. Registers 0-5, 14, and 15 are used by the RACROUTE REQUEST = FASTAUTH macro and are not restored. Registers 6-13 are not altered by RACROUTE REQUEST = FASTAUTH.

CAUTION:
The RACROUTE REQUEST = FASTAUTH macro executes in the addressing mode of the caller. Therefore, to access profiles that reside above 16 megabytes, the program that issues RACROUTE REQUEST = FASTAUTH must be running in 31-bit addressing mode when it issues RACROUTE REQUEST = FASTAUTH.

The standard form of the RACROUTE REQUEST = FASTAUTH macro is written as follows:

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

REQUEST = FASTAUTH	
,ENTITY = *entity addr*	*entity addr*: A-type address or register (2) - (12)
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: DASDVOL or TAPEVOL *class name addr*: A-type address or register (2) - (12)
,ATTR = READ ,ATTR = UPDATE ,ATTR = CONTROL ,ATTR = ALTER ,ATTR = *reg*	**Default:** ATTR = READ *reg*: registers (2) - (12)
,ACEE = *acee addr*	*acee addr*: A-type address or register (2) - (12)
,WKAREA = *area addr*	*area addr*: A-type address or register (2) - (12)
,APPL = *'applname'* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: A-type address or register (2) - (12)
,INSTLN = *parm list addr*	*parm list addr*: A-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6

The parameters are explained as follows:

,ENTITY = *profile name addr*
specifies that RACF authorization checking is to be performed for the resource whose name is pointed to by the specified address. The resource name is a 6-byte volume serial number for CLASS = DASDVOL or CLASS = TAPEVOL. The name must be left justified and padded with blanks. The length of all other resource names is determined from the class descriptor tables.

,CLASS = *'class name'*
,CLASS = *class name addr*
specifies that RACF authorization checking is to be performed for a resource of the specified class. If an address is specified, the address must point to an 8-byte field containing the classname.

,ATTR = READ
,ATTR = UPDATE
,ATTR = CONTROL
,ATTR = ALTER
,ATTR = *reg*
specifies the access authority required by the user or group accessing the resource:

READ — RACF user or group can open the resource only to read.

UPDATE — RACF user or group can open the resource to read or write.

CONTROL — For VSAM data sets, RACF user or group has authority equivalent to the VSAM control password. For non-VSAM data sets and other resources, RACF user or group has UPDATE authority.

ALTER — RACF user or group has total control over the resource.

If a register is specified, the register must contain one of the following codes in the low-order byte of the register:

X'02' — READ
X'04' — UPDATE
X'08' — CONTROL
X'80' — ALTER

,ACEE = *acee addr*
specifies the address of the accessor control environment element (ACEE) to be used to check authorization and to locate the in-storage profiles (REQUEST = LIST) output) for the specified classes. If an ACEE is specified, it is used for authorization checking. If the specified ACEE has an in-storage profile list for the specified class, it is used to locate the resource. If an ACEE is not specified or if there is no in-storage profile list for the specified class in the ACEE, RACF uses the TASK ACEE (TCBSENV) pointer in the extended TCB. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE for the address space to obtain the list of the in-storage profiles. The main ACEE is pointed to by the ASXBSENV field of the address space extension block.

,WKAREA = *area addr*
specifies the address of a 16-word work area to be used by RACROUTE REQUEST = FASTAUTH which contains the following information:

Word 12 contains the reason code that ICHRFC00 will pass back to the RACROUTE REQUEST = FASTAUTH caller via Register 0.

Word 13 contains the return code that RACROUTE REQUEST = FASTAUTH passes back to the caller in register 15.

Word 14 contains the address of the in-storage profile used to determine authorization, or zero if no profile was found.

Word 15 contains a value provided by a pre-processing installation exit, or zero if there was no pre-processing exit. This will be passed back to the caller in register 1.

,APPL = *'applname'*
,APPL = *applname addr*
specifies the name of the application requesting the authorization checking. This information is not used for the authorization checking process but is made available to the installation exit(s). If an address is specified, it should point to an 8-byte area containing the application name, left justified and padded with blanks, if necessary.

,INSTLN = *parm list addr*
specifies the address of an area that contains information for the RACROUTE REQUEST=FASTAUTH installation exit. This address is passed to the exit routine when the exit is given control. The INSTLN parameter is used by application or installation programs to pass information to the RACROUTE REQUEST=FASTAUTH installation exit.

,RELEASE = *number*
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=FASTAUTH macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	The user or group is authorized to use the resource.

Reason Code	Meaning
0	The RACROUTE REQUEST=FASTAUTH return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is not within the scope of the audit/global audit specification.
4	The RACROUTE REQUEST=FASTAUTH return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is within the scope of the audit/global audit specification. The RACROUTE REQUEST=FASTAUTH caller should log the attempt by issuing a RACROUTE REQUEST=AUTH for the resource that the caller is attempting to access.

04	The resource or classname is not defined to RACF.
08	The user or group is not authorized to use the resource.

Reason Code	Meaning
0	The RACROUTE REQUEST=FASTAUTH return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is not within the scope of the audit/global audit specification.
4	The RACROUTE REQUEST=FASTAUTH return code indicates if the caller is authorized or not authorized to the resource, but the access attempt is within the scope of the audit/global audit specification. The RACROUTE REQUEST=FASTAUTH caller should log the attempt by issuing a RACROUTE REQUEST=AUTH for the resource that the caller is attempting to access.

0C	RACF is not active.
10	RACROUTE REQUEST=FASTAUTH installation exit error occurred.

14	RACF CVT does not exist (RACF is not installed or insufficient level of RACF is installed).
24	Indicates the profile has a conditional access list, the port-of-entry field in the security token is blank-filled, and the port-of-entry class is active.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=FASTAUTH macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

RACROUTE REQUEST = FASTAUTH (List Form)

The list form of the RACROUTE REQUEST = FASTAUTH macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = FASTAUTH	
,ENTITY = *entity addr*	*entity addr*: A-type address
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: DASDVOL or TAPEVOL. *class name addr*: A-type address
,ATTR = READ ,ATTR = UPDATE ,ATTR = CONTROL ,ATTR = ALTER	**Default:** ATTR = READ
,ACEE = *acee addr*	*acee addr*: A-type address
,WKAREA = *area addr*	*area addr*: A-type address
,APPL = *'applname'* ,APPL = *applname addr*	*applname addr*: A-type address
,INSTLN = *parm list addr*	*parm list addr*: A-type address
,RELEASE = *number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,MF = L	

The parameters are explained under the standard form of the RACROUTE REQUEST = FASTAUTH macro, with the following exception:

,MF = L
specifies the list form of the RACROUTE REQUEST = FASTAUTH macro.

RACROUTE REQUEST = FASTAUTH (Execute Form)

The execute form of the RACROUTE REQUEST = FASTAUTH macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE. REQUEST = FASTAUTH.
REQUEST = FASTAUTH	
,ENTITY = *entity addr*	*entity addr*: RX-type address or register (2) - (12)
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,ATTR = READ ,ATTR = UPDATE ,ATTR = CONTROL ,ATTR = ALTER ,ATTR = *reg*	*reg*: register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,WKAREA = *area addr*	*area addr*: RX-type address or register (2) - (12)
,APPL = *applname addr*	*applname addr*: RX-type address or register (2) - (12)
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, 1.6 **Default:** RELEASE = 1.6
,MF = (E,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST = FASTAUTH macro, with the following exception:

,MF = (E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST = FASTAUTH macro, using a remote control program parameter list.

,RELEASE = (*number*,CHECK)
,RELEASE = *number*
,RELEASE = (,CHECK)
specifies the RACF release level of the parameter list to be generated by the macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST = FASTAUTH macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE REQUEST = LIST — Build In-Storage Profiles (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see the following:

- **"RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435**
- **"RACLIST — Build In-Storage Profiles (for RACF Release 1.8.1 or earlier)" on page 429.** (IBM recommends that you use RACROUTE with the REQUEST = LIST parameter rather than RACLIST.)

RACROUTE REQUEST = LIST is used to build in-storage profiles for RACF defined resources. RACROUTE REQUEST = LIST processes only those resources described by class descriptors. The primary advantage of using the RACROUTE REQUEST = LIST macro is to use the resource grouping function and to improve resource authorization checking performance.

The module calling the RACROUTE REQUEST = LIST macro must either be authorized (APF-authorized, in system key 0-7, or in supervisor state) or re-entrant in the RACF-authorized caller table and fetched from an authorized library.

Note: If the ACEE is below 16-megabytes, any area, with the exception of generic profiles, chained off an ACEE (for example, RACLIST profiles, list-of-groups table) will be placed below 16-megabytes. Otherwise, the area will be placed above the line. However, a caller executing in 31-bit mode may issue a REQUEST = LIST with LOC = ABOVE, and the profiles will be placed above 16-megabytes, if possible, even if the ACEE is below 16-megabytes.

The standard form of the RACROUTE REQUEST = LIST macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b RACROUTE	One or more blanks must precede RACROUTE.
b	One or more blanks must follow RACROUTE.
REQUEST = LIST	
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: A-type address or register (2) - (12)
,LIST = *list addr*	*list addr*: A-type address or register (2) - (12)
,FILTER = *filter addr*	*filter addr*: A-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: A-type address or register (2) - (12)
,INSTLN = *parm list addr*	*parm list addr*: A-type address or register (2) - (12)
,APPL = *'applname'* ,APPL = *applname addr*	*applname* 1-8 character name *applname addr*: A-type address or register (2) - (12)
,SUBPOOL = (*sub#1,sub#2*)	*sub#1,sub#2*: Decimal digit 0-255
,ENVIR = CREATE ,ENVIR = DELETE	**Default**: ENVIR = CREATE
,OWNER = YES ,OWNER = NO	**Default**: OWNER = NO
,LOC = BELOW ,LOC = ANY ,LOC = ABOVE	**Default**: See parameter description **Note:** LOC can be coded only if REQUEST = VERIFY or REQUEST = LIST is coded.
,RELEASE = *number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6

The parameters are explained as follows:

CLASS = *'class name'*
CLASS = *class name addr*
specifies that RACROUTE REQUEST = LIST is to build an in-storage profile for the resources of the specified class. If an address is specified, the address must point to an 8-byte field containing the class name, left justified and padded with blanks, if necessary. The class name must be defined by a class descriptor; if not, the class is not considered to be defined.

,LIST = *addr*
specifies the address of a list of resource names for which RACROUTE REQUEST = LIST is to build the in-storage profiles. The list consists of a 2-byte field containing the number of the names in the list, followed by one or more variable length names. Each name consists of a 1-byte length field, which is the length of the name, followed by the name. A zero in the 2-byte field causes the operand to be omitted. If LIST= is omitted, in-storage profiles are built for all the profiles defined to RACF in the given class as well as each member for a resource grouping associated with the specified class.

Note: This operand can be specified only with ENVIR = CREATE. If ENVIR = DELETE is specified, the RACROUTE REQUEST = LIST macro issues a return code of 18.

,FILTER = *filter addr*
specifies the address of a generic filter string which RACF uses to search the RACF data set and select profile names for which RACROUTE REQUEST = LIST will build in-storage profiles. The filter consists of a 2-byte length field followed by the filter string. The filter string length must not exceed the length of the profile name as it is specified in the class descriptor table.

The following generic characters have special meaning when used as part of the filter string:

- % - (1 Character in a name)

  You can use the percent sign to represent any **one** character in the profile name, including a generic character. For example, if you specify DASD%% as a filter string, it can represent profile names such as DASD01, DASD2A, and DASD%5. If you specify %%%%% as a filter string, it can represent profile names such as DASD1, DASD2, DASD%, TAPE%, MY%%%, TAPE* and %%%%*.

- * - (0 - n characters in a qualifier)

  You can use a single asterisk to represent any **zero or more characters in a qualifier**, including generic characters. For example, AF*.CD can represent profile names such as AF.CD, ABEF.CD, and ABX.CD. A single asterisk can also represent an entire qualifier. For example, ABC.* represents profile names such as ABC.D, AFC.DEF, ABC.%%%, and ABC.%/DE. RACF generic profiles do not allow a * in the high level qualifier; however, the FILTER operand does allow it.

- ** - (0 - n qualifiers in a name)

  You can use a double asterisk to represent **zero or more qualifiers** in the profile name. For example, AB.**.CD represents profile names such as AB.CD, AB.DE.EF.CD, and AB.XYZ.CD. You can also specify ** as the only characters in the filter-string to represent any entire profile name. You cannot specify other characters with ** within a qualifier. For example, you can specify USER1.**, but not USER1.A**.

Notes:

1. You cannot specify FILTER with LIST on the same RACLIST invocation because the two keywords are mutually exclusive.
2. You can only specify the FILTER keyword with ENVIR = CREATE. If you specify ENVIR = DELETE, RACLIST returns a return code of 18.

,ACEE = *acee addr*

specifies the address of the accessor control environment element (ACEE). The ACEE points to the in-storage profiles. If an ACEE is not specified, RACF uses the TASK ACEE pointer in the extended TCB called the TCBSENV. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE to obtain the list of the in-storage profiles. The main ACEE is pointed to by the ASXBSENV field of the address space extension block. If an ACEE is not specified and there is no main ACEE, the in-storage profiles are not constructed.

,INSTLN = *parm list addr*

specifies the address of an area that contains parameter information for the RACLIST installation exit. The address is passed to the installation exit when the exit is given control by the RACLIST routine. The INSTLN parameter can be used by an application or an installation program to pass information to the RACLIST installation exit.

,APPL = *'applname'*
,APPL = *applname addr*

specifies the name of the application requesting the authorization checking. This information is not used for the authorization checking process but is made available to the installation exit(s). If an address is specified, it should point to an 8-byte area containing the application name, left justified and padded with blanks, if necessary.

,SUBPOOL = (*sub#1,sub#2*)

specifies the subpool numbers of the storage into which the components of the in-storage profiles are to be built. Sub#1 represents the subpool of the profile index. Sub#2 represents the subpool of the profile proper. If the subpools are not specified they default to subpool 255. Registers can be used to specify sub#1 and sub#2.

,ENVIR = CREATE
,ENVIR = DELETE

specifies the action to be performed by the RACROUTE REQUEST = LIST macro.

CREATE - In-storage profiles for the specified class are to be built. The RACROUTE REQUEST = LIST function issues a return code of 18, if an in-storage list currently exists for the specified class.

DELETE - The in-storage profiles for the specified class are to be freed. If class is not specified, the in-storage profiles for all classes are freed.

Note: It is the responsibility of the user issuing the RACROUTE REQUEST = LIST macro to assure that no multi-tasking that results in the issuing of a RACROUTE REQUEST = AUTH, RACROUTE REQUEST = FASTAUTH, RACROUTE REQUEST = VERIFY, RACROUTE REQUEST = LIST macro occurs at the same time that the RACROUTE REQUEST = LIST occurs.

,OWNER = YES
,OWNER = NO

specifies that the resource owner is to be placed in the profile access list with the ALTER authority. If the OWNER = operand is omitted, the default is NO.

,LOC = BELOW
,LOC = ANY
,LOC = ABOVE

LOC can be coded only if REQUEST = VERIFY or REQUEST = LIST is coded.

For REQUEST = VERIFY:
specifies whether the ACEE and related data areas are to be allocated storage below 16 megabytes (LOC = BELOW), or anywhere (LOC = ANY).

If any of the following is true, LOC=BELOW is the default, and LOC=ANY is ignored if specified:

- The ACEE= parameter is not coded.
- The caller is executing in 24-bit mode.

In all other cases, the default is LOC=ANY.

Note: LOC=ABOVE is invalid for REQUEST=VERIFY.

For REQUEST=LIST:
specifies whether the RACROUTE REQUEST=LIST profiles are to reside where the ACEE is located, above or below 16 megabytes (LOC=ANY), or whether the RACROUTE REQUEST=LIST profiles are to reside above 16 megabytes (LOC=ABOVE), if possible, even if the ACEE is below 16 megabytes.

Notes:

1. LOC=BELOW is invalid for RACROUTE REQUEST=LIST.
2. LOC=ANY does not guarantee that storage is allocated above 16 megabytes. If any installation SAF or RACF exit routines execute in 24-bit mode, the storage will be below 16 megabytes.

,RELEASE=*number*
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=LIST macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	RACROUTE REQUEST=LIST function completed successfully.
04	Unable to perform the requested function. Register 0 contains additional codes as follows: 0 - Unable to establish an ESTAE environment. 1 - The function code (the third byte of the parameter list) does not represent a valid function. '01' represents the RACF manager; '02' represents the RACROUTE REQUEST=LIST macro.
08	The specified class is not defined to RACF.
0C	An error was encountered during RACROUTE REQUEST=LIST processing.
10	RACF and/or the resource class is not active.
14	RACLIST installation exit error occurred.
18	Parameter list error. Register 0 contains additional codes as follows: 0 - No ACEE found 4 - Class already RACLISTed 8 - Invalid name length in list of names C - LIST or FILTER specified on delete request 10 - Invalid request type (not DEFINE or DELETE) 14 - LIST and FILTER specified (they are mutually exclusive)
20	Invalid filter sequence
1C	RACF CVT does not exist (RACF is not installed) or an insufficient level of RACF is installed.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=LIST macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Note: If the resource class specified by the CLASS= operand is inactive, RACROUTE REQUEST=LIST does not build the in-storage profiles and a code of 0C is returned. If the resource group class is not active, RACROUTE REQUEST=LIST builds an in-storage profile but only from the individual resource profiles; resource group profiles are ignored.

Example 1

Operation: Use the standard form of the macro to build in-storage profiles for all the profiles in the class named CLASSNAM, and chain them off the ACEE whose address is pointed to by ACEEADDR.

```
RACROUTE REQUEST=LIST,CLASS=CLASSNAM,ACEE=ACEEADDR,ENVIR=CREATE,
         RELEASE=1.9
```

Example 2

Operation: Use the standard form of the macro to build in-storage profiles for all the profiles whose names are in a list named PROFLIST and a class named CLASSNAM. Chain them from the task ACEE or address space ACEE.

```
RACROUTE REQUEST=LIST,CLASS=CLASSNAM,LIST=PROFLIST,ENVIR=CREATE,
         RELEASE=1.9
```

Example 3

Operation: Use the standard form of the macro to delete the in-storage profiles for the CLASSNAM class.

```
RACROUTE REQUEST=LIST,CLASS=CLASSNAM,ENVIR=DELETE,RELEASE=1.9
```

RACROUTE REQUEST = LIST (List Form)

The list form of the RACROUTE REQUEST = LIST macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = LIST	
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: 1-8 character name *class name addr*: A-type address
,LIST = *list addr*	*list addr*: A-type address
,FILTER = *filter addr*	*filter addr*: A-type address
,ACEE = *acee addr*	*acee addr*: A-type address
,INSTLN = *parm list addr*	*parm list addr*: A-type address
,APPL = *'applname'* ,APPL = *applname addr*	*applname addr*: A-type address
,SUBPOOL = (*sub#1,sub#2*)	*sub#1,sub#2*: Decimal digit 0-255 **Default**: 255.
,ENVIR = CREATE ,ENVIR = DELETE	**Default**: ENVIR = CREATE
,OWNER = YES ,OWNER = NO	**Default**: OWNER = NO
,LOC = BELOW ,LOC = ANY ,LOC = ABOVE	**Default**: See parameter description **Note:** LOC can be coded only if REQUEST = VERIFY or REQUEST = LIST is coded.
,RELEASE = *number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,MF = L	

The parameters are explained under the standard form of the RACROUTE REQUEST = LIST macro with the following exception:

,MF = L
specifies the list form of the RACROUTE REQUEST = LIST macro.

RACROUTE REQUEST = LIST (Execute Form)

The execute form of the RACROUTE REQUEST = LIST macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = LIST	
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,LIST = *list addr*	*list addr*: RX-type address or register (2) - (12)
,FILTER = *filter addr*	*filter addr*: RX-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,APPL = *appIname addr*	*appIname addr*: RX-type address or register (2) - (12)
,SUBPOOL = (*sub#1*,*sub#2*)	*sub#1*,*sub#2*: Decimal digit 0-255
,ENVIR = CREATE ,ENVIR = DELETE	
,OWNER = YES ,OWNER = NO	
,LOC = BELOW ,LOC = ANY ,LOC = ABOVE	**Default**: See parameter description **Note:** LOC can be coded only if REQUEST = VERIFY or REQUEST = LIST is coded.
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,MF = (E,,*ctrl addr*)	*ctrl addr*: RX-type address or register (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST = LIST macro with the following exception:

,MF = (E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST = LIST macro using a remote control program parameter list.

,RELEASE = (*number*,CHECK)
,RELEASE = *number*
,RELEASE = (,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST = LIST macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE REQUEST = LIST (Modify Form)

The modify form of the RACROUTE REQUEST = LIST macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = LIST	
,CLASS = *class name addr*	*class name addr*: RX-type address or register (2) - (12)
,LIST = *list addr*	*list addr*: RX-type address or register (2) - (12)
,FILTER = *filter addr*	*filter addr*: RX-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,APPL = *applname addr*	*applname addr*: RX-type address or register (2) - (12)
,SUBPOOL = (*sub#1*,*sub#2*)	*sub#1*,*sub#2*: Decimal digit 0-255
,ENVIR = CREATE ,ENVIR = DELETE	
,OWNER = YES ,OWNER = NO	
,LOC = BELOW ,LOC = ANY ,LOC = ABOVE	**Default**: See parameter description **Note:** LOC can be coded only if REQUEST = VERIFY or REQUEST = LIST is coded.
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE = 1.6
,MF = (M,*ctrl addr*)	*ctrl addr*: RX-type address or register (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST = LIST macro with the following exception:

,MF = (M,*ctrl addr*)
specifies the modify form of the RACROUTE REQUEST = LIST macro using a remote control program parameter list.

,RELEASE = (*number*,CHECK)
,RELEASE = *number*
,RELEASE = (,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACROUTE REQUEST = LIST macro can be done by your specifying the CHECK subparameter on the modify form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the modify form of the macro, the modify form of the macro will not be done.

RACROUTE REQUEST = STAT - Determine RACF Status (for RACF Release 1.9)

This macro description applies to RACF Release 1.9. Do not attempt to execute a program with new Release 1.9 functions until you have installed RACF Release 1.9. If you have RACF Release 1.8.1 or earlier installed on your system, see "RACSTAT - Determines the Status of RACF (for RACF Release 1.8.1 or earlier)" on page 607.

The RACROUTE REQUEST = STAT macro is used to determine if RACF is active and optionally determine if RACF protection is in effect for a given resource class. The RACROUTE REQUEST = STAT macro can also be used to determine if a resource class name is defined to RACF.

RACROUTE REQUEST = STAT is a branch entered service that uses standard linkage conventions.

To use this service, you must also specify RELEASE = 1.9.

The standard form of the RACROUTE REQUEST = STAT macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = STAT	
,CLASS = *'class name'* ,CLASS = *class name addr*	*class name*: DATASET, DASDVOL, or TAPEVOL, or any class defined in the RACF class descriptor table *class name addr*: A-type address or register (2) - (12)
,ENTRY = *entry addr*	*entry addr*: A-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9

The parameters are explained as follows:

,CLASS = *'class name'*
,CLASS = *class name addr*
specifies the classname for which RACF authorization checking is performed. The name can be explicitly defined on the macro by enclosing the name in quotes. If specified, the address must point to an 8-byte field containing the classname, left justified and padded with blanks if necessary. If CLASS = is omitted, the status of RACF is returned.

,ENTRY = *entry addr*
specifies the address of a 4-byte area that is set to the address of the specified class in the RACF class descriptor table. This operand is ignored when the CLASS = operand is omitted.

,RELEASE = *number*
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST = STAT macro can be done by specifying the CHECK subparameter on the execute form of the macro.

Return Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	RACF is active and, if CLASS= was specified, the class is active.
04	RACF is active; the class is inactive.
08	RACF is active; the class is not defined to RACF.
0C	RACF is inactive and, if CLASS= was specified, the class is active.
10	RACF is inactive; the class is inactive.
14	RACF is inactive; the class is not defined to RACF.
18	RACF CVT does not exist (RACF is not installed) or an insufficient level of RACF is installed.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=STAT macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Note: The class descriptor entry for the specified class is returned to the caller (in the 4-byte area addressed by the entry addr), for return codes 00, 04, 0C, and 10.

Example 1

Operation: Determine if the DASDVOL class is active and retrieve the address of its class descriptor. A fullword, CDADDR, contains the class descriptor address.

```
RACROUTE REQUEST=STAT,CLASS='DASDVOL',ENTRY=CDADDR,RELEASE=1.9
```

RACROUTE REQUEST=STAT (List Form)

The list form of the RACROUTE REQUEST=STAT macro is written as follows:

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

REQUEST=STAT	
,CLASS='*class name*' ,CLASS=*class name addr*	*class name*: DATASET, DASDVOL, or TAPEVOL. *class name addr*: A-type address
,ENTRY=*entry addr*	*entry addr*: A-type address
,RELEASE=*number*	*number*: 1.9
,MF=L	

The parameters are explained under the standard form of the RACROUTE REQUEST=STAT macro with the following exception:

,MF=L
specifies the list form of the RACROUTE REQUEST=STAT macro.

RACROUTE REQUEST=STAT (Execute Form)

The execute form of the RACROUTE REQUEST=STAT macro is written as follows:

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

REQUEST=STAT	
,CLASS='*class name*' ,CLASS=*class name addr*	*class name*: DATASET, DASDVOL, or TAPEVOL. *class name addr*: RX-type address or register (2) - (12)
,ENTRY=*entry addr*	*entry addr*: RX-type address or register (2) - (12)
,RELEASE=(*number*CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=STAT macro, with the following exception:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST=STAT macro, using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACSTAT macro can be done by specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE REQUEST=STAT (Modify Form)

The modify form of the RACROUTE REQUEST=STAT macro is written as follows:

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

REQUEST=STAT	
,CLASS='*class name*' ,CLASS=*class name addr*	*class name*: DATASET, DASDVOL, or TAPEVOL. *class name addr*: RX-type address or register (2) - (12)
,ENTRY=*entry addr*	*entry addr*: RX-type address or register (2) - (12)
,RELEASE=(*number*CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9
,MF=(M,*ctrl addr*)	*ctrl addr*: RX-type address or register (1) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=STAT macro, with the following exception:

,MF=(M,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST=STAT macro, using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACSTAT macro can be done by specifying the CHECK subparameter on the modify form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the modify form of the macro will not be done.

RACROUTE REQUEST = TOKENBLD - Modify a UTOKEN (for RACF Release 1.9)

Whereas the RACROUTE REQUEST = VERIFYX is used to build a TOKEN, TOKENBLD is used to modify an existing token. The TOKNIN keyword specifies the location of the existing token from which a modified token is to be built. Note that the modification does not change the input token; instead, the function builds a new modified token from the parameters provided. The TOKNOUT keyword specifies the location where the new modified token will be built.

The following order of priority exists when replacing the fields in the existing TOKEN:

- Keywords specified on the request
- The SUSER, SNODE, and SGROUP fields within the token specified by the STOKEN keyword.
- All fields within the token specified by the TOKNIN keyword.

Thus, if you do not want certain fields overridden, do not specify keywords for those fields.

To issue the RACROUTE REQUEST = TOKENBLD macro, the calling module must be 'authorized' which means

- APF-authorized, or
- in system key 0-7, or
- in supervisor state

To use this service, you must also specify RELEASE = 1.9.

The standard form of the RACROUTE REQUEST = TOKENBLD macro is written as follows:

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

REQUEST = TOKENBLD	
,USERID = *userid addr*	*userid addr*: A-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: A-type address or register (2) - (12) **Default**: GROUP = zero
,TERMID = *terminal addr*	*terminal addr*: A-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,TOKNIN = *input token addr*	*input token addr*: A-type address or register (2) - (12)
,TOKNOUT = *output token addr*	*output token addr*: A-type address or register (2) - (12)
,STOKEN = *stoken addr*	*stoken addr*: A-type address or register (2) - (12)
,SECLABL = *seclabel addr*	*seclabel addr*: A-type address or register (2) - (12)
,EXENODE = *execution node addr*	*execution node addr*: A-type address or register (2) - (12)
,SNODE = *submitting node addr*	*submitting node addr*: A-type address or register (2) - (12)
,SUSERID = *submitting userid addr*	*submitting userid addr*: A-type address or register (2) - (12)
,SGROUP = *submitting group addr*	*submitting group addr*: A-type address or register (2) - (12)

,POE=*port of entry addr*	*port of entry addr*: A-type address or register (2) - (12)
,SESSION=TSO	**Default**: SESSION=TSO
,SESSION=NJEBATCH	
,SESSION=RJEBATCH	
,SESSION=INTBATCH	
,SESSION=EXTBATCH	
,SESSION=XBM	
,SESSION=CONSOPER	
,SESSION=STARTED	
,SESSION=MOUNT	
,SESSION=COMMAND	
,SESSION=SYSAS	
,SESSION=NJEOPER	
,SESSION=RJEOPER	
,SESSION=NJEUNKN	
,TRUSTED=YES	
,TRUSTED=NO	**Default**: TRUSTED=NO

The parameters are explained as follows:

,USERID = *userid addr*
specifies the identification of the operator who has entered the system. The address points to a 1-byte length field, followed by the userid.

,GROUP = *group addr*
specifies the group of the user who has entered the system. The address points to a 1-byte length field, followed by the group name.

,TERMID = *terminal addr*
specifies the address of the identifier of the terminal through which the user is accessing the system. The address points to an 8-byte area containing the terminal identifier. The area must reside in a non-task-related storage subpool.
specifies the RACF release level of the parameter list to be generated by this macro.

To use the parameters associated with a release, you must specify the release number of that release or a later release number. If you specify an earlier release level, the parameter will not be accepted by macro processing, and an error message will be issued at assembly time.

When you specify the RELEASE keyword, checking is done at assembly time.

,TOKNIN = *input token addr*
specifies the address of the UTOKEN or RTOKEN that is to be used as a base for the output token.

The UTOKEN fields are mapped in ICHRUTKN in the Data Areas chapter of the *SPL: RACF.*

,TOKNOUT = *output token addr*
specifies the address of the caller-provided area for the modified token data. The first byte of TOKNOUT contains the actual token length. This provides for downward compatibility with all versions of the token map.

If you specify an STOKEN, the USERID in the STOKEN becomes the SUSER in TOKNOUT, unless you specified the SUSER keyword, in which case, that keyword becomes SUSER in TOKNOUT. In the same way, if you specified NODE in the STOKEN, that becomes the SNODE in TOKNOUT, unless you specified the SNODE keyword. Likewise, if you specified GROUP in the STOKEN, that becomes SGROUP in the TOKNOUT, unless you specified the SGROUP keyword. These are the only fields that are used from the STOKEN. In all cases, the specified keywords on the request override the fields of the STOKEN, if one is specified.

,STOKEN = *stoken addr*
specifies the address of the submittor's UTOKEN The first byte contains the length of the UTOKEN, and the second byte contains the version number. See the ICHRUTKN mapping in the Data Areas Chapter of the *SPL: RACF* for the current version and release.

If you specify an STOKEN, the USERID in the STOKEN becomes the SUSER in TOKNOUT, unless you specified the SUSER keyword, in which case, that keyword becomes SUSER in TOKNOUT. In the same way, if you specified NODE in the STOKEN, that becomes the SNODE in TOKNOUT, unless you specified the SNODE keyword. Likewise, if you specified GROUP in the STOKEN, that becomes SGOUP in the TOKNOUT, unless you specified the SGROUP keyword. These are the only fields that are used from the STOKEN. In all cases, the specified keywords on the request override the fields of the STOKEN, if one is specified.

,SECLABL = *seclabel addr*
specifies the address of an 8-byte left-justified field, padded to the right with blanks which contains the SECLABEL.

An installation would use SECLABELs to establish an association between a specific RACF security level (SECLEVEL) and a set of (zero or more) RACF security categories (CATEGORY). In a B1 system, it is necessary to use SECLABELs to prevent the unauthorized movement of data from one level to another when multiple levels of data are in use on the system at the same time. See the *RACF Security Administrator's Guide* for further information.

,EXENODE = *execution node addr*
specifies the address of an area that contains a one byte length field followed by the name of the node on which the unit of work is to be executed. The node name cannot exceed eight bytes.

,SNODE = *submitting node addr*
specifies the address of an area that contains a 1-byte length field followed by the name of the node from which the unit of work was submitted. The node name cannot exceed eight bytes.

,SUSERID = *submitting userid addr*
specifies the address of an area that contains a one byte length field followed by the userid of the user who submitted the unit of work. The userid cannot exceed eight bytes.

,SGROUP = *submitting group addr*
specifies the address of an area that contains a 1-byte length field followed by the groupid of the user who submitted the unit of work. The groupid cannot exceed eight bytes.

,POE = *port of entry addr*
specifies the address of the port of entry into the system, that is, the name of the input device through which the job was submitted.

The port of entry will be a part of the user's security token (UTOKEN). A flag in the UTOKEN uniquely identifies the RACF general resource class to which the data in the POE field belongs: TERMINAL, CONSOLE, or JESINPUT. The RACF class JESINPUT provides the conditional access support for commands/jobs entered into the system through a JES input device and the CONSOLE class performs the same task for commands/jobs that originate from a console. The TERMINAL class covers the commands/jobs that originate from a VTAM session, for example, TSO.

,SESSION = *type*
specifies the session type(s) to be associated with the request. You can specify mutiple sessions, separated by commas, if necessary. For example, SESSION=CONSOPER,SYSAS. Session types are literals.

The allowable session types are:

- SYSAS = session type is a system address space
- COMMAND = session type is a command
- TSO = session type is a TSO logon
- CONSOPER = session type is console operator
- STARTED = session type is started procedure of started task
- MOUNT = session type is mount
- XBM = session type is execution batch monitor job
- NJEBATCH = session type is batch job from Network Job Entry (NJE)
- RJEBATCH = session type is batch job from Remote Job Entry (RJE)
- INTBATCH = session type is batch job from Internal Reader (INT)
- EXTBATCH = session type is batch job from External Reader (EXT)
- NJEOPER = session type is network job entry
- NJSYSOUT = session type is network sysout
- RJEOPER = session type is remote job entry
- NJEUNKN = session type is unknown user from NJE

,TRUSTED=YES
,TRUSTED=NO

specifies whether or not the submitter of the unit of work is a member of the trusted computer base.

For further information on the trusted computer base, see the *MVS/ESA Planning: B1 Security*.

Return Codes and Reason Codes

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

Hexadecimal Code	Meaning
00	RACROUTE REQUEST=TOKENBLD has completed successfully. In addition, the following return code and reason codes have been saved in the ICHSAFP return and reason code fields: When you execute the macro, space for the return code and reason code is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping by loading the ICHSAFP pointer with the label that you specified on the execute form of the macro. **Return code: 08** Indicates TOKENBLD REQUEST successful Reason code: 10 TOKNOUT area specified was larger than expected, on return the token length field contains the expected length. Reason code: 14 STOKEN area specified was larger than expected; on return the token length field contains the expected length. Reason code: 20 TOKNIN area specified was larger than expected; on return the token length field contains the expected length.

08 The requested function could not be performed.

In addition, the following return code and reason codes have been saved in the ICHSAFP return and reason code fields:

Return code: 00. Error occurred before function could initiate.

Reason code: 00 Recovery environment could not be established.

Return code: 08 TOKENBLD REQUEST error

Reason code: 00 TOKNOUT (required) keyword missing.

Reason code: 04 TOKNOUT area was too small, on return the token length field contains the expected length

Reason code: 08 An error occurred when translating a token within TOKENBLD.

Reason code: 0C STOKEN area was too small, on return the token length field contains the expected length.

Reason code: 18 A token was specified with an undefined version.

Reason code: 1C TOKNIN area was too small, on return the token length field contains the expected length.

Example 1

Operation: The following example shows a RACROUTE REQUEST=TOKENBLD macro can be specified to replace a SECLABEL in an existing token.

```
RACROUTE REQUEST=TOKENBLD,
         SUBSYS=address of caller subsytem,
         REQSTOR=address of caller subsystem control point,
         TOKNOUT=address of caller-supplied token area,
         TOKIN=address of input token,
         SECLABEL=address of session owner SECLABEL,
         RELEASE=1.9
```

Note: Additional keywords such as WORKA, required by RACF to complete the request are specified on RACROUTE itself.

RACROUTE REQUEST=TOKENBLD (List Form)

The list form of the RACROUTE REQUEST=TOKENBLD macro is written as follows:

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

b                                        One or more blanks must precede RACROUTE.

RACROUTE

b                                        One or more blanks must follow RACROUTE.

REQUEST=TOKENBLD

   ,USERID=userid addr                   userid addr: A-type address

   ,GROUP=group addr                     group addr: A-type address

   ,TERMID=terminal addr                 terminal addr: A-type address

,RELEASE=number                          number: 1.9

   ,TOKNIN=input token addr               input token addr: A-type address

   ,TOKNOUT=output token addr            output token addr: A-type address

   ,STOKEN=stoken addr                   stoken addr: A-type address

   ,SECLABL=seclabel addr                seclabel addr: A-type address

   ,EXENODE=execution node addr          execution node addr: A-type address

   ,SNODE=submitting node addr           submitting node addr: A-type address (2) - (12)

   ,SUSERID=submitting userid addr       submitting userid addr: A-type address

   ,SGROUP=submitting group addr         submitting group addr: A-type address

   ,POE=port of entry addr               port of entry addr: A-type address

   ,SESSION=TSO                          Default: SESSION=TSO
   ,SESSION=NJEBATCH
   ,SESSION=RJEBATCH
   ,SESSION=INTBATCH
   ,SESSION=EXTBATCH
   ,SESSION=XBM
   ,SESSION=CONSOPER
   ,SESSION=STARTED
   ,SESSION=MOUNT
   ,SESSION=COMMAND
   ,SESSION=SYSAS
   ,SESSION=NJEOPER
   ,SESSION=RJEOPER
   ,SESSION=NJEUNKN

   ,TRUSTED=YES
   ,TRUSTED=NO                           Default: TRUSTED=NO

,MF=L
```

The parameters are explained under the standard form of the RACROUTE REQUEST=TOKENBLD macro, with the following exception:

,MF=L
specifies the list form of the RACROUTE REQUEST=TOKENBLD macro.

RACROUTE REQUEST = TOKENBLD (Execute Form)

The execute form of the RACROUTE REQUEST = TOKENBLD macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = TOKENBLD	
,USERID = *userid addr*	*userid addr*: RX-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: RX-type address or register (2) - (12) **Default**: GROUP = zero
,TERMID = *terminal addr*	*terminal addr*: RX-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,TOKNIN = *input token addr*	*input token addr*: RX type address or register (2) - (12)
,TOKNOUT = *output token addr*	*output token addr*: RX-type address or register (2) - (12)
,STOKEN = *stoken addr*	*stoken addr*: RX-type address or register (2) - (12)
,SECLABL = *seclabel addr*	*seclabel addr*: RX-type address or register (2) - (12)
,EXENODE = *execution node addr*	*execution node addr*: RX-type address or register (2) - (12)
,SNODE = *submitting node addr*	*submitting node addr*: RX-type address or register (2) - (12)
,SUSERID = *submitting userid addr*	*submitting userid addr*: RX-type address or register (2) - (12)
,SGROUP = *submitting group addr*	*submitting group addr*: RX-type address or register (2) - (12)
,POE = *port of entry addr*	*port of entry addr*: RX-type address or register (2) - (12)
,SESSION = TSO ,SESSION = NJEBATCH ,SESSION = RJEBATCH ,SESSION = INTBATCH ,SESSION = EXTBATCH ,SESSION = XBM ,SESSION = CONSOPER ,SESSION = STARTED ,SESSION = MOUNT ,SESSION = COMMAND ,SESSION = SYSAS ,SESSION = NJEOPER ,SESSION = RJEOPER ,SESSION = NJEUNKN	**Default**: SESSION = TSO
,TRUSTED = YES ,TRUSTED = NO	**Default**: TRUSTED = NO
,MF = (E,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=TOKENBLD macro, with the following exception:

,MF=(E,*ctrl addr***)**
specifies the execute form of the RACROUTE REQUEST=TOKENBLD macro using a remote control program parameter list.

,RELEASE=*number*
specifies the RACF release level of the parameter list to be generated by this macro.

RACROUTE REQUEST = TOKENBLD (Modify Form)

The modify form of the RACROUTE REQUEST = TOKENBLD macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must precede RACROUTE.
REQUEST = TOKENBLD	
,USERID = *userid addr*	*userid addr*: RX-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: RX-type address or register (2) - (12) **Default**: GROUP = zero
,TERMID = *terminal addr*	*terminal addr*: RX-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,TOKNIN = *input token addr*	*input token addr*: RX-type address or register (2) - (12)
,TOKNOUT = *output token addr*	*output token addr*: RX-type address or register (2) - (12)
,STOKEN = *stoken addr*	*stoken addr*: RX-type address or register (2) - (12)
,SECLABL = *seclabel addr*	*seclabel addr*: RX-type address or register (2) - (12)
,EXENODE = *execution node addr*	*execution node addr*: RX-type address or register (2) - (12)
,SNODE = *submitting node addr*	*submitting node addr*: RX-type address or register (2) - (12)
,SUSERID = *submitting userid addr*	*submitting userid addr*: RX-type address or register (2) - (12)
,SGROUP = *submitting group addr*	*submitting group addr*: RX-type address or register (2) - (12)
,POE = *port of entry addr*	*port of entry addr*: RX-type address or register (2) - (12)
,SESSION = TSO ,SESSION = NJEBATCH ,SESSION = RJEBATCH ,SESSION = INTBATCH ,SESSION = EXTBATCH ,SESSION = XBM ,SESSION = CONSOPER ,SESSION = STARTED ,SESSION = MOUNT ,SESSION = COMMAND ,SESSION = SYSAS ,SESSION = NJEOPER ,SESSION = RJEOPER ,SESSION = NJEUNKN	**Default**: SESSION = TSO
,TRUSTED = YES ,TRUSTED = NO	**Default**: TRUSTED = NO
,MF = (M,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=TOKENBLD macro, with the following exception:

,MF=(M,*ctrl addr*)
specifies the modify form of the RACROUTE REQUEST=TOKENBLD macro using a remote control program parameter list.

,RELEASE=*number*
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time.

RACROUTE REQUEST=TOKENMAP - Access Token Fields (for RACF Release 1.9)

This macro is used to map a token. REQUEST=TOKENMAP accepts a token in either internal or external format as input. Internal format refers to the format returned from a RACROUTE REQUEST=VERIFYX or a RACROUTE REQUEST=TOKENXTR. External format refers to that which is mapped by the ICHRUTKN macro. RACROUTE REQUEST=TOKENMAP is the *only* interface used to map token data.

The primary purpose of the RACROUTE REQUEST=TOKENMAP function is to allow a caller to access individual fields within the UTOKEN. The caller only needs to provide the proper length for the corresponding version of a token, and SAF/RACF will map it correctly using this macro.

To use this service, you must also specify RELEASE=1.9.

The standard form of the RACROUTE REQUEST=TOKENMAP macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST=TOKENMAP	
,TOKNIN=*input token addr*	*input token addr*: A-type address or register (2) - (12)
,TOKNOUT=*output token addr*	*output token addr*: A-type address or register (2) - (12)
,FORMOUT=INTERNAL ,FORMOUT=EXTERNAL	**Default**: FORMOUT=EXTERNAL
,RELEASE=*number*	*number*: 1.9

The parameters are explained as follows:

,TOKNIN= *input token addr*
specifies the address of the UTOKEN or RTOKEN that is to be converted to internal or external format.

The UTOKEN fields are mapped in ICHRUTKN in the Data Areas chapter of the *SPL: RACF.*

,TOKNOUT= *output token address*
specifies the address of the user provided area for the converted token data. The first byte of TOKNOUT contains the actual token length. This provides for downward compatibility with all versions of the token map.

,FORMOUT=EXTERNAL
,FORMOUT=INTERNAL
specifies the format of the output token area. Internal format is that which is returned form a RACROUTE REQUEST=VERIFYX or a RACROUTE REQUEST=TOKENXTR. External format is that which is mapped by the ICHRUTKN macro.

,RELEASE= *number*
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=TOKENBLD macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason code is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping by loading the ICHSAFP pointer with the label that you specified on the execute form of the macro.

Hexadecimal Code	Meaning
00	Reason described by the following hex reason codes

Reason Code	Meaning
0	Request was successful
4	TOKEN not converted
8	TOKEN version not defined, assume the most current
C	TOKNOUT area too large, token was successfully extracted

9C7 This ABEND code is described by the following hex reason codes

Reason Code	Meaning
4	TOKNIN required parameter missing (for tokenmap)
8	TOKNOUT required parameter missing
C	TOKNOUT area too small
10	TOKVERS=0

Example 1

Operation: The following is an example of how to invoke the TOKENMAP function:

```
RACROUTE REQUEST=TOKENMAP,
         TOKIN=address of input TOKEN to translate
         TOKNOUT=address of area for output TOKEN,
         FORMOUT=INTERNAL/EXTERNAL (format of output TOKEN
                                    default is EXTERNAL)
         MF=E (Execute)
         RELEASE=1.9
```

Note: Additional keywords specified on the RACROUTE macro are required.

Example 2

Operation: Assume that the caller is Print Services Facility (PSF) and needs to map information from the UTOKEN to determine which type of label to put on the output printed data. The caller invokes the RACROUTE REQUEST=TOKENMAP macro to convert a UTOKEN in internal format into a token in external format so PSF can access the various fields of the UTOKEN.

```
RACROUTE  REQUEST=TOKENMAP,TOKNIN=((2)),                X
          WORKA=RACWK,TOKNOUT=((5)),                    X
          RELEASE=1.9
  .
  .
RACWK     DS  CL512
```

RACROUTE REQUEST = TOKENMAP (List Form)

The list form of the RACROUTE REQUEST = TOKENMAP macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = TOKENMAP	
,TOKNIN = *input token addr*	*input token addr*: A-type address
,TOKNOUT = *output token addr*	*output token addr*: A-type address
,FORMOUT = INTERNAL ,FORMOUT = EXTERNAL	**Default**: FORMOUT = EXTERNAL
,RELEASE = *number*	*number*: 1.9
,MF = L	

The parameters are explained under the standard form of the RACROUTE REQUEST = TOKENMAP macro with the following exception:

,MF = L
specifies the list form of the RACROUTE REQUEST = TOKENMAP macro.

RACROUTE REQUEST=TOKENMAP (Execute Form)

The execute form of the RACROUTE REQUEST=TOKENMAP macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST=TOKENMAP	
,TOKNIN=*input token addr*	*input token addr*: RX-type address or register (2) - (12)
,TOKNOUT=*output token addr*	*output token addr*: RX-type address or register (2) - (12)
,FORMOUT=INTERNAL ,FORMOUT=EXTERNAL	**Default**: FORMOUT=EXTERNAL
,RELEASE=*number*	*number*: 1.9
,MF=(E,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=TOKENMAP macro with the following exception:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST=TOKENMAP macro.

RACROUTE REQUEST = TOKENMAP (Modify Form)

The modify form of the RACROUTE REQUEST = TOKENMAP macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = TOKENMAP	
,TOKNIN = *input token addr*	*input token addr*: RX-type address or register (2) - (12)
,TOKNOUT = *output token addr*	*output token addr*: RX-type address or register (2) - (12)
,FORMOUT = INTERNAL ,FORMOUT = EXTERNAL	**Default**: FORMOUT = EXTERNAL
,RELEASE = *number*	*number*: 1.9
,MF = (M,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

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

,MF = (M,*ctrl addr*)
specifies the modify form of the RACROUTE macro.

RACROUTE REQUEST=TOKENXTR - Extract UTOKENS (for RACF Release 1.9)

This macro is used to extract a UTOKEN from the current task or address space ACEE; SAF performs the extraction. Any information not available from the ACEE will be returned as blanks. The ICHRUTKN macro maps the UTOKEN.

To use this service, you must also specify RELEASE=1.9.

The standard form of the RACROUTE REQUEST=TOKENXTR macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST=TOKENXTR	
,ACEE=*acee addr*	*acee addr*: A-type address or register (2) - (12)
,TOKNOUT=*output token addr*	*output token addr*: A-type address or register (2) - (12)
,RELEASE=*number*	*number*: 1.9

The parameters are explained as follows:

,ACEE=*acee addr*
specifies the address of the ACEE from which information is to be extracted.

If you do not specify the ACEE keyword, then TOKENXTR extracts the information it needs from the TCBSENV field of the task control block if it is non-zero; if it is zero, TOKENXTR extracts information it needs from the ASXBSENV field.

,TOKNOUT=*return token addr*
specifies the address where the requestor wants TOKENXTR to return the UTOKEN that was extracted from the ACEE. The first byte of storage at the address specified must contain the number of bytes of available storage. The second byte must contain the version of the token.

The UTOKEN fields are mapped in ICHRUTKN in the Data Areas chapter of the *SPL: RACF.*

,RELEASE=*number*
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason code is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping by loading the ICHSAFP pointer with the label that you specified on the execute form of the macro.

Hexadecimal Code	Meaning
00	Reason described by the following hex reason codes

Reason Code	Meaning
0	Request was successful
4	Invalid (down level) ACEE supplied
8	No ACEE available
C	TOKNOUT area length was too large

9C7	The ABEND code is described by the following hex reason codes

Reason Code	Meaning
8	Required parameter missing
C	TOKNOUT length too small

Example 1

Operation: The following is an example of how to invoke the TOKENXTR function:

```
RACROUTE REQUEST=TOKENXTR,
         TOKNOUT=address of area for output TOKEN,
         RELEASE=1.9
```

Note: Additional keywords specified on the RACROUTE macro are required.

Example 2

Operation: Assume that the caller is Print Services Facility (PSF) and it needs to extract information from the ACEE to determine which type of label to put on output printed data.

```
RACROUTE  REQUEST=TOKENXTR,TOKNOUT=((2)),              X
          WORKA=RACWK,                                 X
          RELEASE=1.9
  .
  .
RACWK     DS  CL512
```

RACROUTE REQUEST = TOKENXTR (List Form)

The list form of the RACROUTE REQUEST = TOKENXTR macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = TOKENXTR	
,ACEE = *acee addr*	*acee addr*: A-type address
,TOKNOUT = *output token addr*	*output token addr*: A-type address
,RELEASE = *number*	*number*: 1.9
,MF = L	

The parameters are explained under the standard form of the RACROUTE REQUEST = TOKENXTR macro with the following exception:

,MF = L
specifies the list form of the RACROUTE REQUEST = TOKENXTR macro.

RACROUTE REQUEST = TOKENXTR (Execute Form)

The execute form of the RACROUTE REQUEST = TOKENXTR macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = TOKENXTR	
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,TOKNOUT = *output token addr*	*output token addr*: RX-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,MF = E	

The parameters are explained under the standard form of the RACROUTE REQUEST = TOKENXTR macro with the following exception:

,MF = E
specifies the execute form of the RACROUTE REQUEST = TOKENXTR macro.

RACROUTE REQUEST = TOKENXTR (Modify Form)

The modify form of the RACROUTE REQUEST = TOKENXTR macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST = TOKENXTR	
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,TOKNOUT = *output token addr*	*output token addr*: RX-type address or register (2) - (12)
,RELEASE = *number*	*number*: 1.9
,MF = M	

The parameters are explained under the standard form of the RACROUTE REQUEST = TOKENXTR macro with the following exception:

,MF = M
specifies the modify form of the RACROUTE REQUEST = TOKENXTR macro.

RACROUTE REQUEST = VERIFY — Identify a RACF-Defined User (for RACF Release 1.9)

If you have RACF Release 1.8.1 or earlier installed on your system, see the following:

- **"RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435**
- **"RACINIT — Identify a RACF-Defined User (for RACF Release 1.8.1 or earlier)" on page 417.** (IBM recommends that you use RACROUTE with the REQUEST = VERIFY parameter rather than RACINIT.)

The RACROUTE REQUEST = VERIFY macro is used to provide Resource Access Control Facility (RACF) user identification and verification. The macro identifies a user and verifies that the user is defined to RACF and has supplied a valid password and/or operator identification card (OIDCARD parameter).

With RACF Release 1.9, a subsystem can use REQUEST = VERIFY and the new keywords to create an ACEE. If RACF is not active, not installed, or your installation has a RACF release prior to 1.9, then SAF, using the new keywords, will build a default ACEE to satisfy the request. The purpose of this use of REQUEST = VERIFY is for job submission, user verification checking, and propagation of security information from user session to unit of work.

To issue the RACROUTE REQUEST = VERIFY macro the calling module must be "authorized" which means

- APF-authorized, or
- in system key 0-7, or
- in supervisor state.

or the NEWPASS keyword must be omitted and the calling module must:

- be in the RACF-authorized caller table *and*
- fetched from an authorized library *and*
- reentrant.

Notes:

1. It is recommended that if you run programs which issue the RACROUTE REQUEST = VERIFY macro, you run those programs AFP-authorized. See the *SPL: RACF* for important information on the authorized caller table.
2. Unless the caller specifies the ACEE= parameter on a RACROUTE REQUEST = VERIFY,ENVIR = CREATE macro, the ACEE will always be placed below 16-megabytes.
3. If the caller specifies the ACEE= parameter, and is executing in 31-bit addressing mode and does not specify LOC = BELOW on the RACROUTE macro, the ACEE will be placed, if possible, above 16-megabytes.

The standard form of the RACROUTE REQUEST=VERIFY macro is written as follows:

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

REQUEST=VERIFY	
,USERID=*userid addr*	*userid addr*: A-type address or register (2) - (12)
,PASSWRD=*password addr*	*password addr*: A-type address or register (2) - (12)
,START=*procname addr*	*procname addr*: A-type address or register (2) - (12)
,NEWPASS=*new password addr*	*new password addr*: A-type address or register (2) - (12)
,GROUP=*group addr*	*group addr*: A-type address or register (2) - (12) **Default**: GROUP=zero
,PGMNAME=*programmer name addr*	*programmer name addr*: A-type address or register (2) - (12)
,ACTINFO=*account addr*	*account addr*: A-type address or register (2) - (12)
,OIDCARD=*oid addr*	*oid addr*: A-type address or register (2) - (12)
,TERMID=*terminal addr*	*terminal addr*: A-type address or register (2) - (12)
,JOBNAME=*jobname addr*	*jobname addr*: A-type address or register (2) - (12)
,ENVIR=CREATE ,ENVIR=VERIFY ,ENVIR=CHANGE ,ENVIR=DELETE	**Default**: ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr*: A-type address or register (2) - (12)
,APPL='*applname*' ,APPL=*applname addr*	*applname*: 1-8 character name *applname addr*: A-type address or register (2) - (12)
,ACEE=*acee addr*	*acee addr*: A-type address or register (2) - (12)
,SUBPOOL=*subpool number*	*subpool number*: Decimal digit 0-255
,SMC=YES ,SMC=NO	**Default**: SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default**: PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default**: ENCRYPT=YES
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default**: See parameter description **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.

,RELEASE=*number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: A-type address or register (2) - (12)
,STOKEN=*stoken addr*	*stoken addr*: A-type address or register (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: A-type address or register (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: A-type address or register (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: A-type address or register (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: A-type address or register (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: A-type address or register (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: A-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: A-type address or register (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default:** SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default:** TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default:** REMOTE=NO

The parameters are explained as follows:

,USERID=*userid addr*
specifies the user identification of the user who has entered the system. The address points to a 1-byte length field, followed by the userid which can be up to 8 characters.

,PASSWRD=*password addr*
specifies the currently defined password of the user who has entered the system. The address points to a 1-byte length field, followed by the password which can be up to 8 characters.

,START=*procname addr*
specifies the PROC name of the started task for which the RACROUTE REQUEST=VERIFY is being performed. The address points to an 8-byte area containing the PROC name (left-justified and padded with blanks, if necessary). If START= is specified, REQUEST=VERIFY processing searches the started procedure table for the userid (and groupid) to use for this REQUEST=VERIFY request. If the USERID and GROUP keywords are specified, REQUEST=VERIFY will only use those values if it cannot obtain a userid from the started procedure table.

If START is specified, PASSWRD and OIDCARD should not be specified.

,NEWPASS = *new password addr*
specifies the password which is to replace the user's currently defined password. The address points to a 1-byte length field, followed by the password which can be up to 8 characters.

,GROUP = *group addr*
specifies the group specified by the user who has entered the system. The address points to a 1-byte length field, followed by the group name which can be up to 8 characters.

,PGMNAME = *programmer name addr*
specifies the address of the name of the user who has entered the system. This twenty byte area is passed to the RACINIT installation exit routine; it is not used by the RACINIT routine.

,ACTINFO = *account addr*
specifies the address of a field containing accounting information. This 144 byte area is passed to the RACINIT installation exit routine; it is not used by the RACINIT routine. The accounting field, if supplied, should have the following format:

- First byte of field contains the number (binary) of accounting fields.
- Following bytes contain accounting fields, where each entry for an accounting field contains a 1-byte length field, followed by the field.

,OIDCARD = *oid addr*
specifies the address of the currently defined operator identification card of the user who has entered the system. The address points to a 1-byte length field, followed by the operator ID card.

,TERMID = *terminal addr*
specifies the address of the identifier for the terminal through which the user is accessing the system. The address points to an 8-byte area containing the terminal identifier. The area must reside in a non-task-related storage subpool.

,JOBNAME = *jobname addr*
specifies the address of the JOB name of a background job. The address points to an eight byte area containing the JOB name (left justified and padded with blanks, if necessary). The JOBNAME parameter is used by RACINIT during authorization checking to verify the user's authority to submit the job. It is passed to the installation exit routine.

,ENVIR = CREATE
,ENVIR = VERIFY
,ENVIR = CHANGE
,ENVIR = DELETE
specifies the action to be performed by the user initialization component regarding the accessor environment element (ACEE). The default is CREATE.

CREATE - The user should be verified and an ACEE created.

VERIFY - Only a user verification is to be made; however, it can be optionally combined with updating the user's password. The installation can do this through a System Authorization Facility (SAF) installation exit. If the installation does not use SAF to satisfy this request, the RACROUTE caller receives a return code of 4, with RACF return and reason codes of zero.

CHANGE - The ACEE should be modified according to other parameters specified on RACROUTE REQUEST = VERIFY.

DELETE - The ACEE should be deleted. This parameter should only be used if a previous RACROUTE REQUEST = VERIFY has completed successfully.

,INSTLN = *parm list addr*
specifies the address of an area containing parameter information meaningful to the RACINIT installation exit routine. This area is passed to the installation exit when the exit routine is given control from the RACINIT routine.

The INSTLN parameter can be used by an installation having a user verification or job initiation application, and wanting to pass information from one installation module to the RACINIT installation exit routine.

,APPL = *'applname'*
,APPL = *applname addr*
specifies the name of the application issuing the RACROUTE REQUEST = VERIFY. If an address is specified, the address must point to an 8-byte application name, left justified and padded with blanks, if necessary.

,ACEE = *acee addr*
specifies the address of the ACEE.

For ENVIR = DELETE: specifies the address of a fullword that contains the address of the accessor environment element (ACEE) to be deleted. If ACEE = is not specified, and the TCBSENV field for the task using the RACROUTE REQUEST = VERIFY is non-zero, the ACEE pointed to by the TCBSENV is deleted, and TCBSENV is set to zero. If the TCBSENV and ASXBSENV fields both point to the same ACEE, then ASXBSENV is also set to zero. If no ACEE address is passed, and TCBSENV is zero, the ACEE pointed to by ASXBSENV is deleted, and ASXBSENV is set to zero.

For ENVIR = CHANGE: specifies the address of a fullword that contains the address of the accessor environment element (ACEE) to be changed. If ACEE = is not specified, and the TCBSENV field for the task using the RACROUTE REQUEST = VERIFY is non-zero, the ACEE pointed tc by the TCBSENV is changed. If TCBSENV is 0, then the ACEE pointed to by ASXBSENV is changed.

For ENVIR = CREATE: specifies the address of a full word into which the RACROUTE REQUEST = VERIFY function will place the address of the ACEE created. If an ACEE is not specified, the address of the newly created ACEE is stored in the TCBSENV field of the task control block. If the ASXBSENV field is set to binary zeros, the new ACEE address is also stored in the ASXBSENV field of the ASXB. If the ASXBSENV field is non-zero, it is not modified. The TCBSENV field is set unconditionally.

Notes:

1. If you omit USERID, GROUP, and PASSWRD and if you code ENVIR = CREATE or if ENVIR = CREATE is used as the default, you will receive a return code of X'00' and obtain an ACEE that contains an * (X'5C') in place of the USERID and group name.
2. If ACEE is specified with ENVIR = CREATE, RACF suppresses the creation of a modeling table (MDEL) to reduce the amount of CSA and/or LSQA storage required by IMS/VS and CICS/VS installations.

,SUBPOOL = *subpool number*
specifies the storage subpool from which the ACEE and related storage are obtained. The default subpool is 255.

,SMC = YES
,SMC = NO
specifies the use of the step-must-complete function of RACROUTE REQUEST = VERIFY processing. SMC = YES specifies that RACROUTE REQUEST = VERIFY processing should continue to place other tasks for the job step as non-dispatchable. SMC = NO specifies that the step-must-complete function is not used.

Note: SMC = NO should not be used if DADSM ALLOCATE/SCRATCH functions execute simultaneously in the same address space as the RACROUTE REQUEST = VERIFY function.

,PASSCHK = YES
,PASSCHK = NO
specifies whether or not the user's password is to be verified. PASSCHK = YES specifies that RACROUTE REQUEST = VERIFY verifies the user's password. PASSCHK = NO specifies that the user's password is not verified.

,ENCRYPT = YES
,ENCRYPT = NO

specifies whether or not RACROUTE REQUEST = VERIFY will encrypt the old password, the new password, and the OIDCARD data passed to it.

YES signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are not pre-encrypted. RACROUTE REQUEST = VERIFY encrypts the data before storing it in the user profile or using it to compare against stored data. ENCRYPT = YES is the default for this keyword.

NO signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are already encrypted. RACROUTE REQUEST = VERIFY bypasses the encryption of this data before storing it in, or comparing it against, the user profile.

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

Note: The exit routine ICHDEX01 can also perform the encryption.

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

,LOC = BELOW
,LOC = ANY

For REQUEST = VERIFY:
specifies whether the ACEE and related data areas are to be allocated storage below 16 megabytes (LOC = BELOW), or anywhere (LOC = ANY)

If any of the following is true, LOC = BELOW is the default, and LOC = ANY is ignored if specified:

- The ACEE = parameter is not coded.
- The caller is executing in 24-bit mode.

In all other cases, the default is LOC = ANY.

,RELEASE = *number*

specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST = VERIFY macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,STAT = ASIS
,STAT = NO

specifies whether the statistics controlled by the installation's options on the RACF SETROPTS command are to be maintained or ignored for this execution of RACROUTE REQUEST = VERIFY. This parameter also controls whether a message is to be issued when the logon is successful.

Note: Messages are always issued if the RACROUTE REQUEST = VERIFY processing is unsuccessful.

If STAT = ASIS is specified or taken by default, the messages and statistics are controlled by the installation's current options on the RACF SETROPTS command.

If STAT = NO is specified, the statistics are not updated. And, if the logon is successful, no message is issued.

The default is STAT = ASIS.

,LOG = ASIS
,LOG = ALL

specifies when log records are to be generated.

If LOG = ASIS is specified or defaulted to, only those attempts to create an ACEE that fail will generate RACF log records.

If LOG=ALL is specified, any request to create an ACEE, regardless of whether it succeeds or fails, will generate a RACF log record. The default is LOG=ASIS.

,UTOKEN= *utoken addr*
specifies the address of the UTOKEN of the user for which RACF will perform a RACROUTE REQUEST=VERIFY. The first byte contains the length of the UTOKEN, and the second byte contains the version number. The version number is 01.

The UTOKEN fields are mapped in ICHRUTKN in the Data Areas chapter of the *SPL: RACF*.

,STOKEN= *stoken addr*
specifies the address of the submittor's UTOKEN The first byte contains the length of the UTOKEN, and the second byte contains the version number. The version number is 01. See explanation of UTOKEN.

,SECLABL= *seclabel addr*
specifies the address of an 8-byte left-justified character field containing the SECLABEL.

,EXENODE= *execution node addr*
specifies the address of an area that contains a one byte length field followed by the name of the node on which the unit of work is to be executed. The node name cannot exceed eight bytes.

,SNODE= *submitting node addr*
specifies the address of an area that contains a one byte length field followed by the name of the node from which the unit of work was submitted. The node name cannot exceed eight bytes.

,SUSERID= *submitting userid addr*
specifies the address of an area that contains a one byte length field followed by the userid of the user who submitted the unit of work. The userid cannot exceed eight bytes.

,SGROUP= *submitting group addr*
specifies the address of an area that contains a one byte length field followed by the groupid of the user who submitted the unit of work. The groupid cannot exceed eight bytes.

,POE= *port of entry addr*
specifies the address of the port of entry into the system. The address points to the name of the input device through which the job was submitted. The port of entry is an 8 character field which is left-justified and padded with blanks.

The port of entry will be a part of the user's security token (UTOKEN). A flag in the UTOKEN will uniquely identify the RACF general resource class to which the data in the POE field belongs: TERMINAL, CONSOLE, or JESINPUT.

,LOGSTR= *logstr addr*
specifies the address of a one byte length field followed by character data that will be written to the SMF data set together with RACF audit information.

,SESSION= *type*
specifies the session type(s) to be associated with the request. You can specify multiple sessions, separated by commas, if necessary. For example, SESSION=CONSOPER,SYSAS. Session types are literals. When the SESSION keyword is used in combination with the POE keyword, SESSION determines the class with which the POE keyword will be connected. The default session type is TSO.

The allowable session types are:

- SYSAS = session type is a system address space
- COMMAND = session type is a command
- TSO = session type is a TSO logon
- CONSOPER = session type is console operator
- STARTED = session type is started procedure of started task
- MOUNT = session type is mount

- XBM = session type is execution batch monitor job
- BATCH = session type is batch
- NJEOPER = session type is network job entry
- RJEOPER = session type is remote job entry
- INTBATCH = session type is a job that entered system via internal reader
- EXTBATCH = session type is a job that entered system via external reader
- NJEBATCH = session type is a job that entered system via Network Job Entry
- RJEBATCH = session type is a job that entered system via Remote Job Entry

,TRUSTED=YES
,TRUSTED=NO
specifies whether or not the submitter of the unit of work is a member of the trusted computer base.

,REMOTE=YES
,REMOTE=NO
specifies whether or not the job came through the network.

Return Codes and Reason Codes

When you execute the macro, space for the return code and reason codes is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping macro by loading the ICHSAFP pointer with the label that you specified on the list form of the macro.

Hexadecimal Code	Meaning
00	RACROUTE REQUEST=VERIFY has completed successfully.
04	The user profile is not defined to RACF.
08	The password is not authorized.
0C	The password has expired.
10	The new password is invalid.
14	The user is not defined to the group.
18	RACINIT was failed by the installation exit routine.
1C	The user access has been revoked.
20	RACF is not active. For a RACROUTE, this is not an error condition; it means that the requested function could not be performed because RACF was not active.
24	The user's access to the specified group has been revoked.
28	OIDCARD parameter is required but not supplied.
2C	OIDCARD parameter is invalid for specified user.
30	The user is not authorized to the port of entry. Register 0 contains one of the following reason codes: 00 Indicates a normal completion. 04 Indicates the user is not authorized to access the system on this day, or at this time of day. 08 Indicates the port of entry may not be used on this day, or at this time of day. **Note:** The port of entry now refers to TERMINALS, the JESINPUT class, and the CONSOLE class ports of entry.
34	The user is not authorized to use the application.

38 SECLABEL checking occurred. Register 0 contains one of the following reason codes:

- 04 MLACTIVE requires a SECLABEL; none was specified
- 08 Indicates the user is not authorized to the SECLABEL
- 0C The system was in a multi-level secure status, and the dominance check failed

3C RACROUTE REQUEST=VERIFYX SAF error occurred. Reserved return code not set by RACF.

40 RACROUTE REQUEST=VERIFY SAF error occurred. Reserved return code not set by RACF. SAF saves the return code in the first full word of the RACROUTE parameter list mapped by the ICHSAFP macro. One of the following reason codes may also be stored in the second full word.

- 00 Indicates that an internal TOKENMAP failed while translating. an input token
- 04 Indicates a UTOKEN was specified, but its length was too small. On return, the length byte will be set with the minimum length required.
- 08 Indicates a STOKEN was specified, but its length was too small. On return, the length byte will be set with the minimum length required.
- 0C Indicates a UTOKEN was specified, but its length was too large. On return, the length byte will be set with the correct length required. For a RACROUTE, this is not an error condition, but additional information on a SAF return code 0.
- 10 Indicates a STOKEN was specified, but its length was too large. On return, the length byte will be set with the correct length required. For a RACROUTE, this is not an error condition, but additional information on a SAF return code 0.
- 14 Indicates that a token was specified, but the version was not defined.

44 A token error occurred in RACF. Register 0 contains one of the following reason codes:

- 00 Indicates that an internal TOKENMAP failed while translating an input token.
- 04 Indicates a UTOKEN was specified, but its length was too small. On return, the length byte will be set with the minimum length required.
- 08 Indicates a STOKEN was specified, but its length was too small. On return, the length byte will be set with the minimum length required.
- 0C Indicates that a token was specified, but the version was not defined.

48 Indicates that an unprivileged user issued a RACINIT in a tranquil state (MLQUIET).

4C Indicates that RACF denied access to the submitter's node.

50 Indicates that a surrogate submit attempt failed. Register 0 contains one of the following reason codes:

- 04 Indicates the SURROGAT class was inactive.
- 08 Indicates the submitter is not permitted by the user's SURROGAT class profile.
- 0C Indicates that the submitter is not authorized to the SECLABEL under which the job is to run.

54 Indicates that a JESJOBS check failed.

58 RJE or NJE operator FACILITY class profile not found. For a RACROUTE, this is not an error condition; it means that the requested function could not be performed because RACF was not active.

64 Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACROUTE REQUEST=VERIFY macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example 1

Operation: Use the standard form of the macro to do the following:

- Create an ACEE for the userid and its default group
- Chain the ACEE off either the current TCB or ASXB, or both, by not specifying the ACEE keyword
- Verify that the user named USERNAME is a valid user
- Verify that the password called PASSWORD is valid

```
RACROUTE REQUEST=VERIFY ENVIR=CREATE,USERID=USERNAME,PASSWRD=PASSWORD,
         RELEASE=1.9
```

Example 2

Operation: Use the standard form to do the following:

- Verify that the user named USERNAME is a valid user
- Verify that the group named GROUPNAM is a valid group
- Verify that USERNAME is defined to the group
- Create an ACEE for the user and group and put its address in ACEEANCH
- Specify that the user's password is not required

```
RACROUTE REQUEST=VERIFY,ENVIR=CREATE,USERID=USERNAME,    X
         GROUP=GROUPNAM,ACEE=ACEEANCH,                   X
         PASSCHK=NO,RELEASE=1.9
```

Example 3

Operation: Use the standard form of the macro to delete the accessor environment (ACEE) of the current task or address space, or both.

```
RACROUTE REQUEST=VERIFY,ENVIR=DELETE,RELEASE=1.9
```

RACROUTE REQUEST=VERIFY (List Form)

The list form of the RACROUTE REQUEST=VERIFY macro is written as follows:

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

REQUEST=VERIFY	
,USERID=*userid addr*	*userid addr*: A-type address
,PASSWRD=*password addr*	*password addr*: A-type address
,START=*procname addr*	*procname addr*: A-type address
,NEWPASS=*new password addr*	*new password addr*: A-type address
,GROUP=*group addr*	*group addr*: A-type address
,PGMNAME=*programmer name addr*	*programmer name addr*: A-type address
,ACTINFO=*account addr*	*account addr*: A-type address
,OIDCARD=*oid addr*	*oid addr*: A-type address
,TERMID=*terminal addr*	*terminal addr*: A-type address
,JOBNAME=*jobname addr*	*jobname addr*: A-type address
,ENVIR=CREATE ,ENVIR=VERIFY ,ENVIR=CHANGE ,ENVIR=DELETE	**Default**: ENVIR=CREATE **Notes**: 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr*: A-type address
,APPL='*applname*' ,APPL=*applname addr*	*applname*: 1-8 character name *applname addr*: A-type address
,ACEE=*acee addr*	*acee addr*: A-type address
,SUBPOOL=*subpool number*	*subpool number*: Decimal digit 0-255
,SMC=YES ,SMC=NO	**Default**: SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default**: PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default**: ENCRYPT=YES
,LOC=BELOW ,LOC=ANY	**Default**: See parameter description

,LOC=ABOVE	**Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,RELEASE=*number*	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default**: STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default**: LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: A-type address (2) - (12)
,STOKEN=*stoken addr*	*stoken addr*: A-type address (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: A-type address (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: A-type address (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: A-type address (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: A-type address (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: A-type address (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: A-type address (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: A-type address (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default**: SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default**: TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default**: REMOTE=NO
,MF=L	

The parameters are explained under the standard form of the RACROUTE REQUEST=VERIFY macro, with the following exception:

,MF=L
specifies the list form of the RACROUTE REQUEST=VERIFY macro.

RACROUTE REQUEST = VERIFY (Execute Form)

The execute form of the RACROUTE REQUEST = VERIFY macro is written as follows:

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

REQUEST = VERIFY	
,USERID = *userid addr*	*userid addr*: RX-type address or register (2) - (12)
,PASSWRD = *password addr*	*password addr*: RX-type address or register (2) - (12)
,START = *procname addr*	*procname addr*: RX-type address or register (2) - (12)
,NEWPASS = *new password addr*	*new password addr*: RX-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: RX-type address or register (2) - (12) **Default**: GROUP = zero
,PGMNAME = *programmer name addr*	*programmer name addr*: RX-type address or register (2) - (12)
,ACTINFO = *account addr*	*account addr*: RX-type address or register (2) - (12)
.OIDCARD = *oid addr*	*oid addr*: RX-type address or register (2) - (12)
,TERMID = *terminal addr*	*terminal addr*: RX-type address or register (2) - (12)
,JOBNAME = *jobname addr*	*jobname addr*: RX-type address or register (2) - (12)
,ENVIR = CREATE ,ENVIR = VERIFY ,ENVIR = CHANGE ,ENVIR = DELETE	**Default**: ENVIR = CREATE **Notes**: 1. ENVIR = CHANGE may not be specified with USERID =, PASSWRD =, START =, NEWPASS =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters. 2. ENVIR = DELETE may not be specified with APPL =, USERID =, PASSWRD =, START =, NEWPASS =, GROUP =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters.
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,APPL = *applname addr*	*applname addr*: RX-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,SUBPOOL = *subpool number*	*subpool number*: Decimal digit 0-255
,SMC = YES ,SMC = NO	**Default**: SMC = YES
,PASSCHK = YES ,PASSCHK = NO	**Default**: PASSCHK = YES
,ENCRYPT = YES ,ENCRYPT = NO	**Default**: ENCRYPT = YES
,LOC = BELOW ,LOC = ANY	**Default**: See parameter description

,LOC=ABOVE	**Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: RX-type address or register (2) - (12)
,STOKEN=*stoken addr*	*stoken addr*: RX-type address or register (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: RX-type address or register (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: RX-type address or register (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: RX-type address or register (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: RX-type address or register (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: RX-type address or register (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: RX-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: RX-type address or register (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default:** SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default:** TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default:** REMOTE=NO
,MF=(E,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=VERIFY macro, with the following exception:

,MF=(E,*ctrl addr***)**
specifies the execute form of the RACROUTE REQUEST=VERIFY macro using a remote control program parameter list.

,RELEASE=(*number***,CHECK)**
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=VERIFY macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE REQUEST = VERIFY (Modify Form)

The modify form of the RACROUTE REQUEST = VERIFY macro is written as follows:

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

REQUEST = VERIFY	
,USERID = *userid addr*	*userid addr*: RX-type address or register (2) - (12)
,PASSWRD = *password addr*	*password addr*: RX-type address or register (2) - (12)
,START = *procname addr*	*procname addr*: RX-type address or register (2) - (12)
,NEWPASS = *new password addr*	*new password addr*: RX-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: RX-type address or register (2) - (12) **Default**: GROUP = zero
,PGMNAME = *programmer name addr*	*programmer name addr*: RX-type address or register (2) - (12)
,ACTINFO = *account addr*	*account addr*: RX-type address or register (2) - (12)
,OIDCARD = *oid addr*	*oid addr*: RX-type address or register (2) - (12).
,TERMID = *terminal addr*	*terminal addr*: RX-type address or register (2) - (12)
,JOBNAME = *jobname addr*	*jobname addr*: RX-type address or register (2) - (12)
,ENVIR = CREATE ,ENVIR = VERIFY ,ENVIR = CHANGE ,ENVIR = DELETE	**Default**: ENVIR = CREATE **Notes**: 1. ENVIR = CHANGE may not be specified with USERID =, PASSWRD =, START =, NEWPASS =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters. 2. ENVIR = DELETE may not be specified with APPL =, USERID =, PASSWRD =, START =, NEWPASS =, GROUP =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters.
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,APPL = *applname* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: RX-type address or register (2) - (12)
,ACEE = *acee addr*	*acee addr*: RX-type address or register (2) - (12)
,SUBPOOL = *subpool number*	*subpool number*: Decimal digit 0-255
,SMC = YES ,SMC = NO	**Default**: SMC = YES
,PASSCHK = YES ,PASSCHK = NO	**Default**: PASSCHK = YES
,ENCRYPT = YES ,ENCRYPT = NO	**Default**: ENCRYPT = YES
,LOC = BELOW ,LOC = ANY	**Default**: See parameter description

,LOC=ABOVE	**Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number*: 1.9, 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: RX-type address or register (2) - (12)
,STOKEN=*stoken addr*	*stoken addr*: RX-type address or register (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: RX-type address or register (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: RX-type address or register (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: RX-type address or register (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: RX-type address or register (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: RX-type address or register (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: RX-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: RX-type address or register (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default:** SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default:** TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default:** REMOTE=NO
,MF=(M,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=VERIFY macro, with the following exception:

,MF=(M,*ctrl addr***)**
specifies the modify form of the RACROUTE REQUEST=VERIFY macro using a remote control program parameter list.

,RELEASE=(*number***,CHECK)**
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACROUTE REQUEST=VERIFY macro can be done by your specifying the CHECK subparameter on the modify form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the modify form of the macro, the modify form of the macro will not be done.

RACROUTE REQUEST = VERIFYX - Build a UTOKEN (for RACF Release 1.9)

The RACROUTE REQUEST = VERIFYX macro is used to build a UTOKEN. VERIFYX builds the UTOKEN based on the information passed in the parm list. In addition, VERIFYX handles the propagation of userid and surrogate userid information.

You should be aware of two things about the workings of VERIFYX:

- If the caller specifies an already existing STOKEN to VERIFYX, and if the caller additionally specifies any UTOKEN keywords on the request, be aware that the UTOKEN keywords that are specified will override the corresponding parameters in the STOKEN that was passed. Thus, if the caller specified an STOKEN, the caller should not specify any additional parameters unless the caller wants to supplement STOKEN information.
- If RACF is not active or not installed, then SAF builds a default UTOKEN to satisfy the VERIFYX request. This will be indicated by a bit in the mapped UTOKEN being turned on. If SAF cannot build a complete UTOKEN, it returns a UTOKEN containing all the information available from the RACROUTE parameter list. SAF returns the default UTOKEN at the address specified on the UTOKEN keyword. If a valid SECLABEL was not specified on the call, and could not be obtained from SAF/RACF profiles, SAF returns a default SECLABEL of SYSHIGH if the caller specified TRUSTED = YES, and a value of SYSLOW if the caller specified TRUSTED = NO.

  RACF activates the default SECLABELs when the installation sets the SETROPTS MLACTIVE option.

With Release 3.1.3, SAF ensures that correct propagation of security information involving the unit of work takes place.

To issue the RACROUTE REQUEST = VERIFY macro, the calling module must be 'authorized' which means

- APF-authorized, or
- in system key 0-7, or
- in supervisor state.

or the NEWPASS keyword must be omitted and the calling module must:

- be in the RACF-authorized caller table *and*
- fetched from an authorized library *and*
- reentrant.

Note: It is recommended that if you run programs which issue the RACROUTE REQUEST = VERIFYX macro, you run those programs AFP-authorized. See *SPL: RACF* for information on the authorized caller table.

To use this service, you must also specify RELEASE = 1.9.

The standard form of the RACROUTE REQUEST=VERIFYX macro is written as follows:

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

REQUEST=VERIFYX	
,USERID=*userid addr*	*userid addr*: A-type address or register (2) - (12)
,PASSWRD=*password addr*	*password addr*: A-type address or register (2) - (12)
,START=*procname addr*	*procname addr*: A-type address or register (2) - (12)
,NEWPASS=*new password addr*	*new password addr*: A-type address or register (2) - (12)
,GROUP=*group addr*	*group addr*: A-type address or register (2) - (12) **Default**: GROUP=zero
,PGMNAME=*programmer name addr*	*programmer name addr*: A-type address or register (2) - (12)
,ACTINFO=*account addr*	*account addr*: A-type address or register (2) - (12)
,OIDCARD=*oid addr*	*oid addr*: A-type address or register (2) - (12).
,TERMID=*terminal addr*	*terminal addr*: A-type address or register (2) - (12)
,JOBNAME=*jobname addr*	*jobname addr*: A-type address or register (2) - (12)
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default**: ENVIR=CREATE **Notes**: 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr*: A-type address or register (2) - (12)
,APPL='*applname*' ,APPL=*applname addr*	*applname*: 1-8 character name *applname addr*: A-type address or register (2) - (12)
,SMC=YES ,SMC=NO	**Default**: SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default**: PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default**: ENCRYPT=YES
,RELEASE=*number*	*number*: 1.9
,STAT=ASIS ,STAT=NO	**Default**: STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default**: LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: A-type address or register (2) - (12)

,STOKEN=*stoken addr*	*stoken addr*: A-type address or register (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: A-type address or register (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: A-type address or register (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: A-type address or register (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: A-type address or register (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: A-type address or register (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: A-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: A-type address or register (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default**: SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default**: TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default**: REMOTE=NO

The parameters are explained as follows:

,USERID= *userid addr*
specifies the identification of the user who has entered the system. The address points to a 1-byte length field, followed by the userid which can be up to 8 characters long.

,PASSWRD= *password addr*
specifies the currently defined password of the user who has entered the system. The address points to a 1-byte length field, followed by the password which can be up to 8 characters long.

,START= *procname addr*
specifies the PROC name of a started task for which the RACROUTE REQUEST=VERIFYX is being performed. The address points to an 8-byte area containing the PROC name (left-justified and padded with blanks, if necessary). If START= is specified, REQUEST=VERIFYX processing searches the started procedures table for the userid and group to use for this REQUEST=VERIFYX request. If the USERID and GROUP keywords are specified, REQUEST=VERIFYX will only use those values f it is unsuccessful in obtaining a userid and group form the started procedures table.

If START is specified, PASSWRD and OIDCARD should not be specified.

,NEWPASS= *new password addr*
specifies the password which is to replace the user's currently defined password. The address points to a 1-byte length field, followed by the password which can be up to 8 characters long.

,GROUP= *group addr*
specifies the group of the user who has entered the system. The address points to a 1-byte length field, followed by the group name which can be up to 8 characters long.

,PGMNAME = *programmer name addr*
specifies the address of the name of the user who has entered the system. This twenty byte area is passed to the RACINIT installation exit routine; it is not used by the RACINIT routine.

,ACTINFO = *account addr*
specifies the address of a field containing accounting information. This 144 byte area is passed to the RACINIT installation exit routine; it is not used by the RACINIT routine. The accounting field, if supplied, should have the following format:

- First byte of field contains the number (binary) of accounting fields.
- Following bytes contain accounting fields, where each entry for an accounting field contains a 1-byte length field, followed by the field.

,OIDCARD = *oid addr*
specifies the address of the currently defined operator identification card of the user who has entered the system. The address points to a 1-byte length field, followed by the operator ID card.

,TERMID = *terminal addr*
specifies the address of the identifier for the terminal through which the user is accessing the system. The address points to an 8-byte area containing the terminal identifier. The area must reside in a non-task-related storage subpool.

,JOBNAME = *jobname addr*
specifies the address of the JOB name of a background job. The address points to an eight byte area containing the JOB name (left justified and padded with blanks, if necessary). The JOBNAME parameter is used by RACINIT during authorization checking to verify the user's authority to submit the job. It is passed to the installation exit routine.

,INSTLN = *parm list addr*
specifies the address of an area containing parameter information meaningful to the RACINIT installation exit routine. This area is passed to the installation exit when the exit routine is given control from the RACINIT routine.

The INSTLN parameter can be used by an installation having a user verification or job initiation application, and wanting to pass information from one installation module to the RACINIT installation exit routine.

,APPL = *'applname'*
,APPL = *applname addr*
specifies the name of the application issuing the RACROUTE REQUEST = VERIFYX. If an address is specified, the address must point to an 8-byte application name, left justified and padded with blanks, if necessary.

,SMC = YES
,SMC = NO
specifies the use of the step-must-complete function of RACROUTE REQUEST = VERIFYX processing. SMC = YES specifies that RACROUTE REQUEST = VERIFYX processing should continue to place other tasks for the step non-dispatchable. SMC = NO specifies that the step-must-complete function is not used.

Note: SMC = NO should not be used if DADSM ALLOCATE/SCRATCH functions execute simultaneously in the same address space as the RACROUTE REQUEST = VERIFYX function.

,PASSCHK = YES
,PASSCHK = NO
specifies whether or not the user's password is to be verified. PASSCHK = YES specifies that RACROUTE REQUEST = VERIFYX verifies the user's password. PASSCHK = NO specifies that the user's password is not verified.

,ENCRYPT = YES
,ENCRYPT = NO

specifies whether or not RACROUTE REQUEST = VERIFYX will encrypt the old password, the new password, and the OIDCARD data passed to it.

YES signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are not pre-encrypted. RACROUTE REQUEST = VERIFYX encrypts the data before storing it in the user profile or using it to compare against stored data. ENCRYPT = YES is the default for this keyword.

NO signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are already encrypted. RACROUTE REQUEST = VERIFYX bypasses the encryption of this data before storing it in, or comparing it against, the user profile.

Note: The exit routine ICHDEX01 can also perform the encryption.

,RELEASE = *number*

specifies the RACF release level of the parameter list to be generated by this macro.

To use the parameters associated with a release, you must specify the release number of that release or a later release number. If you specify an earlier release level, the parameter will not be accepted by macro processing, and an error message will be issued at assembly time.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST = VERIFYX macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,STAT = ASIS
,STAT = NO

specifies whether the statistics controlled by the installation's options on the RACF SETROPTS command are to be maintained or ignored for this execution of RACROUTE REQUEST = VERIFYX. This parameter also controls whether a message is to be issued when the logon is successful.

Note: Messages are always issued if the RACROUTE REQUEST = VERIFYX processing is unsuccessful.

If STAT = ASIS is specified or taken by default, the messages and statistics are controlled by the installation's current options on the RACF SETROPTS command.

If STAT = NO is specified, the statistics are not updated. And, if the logon is successful, no message is issued.

The default is STAT = ASIS.

,LOG = ASIS
,LOG = ALL

specifies when log records are to be generated.

If LOG = ASIS is specified or defaulted to, only those attempts to create an ACEE that fail will generate RACF log records.

If LOG = ALL is specified, any request to create an ACEE, regardless of whether it succeeds or fails, will generate a RACF log record. The default is LOG = ASIS.

,UTOKEN = *utoken addr*

specifies the address of the UTOKEN of the user for which RACF will perform a RACROUTE REQUEST = VERIFY. The first byte contains the length of the UTOKEN, and the second byte contains the version number. The version number is 01.

The UTOKEN fields are mapped in ICHRUTKN in the Data Areas chapter of the *SPL: RACF.*

,STOKEN = *stoken addr*

specifies the address of the submittor's UTOKEN. The first byte contains the length of the UTOKEN, and the second byte contains the version number. The version number is 01. See explanation of UTOKEN.

,SECLABL = *seclabel addr*
specifies the address of an 8-byte left-justified character field containing the SECLABEL.

,EXENODE = *execution node addr*
specifies the address of an area that contains a one byte length field followed by the name of the node on which the unit of word is to be executed. The node name cannot exceed eight bytes.

,SNODE = *submitting node addr*
specifies the address of an area that contains a one byte length field followed by the name of the node from which the unit of work was submitted. The node name cannot exceed eight bytes.

,SUSERID = *submitting userid addr*
specifies the address of an area that contains a one byte length field followed by the userid of the user who submitted the unit of work. The userid cannot exceed eight bytes.

,SGROUP = *submitting group addr*
specifies the address of an area that contains a one byte length field followed by the groupid of the user who submitted the unit of work. The groupid cannot exceed eight bytes.

,POE = *port of entry addr*
specifies the address of the port of entry into the system. The accress points to the name of the input device through which the job was submitted. The port of entry is an 8 character field which is left-justified and padded with blanks.

The port of entry will be a part of the user's security token (UTOKEN). A flag in the UTOKEN will uniquely identify the RACF general resource class to which the data in the POE field belongs: TERMINAL, CONSOLE, or JESINPUT.

When both the POE and TERMID keywords are specified, the POE keyword will take precedence.

,LOGSTR = *logstr addr*
specifies the address of a one byte length field followed by character data that will be written to the SMF data set together with RACF audit information.

,SESSION = *type*
specifies the session type(s) to be associated with the request. You can specify multiple sessions, separated by commas, if necessary. For example, SESSION = CONSOPER,SYSAS. Session types are literals. When the SESSION keyword is used in combination with the POE keyword, SESSION determines the class with which the POE keyword will be connected.

The allowable session types are:

- SYSAS = a system address space
- COMMAND = a command
- TSO = a TSO logon
- CONSOPER = a console operator
- STARTED = a started procedure of started task
- MOUNT = a mount command
- XBM = a execution batch monitor job
- BATCH = a batch job
- NJEOPER = network job entry
- RJEOPER = remote job entry

,TRUSTED = YES
,TRUSTED = NO
specifies whether or not the submitter of the unit of work is a member of the trusted computer base.

,REMOTE=YES
,REMOTE=NO
specifies whether or not the job came through the network.

Return Codes and Reason Codes

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

Hexadecimal Code	Meaning
00	RACROUTE REQUEST=VERIFYX has completed successfully. When you execute the macro, space for the return code and reason code is reserved in the first two words of the RACROUTE parameter list. You can access them via the ICHSAFP mapping by loading the ICHSAFP pointer with the label that you specified on the execute form of the macro. **Return code: 3C** Request completed successfully, but a VERIFYX error occurred in SAF. Reason code: 20 UTOKEN area specified was too large; on return, the length field will contain the length used. Reason code: 24 STOKEN area specified was too large; on return, the length field will contain the length used.
04	The requested function could not be performed. In addition, the following return code and reason codes have been saved in the ICHSAFP return and reason code fields: **Return code: 00** No security decision could be made. Reason code: 00 The RACF Router was not loaded; the request, resource, subsystem combination could not be found in the RACF ROUTER table; RACF is not active, no successful exit processing, or RACF is not most current release. 08 The requested function failed. In addition, the following return code and reason codes have been saved in the ICHSAFP return and reason code fields: **Return code: 00** Default ACEE/token building error. Reason code: 00 SAF failed to set up a recovery environment. **Return code: 3C** VERIFYX error occurred in SAF. Reason code: 04 Old password required. Message IRR1011 issued. Reason code: 08 Userid required. Message IRR1011 issued. Reason code: 0C Propagation checking could not complete. Reason code: 10 Required UTOKEN keyword missing. Reason code: 14 Required UTOKEN keyword length that was specified was too small. On return, the token length field has the minimum required length. Reason code: 18 An internal TOKENMAP request to decrypt an input token failed. Reason code: 1C STOKEN keyword length that was specified was too small. On return, the token length field has the minimum required length. Reason code: 28 The token that was specified had an undefined version (0).

Example 1

Operation: The following example shows a RACROUTE REQUEST=VERIFYX coded to handle verification checking for a batch job that has been submitted with a valid USERID, GROUPID, SECLABEL, and PASSWORD. The UTOKEN area will be filled with the verified job information.

```
RACROUTE REQUEST=VERIFYX,
         SUBSYS=address of caller subsystem,
         REQSTOR=address of caller subsystem control point,
         SESSION=BATCH,
         PASSWRD=address of password supplied,
         UTOKEN=address of call-supplied UTOKEN area,
         EXENODE=address of execution node,
         USERID=addess of session owner userid,
         GROUP=addess of session owner group name,
         SECLABEL=address of session owner SECLABEL,
         STOKEN=address of submitter utoken area,
         TRUSTED=NO,
         ,RELEASE=1.9
```

Note: Additional keywords such as WORKA, required by RACF to complete the request, are specified on RACROUTE itself.

RACROUTE REQUEST = VERIFYX (List Form)

The list form of the RACROUTE REQUEST = VERIFYX macro is written as follows:

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

REQUEST = VERIFYX	
,USERID = *userid addr*	*userid addr*: A-type address
,PASSWRD = *password addr*	*password addr*: A-type address
,START = *procname addr*	*procname addr*: A-type address
,NEWPASS = *new password addr*	*new password addr*: A-type address
,GROUP = *group addr*	*group addr*: A-type address
,PGMNAME = *programmer name addr*	*programmer name addr*: A-type address
,ACTINFO = *account addr*	*account addr*: A-type address
,OIDCARD = *oid addr*	*oid addr*: A-type address
,TERMID = *terminal addr*	*terminal addr*: A-type address
,JOBNAME = *jobname addr*	*jobname addr*: A-type address
,INSTLN = *parm list addr*	*parm list addr*: A-type address
,APPL = *'applname'* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: A-type address
,SMC = YES ,SMC = NO	**Default**: SMC = YES
,PASSCHK = YES ,PASSCHK = NO	**Default**: PASSCHK = YES
,ENCRYPT = YES ,ENCRYPT = NO	**Default**: ENCRYPT = YES
,RELEASE = *number*	*number*: 1.9
,STAT = ASIS ,STAT = NO	**Default**: STAT = ASIS
,LOG = ASIS ,LOG = ALL	**Default**: LOG = ASIS
,UTOKEN = *utoken addr*	*utoken addr*: A-type address (2) - (12)
,STOKEN = *stoken addr*	*stoken addr*: A-type address (2) - (12)
,SECLABL = *seclabel addr*	*seclabel addr*: A-type address (2) - (12)
,EXENODE = *execution node addr*	*execution node addr*: A-type address (2) - (12)
,SNODE = *submitting node addr*	*submitting node addr*: A-type address (2) - (12)

,SUSERID=*submitting userid addr*	*submitting userid addr*: A-type address (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: A-type address (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: A-type address (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: A-type address (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default**: SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default**: TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default**: REMOTE=NO
,MF=L	

The parameters are explained under the standard form of the RACROUTE REQUEST=VERIFYX macro, with the following exception:

,MF=L
specifies the list form of the RACROUTE REQUEST=VERIFYX macro.

RACROUTE REQUEST = VERIFYX (Execute Form)

The execute form of the RACROUTE REQUEST = VERIFYX macro is written as follows:

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

REQUEST = VERIFYX	
,USERID = *userid addr*	*userid addr*: RX-type address or register (2) - (12)
,PASSWRD = *password addr*	*password addr*: RX-type address or register (2) - (12)
,START = *procname addr*	*procname addr*: RX-type address or register (2) - (12)
,NEWPASS = *new password addr*	*new password addr*: RX-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: RX-type address or register (2) - (12) **Default**: GROUP = zero
,PGMNAME = *programmer name addr*	*programmer name addr*: RX-type address or register (2) - (12)
,ACTINFO = *account addr*	*account addr*: RX-type address or register (2) - (12)
,OIDCARD = *oid addr*	*oid addr*: RX-type address or register (2) - (12)
,TERMID = *terminal addr*	*terminal addr*: RX-type address or register (2) - (12)
,JOBNAME = *jobname addr*	*jobname addr*: RX-type address or register (2) - (12)
,ENVIR = CREATE ,ENVIR = CHANGE ,ENVIR = DELETE	**Default**: ENVIR = CREATE **Notes:** 1. ENVIR = CHANGE may not be specified with USERID =, PASSWRD =, START =, NEWPASS =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters. 2. ENVIR = DELETE may not be specified with APPL =, USERID =, PASSWRD =, START =, NEWPASS =, GROUP =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters.
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,APPL = *applname* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: RX-type address or register (2) - (12)
,SMC = YES ,SMC = NO	**Default**: SMC = YES
,PASSCHK = YES ,PASSCHK = NO	**Default**: PASSCHK = YES
,ENCRYPT = YES ,ENCRYPT = NO	**Default**: ENCRYPT = YES
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9
,STAT = ASIS ,STAT = NO	**Default**: STAT = ASIS

,LOG=ASIS ,LOG=ALL	**Default**: LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: RX-type address or register (2) - (12)
,STOKEN=*stoken addr*	*stoken addr*: RX-type address or register (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: RX-type address or register (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: RX-type address or register (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: RX-type address or register (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: RX-type address or register (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: RX-type address or register (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: RX-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: RX-type address or register (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default**: SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default**: TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default**: REMOTE=NO
,MF=(E,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=VERIFYX macro, with the following exception:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACROUTE REQUEST=VERIFYX macro using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACROUTE REQUEST=VERIFYX macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done.

RACROUTE REQUEST = VERIFYX (Modify Form)

The modify form of the RACROUTE REQUEST = VERIFYX macro is written as follows:

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

REQUEST = VERIFYX	
,USERID = *userid addr*	*userid addr*: RX-type address or register (2) - (12)
,PASSWRD = *password addr*	*password addr*: RX-type address or register (2) - (12)
,START = *procname addr*	*procname addr*: RX-type address or register (2) - (12)
,NEWPASS = *new password addr*	*new password addr*: RX-type address or register (2) - (12)
,GROUP = *group addr*	*group addr*: RX-type address or register (2) - (12) **Default**: GROUP = zero
,PGMNAME = *programmer name addr*	*programmer name addr*: RX-type address or register (2) - (12)
,ACTINFO = *account addr*	*account addr*: RX-type address or register (2) - (12)
,OIDCARD = *oid addr*	*oid addr*: RX-type address or register (2) - (12)
,TERMID = *terminal addr*	*terminal addr*: RX-type address or register (2) - (12)
,JOBNAME = *jobname addr*	*jobname addr*: RX-type address or register (2) - (12)
,ENVIR = CREATE ,ENVIR = CHANGE ,ENVIR = DELETE	**Default**: ENVIR = CREATE **Notes**: 1. ENVIR = CHANGE may not be specified with USERID =, PASSWRD =, START =, NEWPASS =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters. 2. ENVIR = DELETE may not be specified with APPL =, USERID =, PASSWRD =, START =, NEWPASS =, GROUP =, ACTINFO =, PGMNAME =, OIDCARD =, or TERMID = parameters.
,INSTLN = *parm list addr*	*parm list addr*: RX-type address or register (2) - (12)
,APPL = *applname* ,APPL = *applname addr*	*applname*: 1-8 character name *applname addr*: RX-type address or register (2) - (12)
,SMC = YES ,SMC = NO	**Default**: SMC = YES
,PASSCHK = YES ,PASSCHK = NO	**Default**: PASSCHK = YES
,ENCRYPT = YES ,ENCRYPT = NO	**Default**: ENCRYPT = YES
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number*: 1.9
,STAT = ASIS ,STAT = NO	**Default**: STAT = ASIS

,LOG=ASIS ,LOG=ALL	**Default**: LOG=ASIS
,UTOKEN=*utoken addr*	*utoken addr*: RX-type address or register (2) - (12)
,STOKEN=*stoken addr*	*stoken addr*: RX-type address or register (2) - (12)
,SECLABL=*seclabel addr*	*seclabel addr*: RX-type address or register (2) - (12)
,EXENODE=*execution node addr*	*execution node addr*: RX-type address or register (2) - (12)
,SNODE=*submitting node addr*	*submitting node addr*: RX-type address or register (2) - (12)
,SUSERID=*submitting userid addr*	*submitting userid addr*: RX-type address or register (2) - (12)
,SGROUP=*submitting group addr*	*submitting group addr*: RX-type address or register (2) - (12)
,POE=*port of entry addr*	*port of entry addr*: RX-type address or register (2) - (12)
,LOGSTR=*logstr addr*	*logstr addr*: RX-type address or register (2) - (12)
,SESSION=TSO ,SESSION=BATCH ,SESSION=XBM ,SESSION=CONSOPER ,SESSION=STARTED ,SESSION=MOUNT ,SESSION=COMMAND ,SESSION=SYSAS ,SESSION=NJEOPER ,SESSION=RJEOPER	**Default**: SESSION=TSO
,TRUSTED=YES ,TRUSTED=NO	**Default**: TRUSTED=NO
,REMOTE=YES ,REMOTE=NO	**Default**: REMOTE=NO
,MF=(M,*ctrl addr*)	*cntl addr*: RX-type address or register (1) or (2) - (12)

The parameters are explained under the standard form of the RACROUTE REQUEST=VERIFYX macro, with the following exception:

,MF=(M,*ctrl addr*)
specifies the modify form of the RACROUTE REQUEST=VERIFYX macro using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level 1.9 of the parameter list to be generated by this macro.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and modify forms of the RACROUTE REQUEST=VERIFYX macro can be done by your specifying the CHECK subparameter on the modify form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the modify form of the macro, the modify form of the macro will not be done.

RACSTAT - Determines the Status of RACF (for RACF Release 1.8.1 or earlier)

This macro description applies to RACF Release 1.8.1 or earlier. If you have RACF Release 1.9 installed on your system, you can still invoke the RACSTAT macro directly. See the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST=STAT:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 445**
- **"RACROUTE REQUEST=STAT - Determine RACF Status (for RACF Release 1.9)" on page 547.**

The RACSTAT macro determines if RACF is active and optionally determines if RACF protection is in effect for a given resource class. You can also use the RACSTAT macro to determine if a resource class name is defined to RACF.

RACSTAT is a branch entered service that uses standard linkage conventions.

Note: For RACF release 1.6 and prior releases, only callers in 24-bit addressing mode can issue this macro.

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

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

CLASS=*'classname'*	*classname:* DATASET, DASDVOL, or TAPEVOL, or any class defined in the RACF class descriptor table
CLASS=*classname addr*	*classname addr:* A-type address, or register (2) - (12).
ENTRY=*entry addr*	*entry addr:* A-type address, or register (2) - (12).
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6

The parameters are explained as follows:

CLASS =*'classname'*
CLASS =*classname addr*
specifies the classname for which RACF authorization checking is performed. You can explicitly define the name on the macro by enclosing the name in quotes. If specified, the address must point to an 8-byte field containing the classname, left justified and padded with blanks if necessary. If you omit CLASS=, the status of RACF is returned.

ENTRY =*entry addr*
specifies the address of a 4-byte area that is set to the address of the specified class in the class descriptor table. This operand is ignored when you omit the CLASS= operand.

,RELEASE= *number*
specifies the RACF release level of the parameter list that this macro will generate.

You can specify certain parameters only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 19.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. To get execution-time validation of the compatibility between the list and execute forms of the RACSTAT macro, specify the CHECK subparameter on the execute form of the macro.

Parameters for RELEASE=1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Figure 19. RACSTAT Parameters for RELEASE=1.6 and Later

Parameter	RELEASE=1.6	RELEASE=1.7	RELEASE=1.8 or 1.8.1
CLASS=	X	X	X
ENTRY=	X	X	X
RELEASE=	X	X	X

Return Codes

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

Hexadecimal Code	Meaning
00	RACF is active and, if CLASS= was specified, the class is active.
04	RACF is active; the class is inactive.
08	RACF is active; the class is not defined to RACF.
0C	RACF is inactive and, if CLASS= was specified, the class is active.
10	RACF is inactive; the class is inactive.
14	RACF is inactive; the class is not defined to RACF.
18	RACF CVT does not exist (RACF is not installed) or an insufficient level of RACF is installed.
64	Indicates that you specified the CHECK subparameter of the RELEASE keyword on the execute form of the RACSTAT macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Note: The class descriptor entry for the specified class is returned to the caller (in the 4-byte area addressed by the entry addr) for return codes 00, 04, 0C, and 10.

Example 1

Operation: Determine if the DASDVOL class is active and retrieve the address of its class descriptor. A fullword, CDADDR, contains the class descriptor address.

```
RACSTAT CLASS='DASDVOL',ENTRY=CDADDR
```

RACSTAT (List Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACSTAT.
RACSTAT	
b	One or more blanks must follow RACSTAT.
CLASS=*'classname'* CLASS=*classname addr*	*classname:* DATASET, DASDVOL, or TAPEVOL. *classname addr:* A-type address.
ENTRY=*entry addr*,	*entry addr:* A-type address.
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
MF=L	

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

MF=L
specifies the list form of the RACSTAT macro.

RACSTAT (Execute Form)

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

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

CLASS=*'classname'* CLASS=*classname addr*	*classname:* DATASET, DASDVOL, or TAPEVOL. *classname addr:* RX-type address or register (2) - (12).
ENTRY=*entry addr*	*entry addr:* RX-type address or register (2) - (12).
,RELEASE=*(number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.8.1, 1.8, 1.7, 1.6 **Default:** RELEASE=1.6
MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address or register (1) - (12).

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

MF =(E,*ctrl addr*)
specifies the execute form of the RACSTAT macro, using a remote control program parameter list.

,RELEASE =(*number*,CHECK)
,RELEASE= *number*
,RELEASE= (,CHECK)
specifies the RACF release level of the parameter list that this macro will generate.

You can specify certain parameters only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 19.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. To get execution-time validation of the compatibility between the list and execute forms of the RACSTAT macro, specify the CHECK subparameter on the execute form of the macro.

When you request CHECK processing and the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be generated. Instead, RACSTAT generates a return code of 'X64'.

RACXTRT — Retrieve Fields from RACF User Profile (for RACF Release 1.8.1 or earlier)

Note: The RACROUTE macro is the preferred programming interface.

This macro description applies to RACF Release 1.8.1 or earlier. Your program can invoke the RACXTRT macro directly; however, IBM recommends that you invoke the equivalent function through the RACROUTE macro, using the REQUEST=EXTRACT parameter. See "RACROUTE — MVS Router Interface (for RACF Release 1.8.1 or earlier)" on page 435 for the applicable RACROUTE macro description.

If you have RACF Release 1.9 installed on your system, you can still invoke the RACXTRT macro directly; however, to use the new Release 1.9 functions, you must use the RACROUTE macro and specify REQUEST=EXTRACT. See the following for the applicable descriptions of RACROUTE and RACROUTE REQUEST=EXTRACT:

- **"RACROUTE — Router Interface (for RACF Release 1.9)" on page 445**
- **"RACROUTE REQUEST=EXTRACT — Replace or Retrieve Fields (for RACF Release 1.9)" on page 511.**

The RACXTRT macro retrieves or replaces certain specified fields from a RACF profile or encrypts certain clear-text (readable) data.

Note: Encryption and extraction are mutually exclusive.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to use the RACXTRT function, can code the RACROUTE macro.

PROGRAMMING INTERFACES

ONLY the following RACXTRT functions are general-use programming interfaces:

- Retrieving or updating fields in the TSO segment in the user profile
- Retrieving or updating fields in the user and data set profiles
- Retrieving or updating the following installation-reserved fields:
  - USERDATA
  - USRCNT
  - USRDATA
  - USRFLG
  - USRNM

PRODUCT-SENSITIVE PROGRAMMING INTERFACE

ONLY the following RACXTRT functions are product-sensitive programming interfaces:

- Retrieving or updating fields in the base segment

End of PRODUCT-SENSITIVE PROGRAMMING INTERFACE

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACXTRT.
RACXTRT	
b	One or more blanks must follow RACXTRT.
TYPE=EXTRACT TYPE=EXTRACTN TYPE=REPLACE TYPE=ENCRYPT	
,ENTITY=*profile name addr*	*profile name addr:* A-type address, or register (12)
RELEASE=*number*	*number:* 1.6, 1.7, 1.8, or 1.8.1 **Default:** RELEASE=1.6
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12)
,VOLSER=*volser addr*	*volser addr :* A-type address, or register (2) - (12)
,GENERIC=ASIS ,GENERIC=YES	**Default:** ASIS
,FLDACC=YES ,FLDACC=NO	**Default:** NO
If you specify TYPE=EXTRACT or EXTRACTN:	
,SUBPOOL=*subpool number*	*subpool number:* decimal digit, 0-255 **Default:** SUBPOOL=229
,DERIVE='YES'	*see explanation of keyword* **Default:** normal processing
,CLASS='*class name*'	*the entity's class name* **Default:** USER
,CLASS=*class name addr*	*class name addr*: A-type address or register (2) - (12)
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*'segment name'*: 1-8 character name segment name addr: A-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr:* A-type address or register (2) - (12)
If you specify TYPE=REPLACE:	
,CLASS='*class-name*'	*the entity's class name* **Default:** USER
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*'segment name'*: 1-8 character name *segment name addr:* A-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr:* A-type address or register (2) - (12)
,SEGDATA= *segment data addr*	*segment data addr:* A-type address or register (2) - (12)
If you specify TYPE=ENCRYPT:	
,ENCRYPT=(*data addr*,DES) ,ENCRYPT=(*data addr*,HASH) ,ENCRYPT=(*data addr*,INST)	*data addr:* A-type address or register (2) - (12)

Note: If you specify TYPE=ENCRYPT, the only other allowable parameters are ENTITY, RELEASE, ENCRYPT, with ENCRYPT being required.

The parameters are explained as follows:

TYPE = EXTRACT

specifies the function to be performed by the extract function routine.

With Release 1.8 and later, RACXTRT can provide additional function: it can extract information from any field in any profile. The profile templates in Chapter 3 define the type and name of each field in each profile. If you specify EXTRACT, the macro extracts information from the profile determined by the ENTITY and CLASS keywords. Specifically, RACF extracts the fields specified in the FIELDS keyword from the segment specified by the SEGMENT keyword. If you do not specify ENTITY, RACF retrieves the desired information from the current user's profile.

To use TYPE = EXTRACT to extract field information from a profile, you must specify Release = 1.8.

Note: If you specify TYPE = EXTRACT, do not specify ENCRYPT.

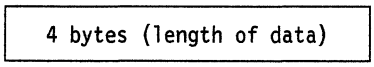
Upon return, register 1 contains the address of a result area that begins with a fullword containing the area's subpool number and length. It is your responsibility to issue a FREEMAIN to release the area after you are through using it.

The fields in the result area are in the order below:

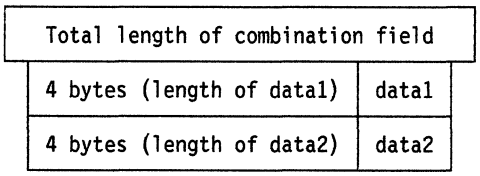
Offset (Dec)	Data	Length (Dec)
0	subpool of area	1
1	length of area	3
4	offset to start of optional field to contain segment data	2
6	reserved	18
24	specified or current user's userid, if CLASS = USER	8
32	specified user's default connect group or current user's current connect group, if CLASS = USER	8

In general, RACF returns field data in the order it was specified, with a four byte length field preceding each profile field. For example, if you were extracting the following:

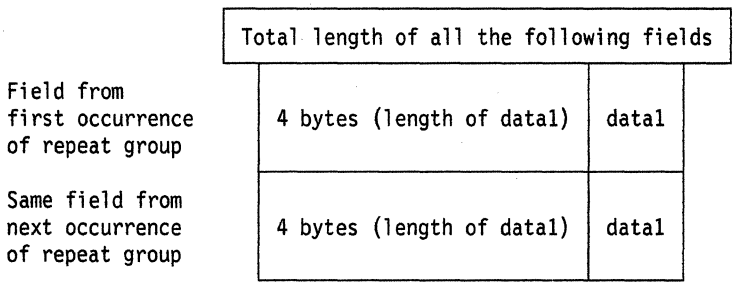
- Single field, you would receive
  - A 4-byte length field that contains the length of the fields that follow.
  - No additional length byte if the requested field is a variable length field.



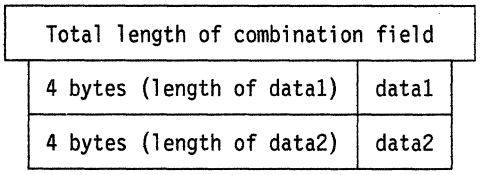
- Combination field (representing one or more fields), you would receive
  - A 4-byte length field that contains the combined length of all the fields that follow.
  - A combination field made up of 4-byte length fields followed by their respective individual data fields.



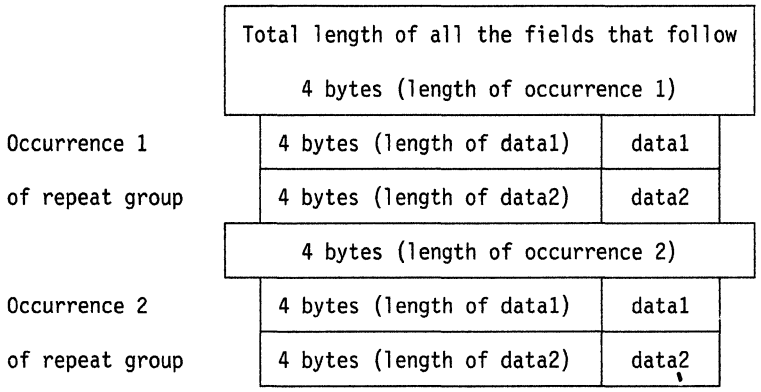
- Single field within a repeat group, you would receive
  - A 4-byte length field that contains the combined length of all the fields that follow.
  - A 4-byte length field that indicates the length of the specified field in the first occurrence of the repeat group. This is followed by a 4-byte length field that indicates the length of the specified field in the second occurrence of the repeat group, and so on, until all the occurrences of the repeat group are accounted for.



- Combination field (representing one or more fields) within a repeat group, you would receive
  - A 4-byte length field that contains the combined length of all the fields that follow.
  - A combination field consisting of a 4-byte length field indicating the length of the individual data field that follows it, followed by the next 4-byte length field indicating the length of the next individual data field, and so on, until all the individual fields that make up the combination field are accounted for. We then move on to the next occurrence of the repeat group and begin again.



- A repeat group count field, you would receive
  - A 4-byte length field that contains the total length of all the fields contained in the repeat group.
  - A 4-byte field that indicates the length of the fields in the *first* occurrence of the repeat group. This is followed by a 4-byte length field that indicates the length of the first data field in the repeat group followed by its data field and another 4-byte length field that indicates the length of the next field followed by its data field. This continues until all fields within the repeat group are accounted for. When the first occurrence of the repeat group count field is accounted for, there is another 4 byte field that indicates the length of the fields in the *second* occurrence of the repeat group, again followed by the same field configurations until all the occurrences of the repeat group are accounted for. The only difference between this example and the one above is that the length of each occurrence precedes that occurrence.



When a field to be extracted is empty, the following occurs:

- For fixed length fields, RACF returns the default as specified by the template definitions. The default for flag fields is X'00'. The default for fixed length fields is the BASE segment of the profile in binary ones. The default for fixed length fields in other segments is binary zeros.
- For variable length fields, RACF returns a length of zero and no data.

  If CLASS=USER when you specify EXTRACT, the macro extracts the userid, connect group, and, optionally, the encrypted password from the user profile.

TYPE =EXTRACTN
specifies the function to be performed by the EXTRACT function routine.

Note: If you specify TYPE=EXTRACTN, do not specify ENCRYPT=.

Upon return, register 1 contains the address of a result area that begins with a fullword containing the area's subpool number and length. To see the format of the result area, see the explanation of TYPE=EXTRACT above.

If you specify EXTRACTN, the macro extracts information from the profile that follows the profile determined by the ENTITY and CLASS keywords. From that next profile, RACF extracts the fields specified in the FIELDS keyword from the segment specified by the SEGMENT keyword. In addition, RACF returns the name of the profile from which it extracted the data.

TYPE =REPLACE
specifies the function that the EXTRACT function routine will perform.

Note: If you specify TYPE=REPLACE, do not specify ENCRYPT=.

Using the REPLACE option to update a profile requires a thorough knowledge of the inter-relationships of fields within a profile and the potential relationships between profiles. For instance, if you use RACXTRT to update a password, you should also update the password change date and password history information.

If you specify TYPE=REPLACE, RACF takes the information in the fields specified in the FIELDS parameter and pointed to by SEGDATA, and places that information in the designated SEGMENT. (The SEGMENT is within the profile determined by the ENTITY and CLASS keywords.) If you do not specify ENTITY, RACF returns an error code. If you specify TYPE=REPLACE, you must specify FIELDS, SEGDATA=, and RELEASE=1.8 or later. If you want to replace a SEGMENT other than the BASE segment, you must specify the SEGMENT keyword with the segment you want. If you do not specify SEGMENT, the segment defaults to the BASE segment.

With 1.8 and later, if you want to create a TSO segment, you can do so by specifying the RACXTRT macro in the following way:

TYPE=REPLACE SEGMENT=TSO

TYPE=ENCRYPT
specifies the function to be performed by the extract function routine.

If you specify TYPE=ENCRYPT, the operation performed is data encryption. The ENCRYPT keyword specifies the data to be encrypted and the encryption method used. The DES (Data Encryption Standard) encryption routine will use the first eight bytes of the area to which the ENTITY operand points. If you do not specify ENTITY, the userid from the current ACEE will be used instead. If you specify TYPE=ENCRYPT, no work area will be returned.

,SUBPOOL=*subpool number*
specifies the storage subpool from which the extract function routine obtains an area needed for the extraction. If you do not specify this parameter, it defaults to 229.

,DERIVE=YES
specifies that the desired field will be obtained from the DFP segment of the appropriate profile. To specify DERIVE, you must also specify RELEASE=1.8 or later.

DERIVE requests are limited to the DFP segment of the DATASET and USER profiles. The following is an explanation of the DERIVE processing for both a DATASET and USER request.

- DATASET

  Specifying the DERIVE=YES keyword with CLASS=DATASET and FIELDS=RESOWNER causes RACF to perform additional processing other than simply extracting the data set resource owner from the data set profile.

  DFP uses this retrieved information for authority checking when allocating a new data set.

  To process the request, RACF first attempts to extract the RESOWNER field from the DATASET profile specified by the ENTITY keyword. If the profile exists and the RESOWNER field contains data, RACF checks to see if that data is the userid of a USER or GROUP currently defined to RACF. If so, RACF returns that userid along with a reason code which indicates whether the userid is that of a USER or GROUP.

  If RACF does not find a profile that matches the DATASET name specified by the ENTITY keyword, RACF attempts to locate the generic DATASET profile that protects that DATASET name.

  If it finds the generic profile, and the RESOWNER field contains data, RACF checks to see if that data is the userid of a USER or GROUP currently defined to RACF. If so, RACF returns that userid along with a reason code which indicates whether the userid is that of a USER or GROUP.

  If RACF does not find a generic profile or the retrieved data is neither a USER or GROUP, RACF returns the high-level qualifier from the name specified on the ENTITY keyword along with a reason code which indicates whether that high-level qualifier matches a defined USER, a GROUP, or neither.

  Specify a DERIVE request for RESOWNER as follows:

```
RACROUTE Request= EXTRACT,
ENTITY=data set name,
VOLSER=mydasd,
CLASS=DATASET,
FIELDS='RESOWNER', SEGMENT='DFP',
DERIVE=YES, RELEASE=1.8.1
```

  Note: You must specify all the keywords in the example for the DERIVE request to work.

- USER

  The purpose of specifying the DERIVE=YES keyword with CLASS=USER is to obtain the desired DFP field information (STORCLAS or MGMTCLAS) from the profile of the user. If the user's profile does not contain the desired DFP fields, RACF then goes to the user's default group and attempts to obtain the information for the remaining fields from the GROUP profile (the remaining fields being those that did not contain information in the USER profile.)

  You would specify a DERIVE request for information from a USER profile as follows:

```
RACROUTE Request= EXTRACT,
ENTITY=user name,
CLASS=USER,
FIELDS='STORCLAS', SEGMENT='DFP',
DERIVE=YES, RELEASE=1.8.1
```

RACF only processes the DERIVE keyword if you specify it with the DATASET or USER class. In addition, for DERIVE processing to occur, you must also specify SEGMENT=DFP and RELEASE=1.8.1.

,FIELDS= *address*

Specifies the address of a variable length list. The first field is a 4-byte field that contains the number of profile field names in the list that follows. Each profile field name is 8 bytes long, left-justified, and padded to the right with blanks. The allowable field names for each type of profile are in the template listings in Chapter 3. To see how to specify the FIELDS keyword, see the TYPE=REPLACE example below.

- If you specify Release=1.6 or later, or allow the keyword to default, then the following options exist:
  - The only acceptable value of the count field is 1.
  - The only acceptable field name is PASSWORD. Use this parameter when you want to extract the user's encrypted password in addition to his/her userid and connect group. RACF returns the encrypted password in the result area at an offset from the start of the area specified by the halfword at offset 4. (See the result area under TYPE=EXTRACT.)
- If you specify Release=1.8 or later, then the following options exist:
  - The count field can contain numbers from 1 - 255.
  - The field names can be any of the field names in the the template listings.

If you specify TYPE=EXTRACT or EXTRACTN, RACF retrieves the contents of the named fields from the RACF profile indicated by the CLASS= and ENTITY= parameters, and returns the contents in the result area. (See result area explained under the EXTRACT keyword.)

With Release 1.8, you can specify TYPE=REPLACE. RACF replaces or creates the indicated fields in the profile specified on the CLASS and ENTITY keywords with the data pointed to by the SEGDATA keyword.

Notes:

1. Do not replace a repeat group count field. Doing so will cause unpredictable results.
2. You cannot replace an entire repeat group, a single occurrence of a repeat group, or a single existing field in a repeat group. If you attempt to do so, RACF adds the data to the existing repeat group(s).

   The only thing you can do is retrieve all occurrences of specified fields within a repeat group or add a new occurrence of a repeat group.
3. If you add occurrences of a repeat group, RACF places those additions at the beginning (front) of the repeat group.

The following example of TYPE=REPLACE replaces fields in the BASE segment. It shows one way to code the macro and the declares necessary to make the macro work.

```
RACXTRT  TYPE=REPLACE ,
         CLASS='USER',
         ENTITY=USERID,
         FIELDS=FLDLIST,
         SEGDATA=SEGDLIST,
         SEGMENT=BASE

         ..............................

USERID   DC  AL1(4), C'BILL'
FLDLIST  DC  A(3)
         DC  CL8'AUTHOR'
         DC  CL8'DFLTGRP'
         DC  CL8'NAME'
SEGDLIST DC  AL4(6),CL6'JSMITH'
         DC  AL4(8),CL8'SECURITY'
         DC  AL4(11),CL11'BILL THOMAS'
```

When the replacement action takes place, the following occurs:

- 'JSMITH' will be placed in the 'AUTHOR' field in the profile.
- 'SECURITY' will be placed in the 'DFLTGRP' field in the profile.
- 'BILL THOMAS' will be placed in the 'NAME' field in the profile.

,ENCRYPT =(*data addr***, DES)**
,ENCRYPT =(*data addr***, HASH)**
,ENCRYPT =(*data addr***, INST)**

specifies the data to be encrypted, and a method of encryption. The address points to a one-byte length field followed by 1 to 255 bytes of clear-text data to be encrypted. The second subparameter specifies the encryption method: the DES algorithm, the RACF hashing algorithm, or whatever scheme the installation uses (INST value). Upon return to the macro issuer, the first subparameter will now contain the address of an area that contains a one-byte length followed by the encrypted version of the data. Neither the address itself nor the length is changed.

Note: When you use the DES algorithm, RACF actually encrypts the data to which the ENTITY profile points, the userid, using the data as the encryption key. Data is one-way encrypted, that is, no facility is provided to recover the data in readable form. If you specify HASH, the RACF hashing algorithm is used and data is masked instead of encrypted.

,ENTITY=*resource name addr*

specifies the address of an area containing the resource name (USERID for CLASS=USER) for which profile data is to be extracted, or the userid to be used when encrypting. The area is 8 bytes long for USER and GROUP, 17 bytes long for CLASS=CONNECT, and 44 bytes long for DATASET. The class descriptor table determines the lengths of all other profile names. The name must be left-justified in the field and padded with blanks. If you do not specify this parameter, a default value of zero will indicate to RACF that the userid from the current ACEE will be used.

,RELEASE =1.6|1.7|1.8|1.8.1

specifies the RACF release level of the parameter list that this macro will generate.

To use the parameters associated with a release, specify the release number of that release or a later release number. If you specify an earlier release level, macro processing will not accept the parameter, and an error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 20 on page 620.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. To get execution-time validation of the compatibility between the list and execute forms of the RACXTRT macro, specify the CHECK subparameter on the execute form of the macro.

,ACEE = *acee addr*

specifies an alternate ACEE for RACF to use rather than the current ACEE. For example, if you do not specify the ENTITY parameter, RACF refers to the ACEE during extract processing of user data. If you want to use the ACEE parameter, you must specify RELEASE = 1.8 or later.

,VOLSER = *volser addr* **(only valid with ,CLASS = 'DATASET')**

specifies the volume serial as follows:

- For non-VSAM DASD data sets and tape data sets, specifies the volume serial number of the volume on which the data set resides.
- For VSAM DASD data sets and tape data sets, specifies the volume serial number of the catalog controlling the data set.

The field to which the vol-address points contains the volume serial number. If necessary, pad it to the right with blanks so it contain six characters.

If you specify VOLSER, you must specify RELEASE = 1.8 or later.

,GENERIC = ASIS|YES

When CLASS is DATASET, specifies whether RACF is to treat the entity name as a generic profile name.

- If you specify GENERIC = YES, RACF considers the entity name a generic profile name, even if it does not contain any of the generic characters (an asterisk or a percent sign).
- If you specify GENERIC = ASIS, RACF considers the entity name a generic only if it contains one or both of the generic characters.

If you specify GENERIC, you must specify RELEASE = 1.8 or later.

,FLDACC = NO|YES

Specifies whether field level access checking should be performed. If you specify FLDACC = YES, the RACF data base manager will check to see that the user running your program has the authority to extract or modify the fields that have been specified in the RACXTRT macro.

Notes:

1. For field level access checking to occur, you must specify RELEASE = 1.8 or later when you code the macro. In addition, before the program executes, the security administrator must activate the FIELD class. If these conditions are not satisfied, the RACF manager behaves as though you had specified FLDACC = NO.
2. In addition, the security administrator must issue the RDEFINE and PERMIT commands to designate those users who will have the authority to access the fields designated in the RACXTRT macro.
3. If you specify FLDACC = NO or omit the parameter, the manager ignores field level access checking.

,CLASS = *'class name'*

specifies the class the entity is in. The class name can be USER, GROUP, CONNECT, DATASET, or any general resource class defined in the class descriptor table. If you specify CLASS, you must specify RELEASE = 1.8 or later.

,SEGMENT = *'segment name'*

,SEGMENT = *segment name addr*

specifies the RACF profile segment that RACF is to update or from which it is to extract data. *segment name addr* is a fullword. If you specify SEGMENT, you must also specify the CLASS and FIELDS keywords, and RELEASE = 1.8 or a later release number. If you allow the SEGMENT parameter to default, RACF assumes that you want to extract information from the BASE segment.

,SEGDATA = *segment data addr*

specifies the address of a list of data items to be placed into the respective fields named by the FIELDS= parameter. Use the SEGDATA parameter when you specify TYPE=REPLACE. If you specify SEGDATA, you must also specify CLASS, FIELDS, and RELEASE=1.8 or a later release number. The stored data is paired in the following format:

- a 4-byte length field that contains the length of the data field that follows
- a data field of variable length

Each length field is followed immediately by a data field until you reach the end of the replacement data. The count field to which the first field in the FIELDS parameter points, contains the total number of length-data pairs.

Parameters for RELEASE=1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Figure 20. RACXTRT Parameters for RELEASE=1.6 and Later

Parameter	RELEASE=1.6	RELEASE=1.7	RELEASE=1.8 or 1.8.1
ACEE=			X
CLASS=			X
DERIVE=YES			X
ENCRYPT=	X	X	X
ENTITY=	X	X	X
EXTRACT=	X	X	X
EXTRACTN=			X
FLDACC=			X
FIELDS=	X	X	X
GENERIC=			X
RELEASE=	X	X	X
REPLACE=			X
SEGDATA=			X
SEGMENT=			X
SUBPOOL=	X	X	X
TYPE=	X	X	X
VOLSER=			X

Return Codes and Reason Codes

When control is returned, register 15 contains one of the following return codes, and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	The extraction or encryption completed successfully. **Reason code - For Derive requests** 0 - Some of the values are derived from the USER profile, and some may be derived from the GROUP profile. 4 - High-level qualifier returned as RESOWNER, and it matched a valid USER 8 - DFP data returned from an EXTRACT request from USER profile was actually from the user's default group. C - High-level qualifier returned as RESOWNER, and it matched a valid GROUP 24 - RESOWNER field matched a valid USER 28 - RESOWNER field matched a valid GROUP
04	An ESTAE environment was not able to be established, or if Register 0 contains a reason code of 1, you specified neither EXTRACT nor ENCRYPT for TYPE=.
08	For TYPE=EXTRACT, TYPE=EXTRACTN, or TYPE=REPLACE the profile could not be found. The hexadecimal reason codes are: 0 - No profile found 4 - Field level access checking failed 8 - Segment not found 14 - Neither the RESOWNER field nor the high level qualifier matched a valid USER or GROUP
C	RACF is inactive.
10	The extract operation failed. Register 0 contains the RACF manager return code which caused termination. This return code is not used for the encrypt function. The manager return code and reason codes are returned in the low order and high order half words of R0.
14	An ACEE address was not found when required, or if found, was not for a defined user. The hexadecimal reason codes are: 0 - No ACEE exists 4 - ACEERACF bit is off
18	A parameter list error was encountered. The hexadecimal reason codes are: 4 - No fields for request type REPLACE 8 - Invalid type specified C - Invalid number of fields 10 - Invalid class name specified 14 - Invalid version in parameter list 18 - Invalid subpool specified 1C - Invalid parameter length 20 - No segdata specified 24 - Invalid entity name specified 2C - No encryption data 30 - Invalid encryption method 34 - No ENTITY specified with TYPE=REPLACE or TYPE=EXTRACTN 38 - Multiple profiles no volume specified 3C - Profile found wrong volser specified
64	Indicates that you specified the CHECK subparameter of the RELEASE keyword on the execute form of the RACXTRT macro; however, the list form of the macro does not have the proper RELEASE parameter. It also indicates that the TYPE parameters specified on the list and execute forms may not be the same TYPE. Macro processing terminates.

RACXTRT (List Form)

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

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

TYPE=EXTRACT
TYPE=EXTRACTN
TYPE=REPLACE
TYPE=ENCRYPT

,ENTITY=*resource name addr*	*resource name addr:* A-type address.
,RELEASE=*number*	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12)
,VOLSER=*volser addr*	*volser addr :* A-type address,
,GENERIC=ASIS ,GENERIC=YES	**Default:** ASIS
,FLDACC=YES ,FLDACC=NO	**Default:** NO

,MF=L

If you specify TYPE=EXTRACT or EXTRACTN:

,SUBPOOL=*subpool number*	*subpool number:* decimal digit, 0-255 **Default:** SUBPOOL=229
,DERIVE='YES'	*see explanation of keyword* **Default:** normal processing
,CLASS='*class name*'	*class name*: the entity's class name **Default:** USER
,CLASS=*class name addr*	*class name addr*: A-type address or register (2) - (12)
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*'segment name':* 1-8 character name *segment name addr:* A-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr:* A-type address

If you specify TYPE=REPLACE:

,CLASS='*class name*'	*the entity's class name* **Default:** USER
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*'segment name':* 1-8 character name *segment name addr:* A-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr:* A-type address or register (2) - (12)
,SEGDATA=*segment data addr*	*segment data addr:* A-type address or register (2) - (12)

If you specify TYPE=ENCRYPT:

,ENCRYPT=(*data addr*,DES)
,ENCRYPT=(*data addr*,HASH)
,ENCRYPT=(*data addr*,INST)

data addr: A-type address

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

,MF=L
specifies the list form of the RACXTRT macro.

RACXTRT (Execute Form)

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

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

TYPE=EXTRACT TYPE=ENCRYPT	
,ENTITY=*resource name addr*	*resource name addr:* RX-type address or register (2)-(12)
,RELEASE=*(number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.8.1, 1.8, 1.7, or 1.6 **Default:** RELEASE=1.6
,ACEE=*acee addr*	*acee addr:* RX-type address or register (2) - (12)
,VOLSER=*volser addr*	*volser addr:* RX-type address or register (2) - (12)
,GENERIC=ASIS ,GENERIC=YES	**Default:** ASIS
,FLDACC=YES ,FLDACC=NO	**Default:** NO
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address, register (1), or register (2)-(12)
If you specify TYPE=EXTRACT or EXTRACTN:	
,SUBPOOL=*subpool number*	*subpool number:* decimal digit, 0-255 **Default:** SUBPOOL=229
,DERIVE='YES'	*see explanation of keyword* **Default:** normal processing
,CLASS=*class name*	*the entity's class name* **Default:** USER
,CLASS=*class name addr*	*class name addr*: RX-type address or register (2) - (12)
,SEGMENT=*segment name addr* ,FIELDS=*field addr*	*segment name addr:* RX-type address or register (2) - (12) *field addr:* RX-type address or register (2)-(12)
If you specify TYPE=REPLACE:	
,CLASS=*class-name*	*the entity's class name* **Default:** USER
,SEGMENT='*segment name*' ,SEGMENT=*segment name addr*	*'segment name'*: 1-8 character name *segment name addr*: RX-type address or register (2) - (12)
,FIELDS=*field addr*	*field addr:* RX-type address or register (2) - (12)
,SEGDATA=*segment data addr*	*segment data addr:* RX-type address or register (2) - (12)
If you specify TYPE=ENCRYPT:	
,ENCRYPT=(*data address*,DES) ,ENCRYPT=(*data address*,HASH) ,ENCRYPT=(*data address*,INST)	*data address:* RX-type address or register (2)-(12)

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

,RELEASE =(*number*,CHECK)
,RELEASE= *number*
,RELEASE= (,CHECK)
specifies the RACF release level of the parameter list that this macro will generate.

You can specify certain parameters only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 20.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. To get execution-time validation of the compatibility between the list and execute forms of the RACXTRT macro, specify the CHECK subparameter on the execute form of the macro.

When you request CHECK processing and the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be generated. Instead, RACXTRT will generate a return code X'64'.

,MF=(E,*ctrl addr*)
specifies the execute form of the RACXTRT macro using a remote control program parameter list.

RESERVE — Reserve a Device (Shared DASD)

The RESERVE macro reserves a device for use by a particular system; it must be issued by each task needing device reservation. The RESERVE macro protects the issuing task from interference by other tasks in the system and locks out other systems. The reserve actually occurs when the first I/O is done to the device after the RESERVE macro is issued. When the reserving program no longer needs the reserved device, it should issue a DEQ macro to release the resource. For information about how to obtain the UCB address for a device, see the section "Finding the UCB Address for the RESERVE Macro" in *SPL: Application Development Guide*.

If a task issues two RESERVE instructions for the same device without an intervening DEQ, an abnormal termination results unless the second RESERVE specifies the keyword parameter RET= or ECB=. (If a restart occurs when a RESERVE is in effect for resources, the system does not restore the RESERVE; the user's program must reissue the RESERVE.) If a DEQ is not issued for a particular resource, termination routines release resources reserved by a terminating task.

If global resource serialization is active, the hardware RESERVE can be suppressed leaving a SYSTEMS ENQ depending on the contents of the resource name lists. See *Planning: Global Resource Serialization* for information on resource name lists.

Global resource serialization counts and limits the number of concurrent resource requests in an address space. If an unconditional RESERVE (a RESERVE that uses the RET=NONE option) causes the count of global resource serialization requests to exceed the sum of a threshold value plus a tolerance value, an authorized caller is abended with a system code of X'538'. See "Limiting Global Resource Serialization Requests" in *SPL: Application Development Guide*.

The ECB parameter is restricted in use to callers in supervisor state, PSW key 0-7, or with APF authorization. Except for the UCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

A RESERVE used with the MASID and MTCB operands provides a special form of the RESERVE macro that allows a further conditional control of a resource. One task, called the "issuing task" can issue a RESERVE macro for a resource specifying the ASID and TCB of another task, called the "matching task". The MTCB and MASID operands are specified with RET=HAVE and ECB= to provide additional return codes. If the issuing task does not acquire control of the resource, it may receive a return code indicating that the resource is controlled by the matching task. Upon receiving this return code, the issuing task could use the resource, if serialization between itself and the matching task has been accomplished by some pre-arranged protocol known to both the issuing and matching tasks.

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

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

(	
qname addr	*qname addr:* A-type address, or register (2) - (12).
,*rname addr*	*rname addr:* A-type address, or register (2) - (12).
,	**Default:** E
,E	
,S	
,	
,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12).
,SYSTEMS	
)	
,RET=TEST	
,RET=USE	
,RET=HAVE	
,RET=NONE	
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr:* A-type address, or register (2) - (12).
,MASID=*matching-asid addr*	*matching-asid addr:* A-type address, or register (2) - (12).
,MTCB=*matching-tcb addr*	*matching-tcb addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

(
specifies the beginning of the resource description.

qname addr
specifies the address in virtual storage of an 8-character name. The name should not start with SYS, so that it will not conflict with system names. Every task issuing RESERVE against the same resource must use the same qname and rname to represent the resource.

,rname addr
specifies the address in virtual storage of the name used in conjunction with qname to represent a single resource. The name can be qualified, and must be from 1 to 255 bytes long.

,
,E
,S
specifies whether the request is for exclusive (E) or shared (S) control of the resource. If the resource is modified while under control of the task, the request must be for exclusive control; if the resource is not modified, the request should be for shared control.

,

,rname length

specifies the length of the *rname* described above. If this parameter is omitted, the assembled length of the *rname* is used. You can specify a value between 1 to 255 to override the assembled length, or you may specify a value of 0. If 0 is specified, the length of the *rname* must be contained in the first byte at the *rname addr* specified above.

,SYSTEMS

specifies that the resource is shared among systems.

)

specifies the end of the resource description.

,RET = TEST
,RET = USE
,RET = HAVE
,RET = NONE

RET = TEST, RET = USE, and RET = HAVE specify a conditional request for all the resources named above, as follows:

- RET = TEST - the availability of the resources is to be tested, but control of the resources is not requested.
- RET = USE - control of the resources is to be assigned to the active task only if the resources are immediately available.
- RET = HAVE - control of the resources is requested only if a request has not been made previously for the same task.

RET = NONE specifies an unconditional request for all the resources named above.

,ECB = *ecb addr*

specifies the address of an ECB, and conditionally requests the resource named in the macro. If the return code for one or more requested resources is 4 and the request is not nullified by a corresponding DEQ, the ECB is posted when all the requested resources (specifically, those that initially received a return code of 4) are assigned to the requesting task.

,UCB = *ucb addr*

specifies the address of a fullword that contains the address of the UCB for the device to be reserved. The UCB must be allocated to the job step before RESERVE is issued unless the issuer is in supervisor state, system key, or APF-authorized.

,MASID = *matching-asid addr*

specifies the matching task (by defining a matching ASID) for the RESERVE, if used in conjunction with the MTCB parameter. MASID defines the ASID of a task that may be using a resource desired by the issuer of the RESERVE macro.

Note: MASID can only be specified if MTCB is also specified.

,MTCB = *matching-tcb addr*

specifies the matching task (by defining a matching TCB) for the RESERVE, if used in conjunction with the MASID parameter. MTCB defines the TCB of a task that may be using a resource desired by the issuer of the RESERVE macro.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the issuer of the RESERVE and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

The MASID and MTCB parameters are specified with RET = HAVE, RET = TEST, and/or ECB = parameters to elicit additional return codes that provide information about the owner of the resource.

Note: MTCB can only be specified if MASID is also specified.

,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 codes are provided by the control program only if you specify RET = TEST, RET = USE, RET = HAVE, or ECB = ; otherwise, return of the task to the active condition indicates that control of the resource has been assigned to the task. If return code for the resource named in the RESERVE macro is 0, register 15 contains 0. If the return code is not 0, register 15 contains the address of a storage area containing the return code, as shown in Figure 21.

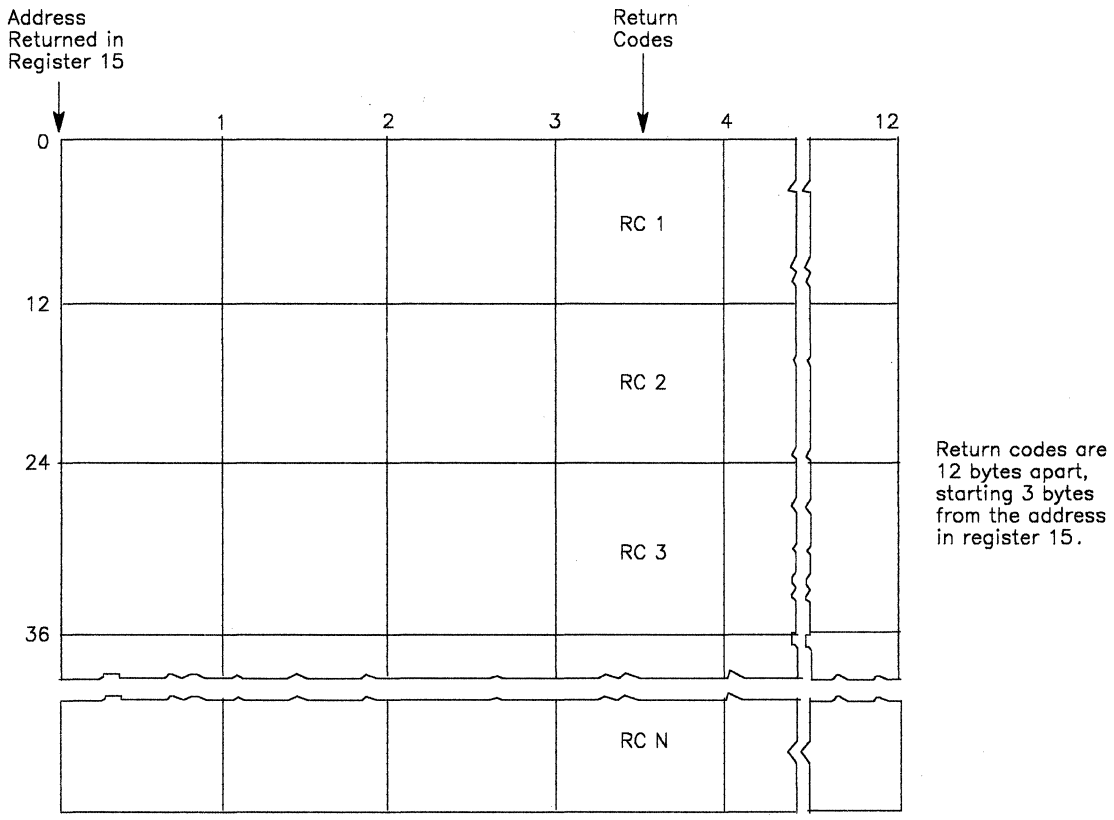


Figure 21. Return Code Area Used by RESERVE

The return code is placed in the parameter list resulting from the macro expansion. The return codes are shown below.

Hexadecimal Code	Meaning
0	For RET = TEST, the resource was immediately available. For RET = USE, RET = HAVE, or ECB = , control of the resource has been assigned to the active task.
4	For RET = TEST or RET = USE, the resource is not immediately available. For ECB = , the ECB will be posted when available.
8	A previous request for control of the same resource has been made for the same task. Task has control of resource. If bit 3 is on - shared control of resource; if bit 3 is off - exclusive control.
14	A previous request for control of the same resource has been made for the same task. Task does not have control of resource.

18 For RET=HAVE, RET=USE, or ECB=, the limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. The ECB is not posted.

20 The matching task (the task specified in the MASID/MTCB parameters) owns the resource. The issuer of the RESERVE macro may use the resource but it must ensure that the owning task does not terminate while the issuing task is using the resource. If the issuing task requested exclusive control then this return code indicates that the matching task is the only task that currently owns the resource. If the issuer of the RESERVE requested shared control and the owning task had requested shared control, this return code may indicate that a previous task had requested exclusive control. The issuing task must issue a DEQ to cancel this RESERVE. The ECB will not be posted.

24 The issuing task will have exclusive control after the ECB is posted. The issuing task may use the resource but must ensure that the matching task does not terminate while the issuing task is using the resource. The issuing task must issue a DEQ to cancel the RESERVE.

28 The issuing task cannot obtain exclusive control of the resource using the MASID/MTCB RESERVE. The matching task's involvement with other tasks precludes control by the issuing task. This task must not issue a DEQ to cancel the RESERVE. The ECB will not be posted.

44 The issuing task is violating a restriction of the MASID/MTCB RESERVE in one or more of the following ways:

- Another task has already issued this RESERVE for this resource specifying the same MASID/MTCB.
- The MASID/MTCB parameters specify a task that acquired control of the resource by using the MASID/MTCB RESERVE.
- The matching task requested ownership of the resource but has not yet been granted ownership.

The ECB will not be posted. Return code 44 is never given by a RESERVE RET=TEST, return code 4 is given instead.

Example

Operation: Unconditionally reserve exclusive control of a device. The length of the rname is allowed to default.

```
RESERVE (MAJOR3,MINOR3,E,,SYSTEMS),UCB=(R3)
```

RESERVE (List Form)

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

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

ƀ                                One or more blanks must precede RESERVE.

RESERVE

ƀ                                One or more blanks must follow RESERVE.
```

```
(
   qname addr                    qname addr: A-type address.

   ,                             rname addr: A-type address.
   ,rname addr

   ,
   ,E
   ,S

   ,                             rname length: symbol or decimal digit.
   ,rname length

   ,
   ,SYSTEMS
)

   ,RET=TEST
   ,RET=USE
   ,RET=HAVE
   ,RET=NONE

   ,ECB=ecb addr                 ecb addr: A-type address.

,UCB=ucb addr                    ucb addr: A-type address or 0.

   ,MASID=0
   ,MTCB=0

   ,RELATED=value                value: A-type address.

,MF=L
```

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

,MF=L
specifies the list form of the RESERVE macro.

Note: If you specify the ECB parameter on the execute form of the macro, you must also specify it on the list form of the macro. The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB are specified, these labels are offset; allowance must be made for the parameter list prefix.

RESERVE (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RESERVE.
RESERVE	
b	One or more blanks must follow RESERVE.
(	**Note:** (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired, the (,), and all parameters between (and) should not be specified. If something in the list is desired, then (,), and all parameters in the list should be specified as indicated at the left.
qname addr	*qname addr:* RX-type address, or register (2) - (12).
, ,*rname addr*	*rname addr:* RX-type address, or register (2) - (12).
, ,E ,S	
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12). **Note:** *rname length* must be coded if a register is specified for *rname addr* above.
, ,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE ,RET=NONE	
,ECB=*ecb addr*	*ecb addr:* RX-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr:* RX-type address, or register (2) - (12).
,MASID=*matching-asid addr*	*matching-asid addr:* A-type address, or register (2) - (12).
,MTCB=*matching-tcb addr*	*matching-tcb addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E, *ctrl addr*)	*ctrl addr:* RX-type address, or register (1) - (12).

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

,MF=(E,*ctrl addr)*
specifies the execute form of the RESERVE macro using a remote control program parameter list.

Note: If the ECB parameter is specified on the execute form of the macro, it must also be specified on the list form of the macro. If MASID and MTCB are specified, MASID=0 and MTCB=0 must be specified in the list form.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB are specified, these labels are offset; allowance must be made for the parameter list prefix.

RESMGR - Add or Delete Resource Manager

RESMGR adds or deletes a resource manager that receives control whenever a task or a dynamic address space terminates. To invoke RESMGR, you must be enabled, in supervisor state, and unlocked. You can be in 24-bit or 31-bit addressing, and in primary or access register mode.

When you invoke RESMGR in access register mode, the parameter list must be in the primary address space and it must be qualified by an ALET of 0.

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

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

ADD DELETE	
,TOKEN=*tokaddr*	*tokaddr:* A-type address, or register (2) - (12).
,TYPE=ADDRSPC ,TYPE=TASK	
,ASID=CURRENT ,ASID=ALL ,ASID=*asid*	*asid*: A constant or register (2) - (12).
,TCB=CURRENT ,TCB=ALL ,TCB=*tcbaddr*	*tcbaddr*: A-type address or register (2) - (12).
,ROUTINE=(LINK, *pgname*) ,ROUTINE=(BRANCH, *pgaddr*) ,ROUTINE=(PC, *pcnum*)	*pgname*: C-type constant, A-type address, or register (2) - (12). *pgaddr*: A-type address, or register (2) - (12). *pcnum*: a constant or register (2) - (12).
,ECB=*ecbaddr*	*ecbaddr*: A-type address, or register (2) - (12).
,PARAM=*paddr*	*paddr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

The parameters are explained as follows:

ADD
DELETE
specifies whether a resource manager is to be added or deleted. You must specify the same values on TYPE, TCB and ASID on DELETE as you specified on those parameters on ADD. On DELETE, you must specify the token that ADD returned so the system can identify the resource manager that you want to delete. TOKEN has no default. You must specify either ADD or DELETE unless you specify MF=(E,list-addr) to indicate the execute form of RESMGR.

,TOKEN=*tokaddr*
specifies the address of the full word where you want the system to store the token that it returns after an ADD. The token represents the resource manager that the system added. On DELETE, however, you store the token in this full word before invoking the delete function. TOKEN is required.

,TYPE = ADDRSPC
,TYPE = TASK
specifies whether the resource manager is an address space resource manager (ADDRSPC) or a task resource manager (TASK). The default is address space.

,ASID = CURRENT
,ASID = ALL
,ASID = *asid*
specifies the id of the address space that is to be monitored for termination. If you specify ALL, the system monitors all address spaces. If you specify CURRENT, the system monitors the home address space. If you specify an *asid*, the system monitors only that address space.

,TCB = CURRENT
,TCB = ALL
,TCB = *tcbaddr*
specifies the TCB address of the task that is to be monitored for termination. If you specify ALL, the system monitors all tasks in the specified address space. If you specify CURRENT, the system monitors the current task. If you specify a *tcbaddr*, the system monitors only that task. If a specific task is to be monitored, the TCB you specify must be in the home address space. A program in SRB mode cannot issue TCB = CURRENT.

,ROUTINE = (LINK, *pgname***)**
,ROUTINE = (BRANCH, *pgaddr***)**
,ROUTINE = (PC, *pcnum***)**
specifies:

- the type of linkage to be used by the system when giving control to the resource manager program.
- the resource manager program to receive control when the task or address space terminates.

The resource manager always receives control in 31-bit addressing mode and in primary ASC mode. Resource managers never receive control in cross memory mode. The chapter on resource management in *SPL: Application Development Guide* describes the registers on entry, the resource manager parameter list (RMPL), and some of the responsibilities of the resource manager.

If you specify PC, the resource manager receives control through a PC instruction. *pcnum* is the PC number of the PC instruction that gives control to the resource manager. The address space in which the termination occurs must have the authority to issue PC. That is, the PC number must be globally defined.

If you specify BRANCH, the resource manager receives control through a branch instruction and the resource managers (whether address space termination resource managers or task termination resource managers) must reside in common storage.

If you specify LINK, the resource manager must reside in a link list library.

If you specify ADD, you must specify ROUTINE.

,ECB = *ecbaddr*
The processing to delete a resource manager might not be complete when RESMGR returns. If you require notification after DELETE has completed, code ECB. The ECB will be posted when DELETE is completed. Note, however, that DELETE may already be complete upon return, in which case the system does not post any completion ECB. Check the return code from RESMGR before you wait on the ECB.

The system associates the completion ECB with the home address space of the DELETE requestor. If you specify ECB, you must also specify DELETE. You must also specify either of the following when you specify ECB:

- TYPE = ADDRSPC and ASID = ALL
- TYPE = TASK and ASID = ALL and TCB = ALL.

,PARAM = *paddr*
specifies the address of parameter data to be used by the resource manager when it receives control. The parameter data must reside in the caller's primary address space. PARAM is valid only with ADD.

,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 codes from the ADD function are as follows:

Figure 22. Return codes from ADD

Return Code	Meaning
0	The resource manager was successfully established. The word provided by the TOKEN parameter contains the token required to delete the resource manager.
12	The resource manager was not established. The caller did not provide the address of a word to contain the token of the new resource manager.
16	The resource manager was not established. The caller did not provide the resource manager description via the ROUTINE parameter.
20	The resource manager was not established. The TCB address provided did not represent a valid TCB.
24	The resource manager was not established. The ASID provided did not represent a valid ASCB.
28	The resource manager was not established. The request type was not ADD or DELETE.
32	The resource manager was not established. The dynamic resource manager service was unable to obtain storage for a work area it needed in order to process the request.
36	The resource manager was not established. A lock was held on entry to the dynamic resource manager service routine.
40	The resource manager was not established. It is invalid to establish a task resource manager for a specific task which is not in the home address space of the requestor.
44	AN unrecoverable error occurred while processing the request.
48	The resource manager was not established. The dynamic resource manager service was unable to obtain storage for a resource manager element needed to maintain information about the dynamic resource manager.
52	The resource manager was not established. The caller is not authorized to use the RESMGR function.
56	The resource manager was not established. The TCB was already in termination and no more dynamic resource managers can be established for it.
60	The resource manager was not established. The ASCB was already in termination and no more dynamic resource managers can be established for it.

Return codes from the DELETE function are as follows:

Figure 23. Return codes from DELETE

Return Code	Meaning
0	The resource manager was successfully deleted. An ECB is never posted for this return code.
4	The resource manager is currently in use. It has been queued for deletion. The ECB, if provided, will be posted when the delete process has completed.
8	The resource manager was queued for deletion by a previous request. It is still active and will be deleted as soon as it is no longer in use.
12	The resource manager was not deleted. The caller did not provide a token to the dynamic resource manager service.
16	The resource manager was not deleted. The token provided did not represent a currently established resource manager.
20	The resource manager was not deleted. The TCB address provided did not represent a valid TCB.
24	The resource manager was not deleted. The ASID provided did not represent a valid ASCB.
28	The resource manager was not deleted. The request type was not ADD or DELETE.
32	The resource manager was not deleted. The dynamic resource manager service was unable to obtain storage for a work area it needed in order to process the request.
36	The resource manager was not deleted. A lock was held on entry to the dynamic resource manager service routine.
40	The resource manager was not deleted. It is invalid to delete a task resource manager for a specific task that is not in the home address space of the requestor.
44	An unrecoverable error occurred while processing the request.
48	The resource manager was not deleted. The ECB parameter was specified but is not supported for the particular type of delete request.
52	The resource manager was not deleted. The caller is not authorized to use the RESMGR function.

Example

Operation: The following RESMGR macros establish a resource manager that receives control for every address space termination and every task termination. This resource manager is equivalent to having included the name IAMARESM in the IEAVTRML table.

```
RESMGR ADD,TYPE=ADDRSPC,ASID=ALL,
   ROUTINE=(LINK,'IAMARESM')
RESMGR ADD,TYPE=TASK,ASID=ALL,TCB=ALL,
   ROUTINE=(LINK,'IAMARESM')
```

RESMGR (List Form)

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

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

ADD DELETE	
,TOKEN=*tokaddr*	*tokaddr:* A-type address, or register (2) - (12).
,TYPE=ADDRSPC ,TYPE=TASK	
,ASID=CURRENT ,ASID=ALL ,ASID=*asid*	*asid*: a constant.
,TCB=CURRENT ,TCB=ALL ,TCB=*tcbaddr*	*tcbaddr*: A-type address
,ROUTINE=(LINK, *pgname*) ,ROUTINE=(BRANCH, *pgaddr*) ,ROUTINE=(PC, *pcnum*)	*pgname*: C-type constant or A-type address. *pgaddr*: A-type address. *pcnum*: A constant.
,ECB=*ecbaddr*	*ecbaddr*: A-type address.
,PARAM=*paddr*	*paddr*: A-type address.
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the RESMGR macro.

RESMGR (Execute Form)

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

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

ADD DELETE	
,TOKEN=*tokaddr*	*tokaddr:* A-type address, or register (2) - (12).
,TYPE=ADDRSPC ,TYPE=TASK	
,ASID=CURRENT ,ASID=ALL ,ASID=*asid*	*asid*: A constant or register (2) - (12).
,TCB=CURRENT ,TCB=ALL ,TCB=*tcbaddr*	*tcbaddr*: RX-type address or register (2) - (12).
,ROUTINE=(LINK, *pgname*) ,ROUTINE=(BRANCH, *pgaddr*) ,ROUTINE=(PC, *pcnum*)	*pgname*: a C-type constant, RX-type address, or register (2) - (12). *pgaddr*: RX-type address, or register (2) - (12). *pcnum*: A constant, an expression, or register (2) - (12).
,ECB=*ecbaddr*	*ecbaddr*: RX-type address, or register (2) - (12).
,PARAM=*paddr*	*paddr*: RX-type address, or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=(E,*listaddr*)	*listaddr*: RX-type address, or register (2) - (12).

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

,MF=(E,*listaddr*)
E specifies the execute form of the RESMGR macro and *listaddr* specifies the address of the parameter list.

RESUME — Resume Execution of a Suspended Request Block

The RESUME macro causes suspended RBs to resume execution. Callers in AR mode can issue RESUME. However, the caller must have current addressability to the address space of the task being resumed. That is, the address space must be the current address space. For restrictions on using the RESUME macro, see "Resuming Execution of a Suspended Request Block" in *SPL: Application Development Guide*.

The RESUME macro is coded as follows:

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

TCB=(4) TCB=*tcbaddr*	**Default:** TCB address contents of register (4). *tcbaddr:* A-type address or registers (2) - (12).
,RB=(5) ,RB=*rbaddr*	**Default:** RB address contents of register (5). *rbaddr:* A-type address or registers (2) - (12).
,RETURN=Y ,RETURN=N	**Default:** RETURN=Y
,MODE=UNCOND ,MODE=COND	**Default:** MODE=UNCOND.
,ASYNC=Y ,ASYNC=N	**Default:** ASYNC=N
,ASCB=*ascbaddr*	**Default:** ASCB address of the home address space. *ascbaddr:* RX-type address or registers (1) or (2) - (3) or (6) - (12).

The parameters are explained as follows:

TCB=(4)
TCB=*tcbaddr*
specifies the TCB address of the task to be resumed. Register (4) is the default; it is assumed to contain the TCB address.

Note: The TCB resides in storage below 16 megabytes.

,RB=(5)
,RB=*rbaddr*
specifies the address of the RB to be resumed. Register (5) is the default; it is assumed to contain the address of the RB to be resumed. The RB operand determines which RB will have its suspend count decremented (that is, which RB will be made ready for resumption of execution).

Note: The RB resides in storage below 16 megabytes.

,RETURN=Y
,RETURN=N
specifies whether control is to be returned to the caller (RETURN=Y) or not (RETURN=N). RETURN=N causes RESUME to make the specified TCB/RB dispatchable and gives the specified TCB/RB control directly. Only programs running under an SRB in primary ASC mode can issue RETURN=N. If you specify RETURN=N, you must also specify MODE=UNCOND and ASYNC=N and must not specify ASCB.

,MODE = UNCOND
,MODE = COND

If MODE = COND is specified, the action RESUME takes if the function cannot be completed synchronously depends on the ASYNC option. If ASYNC = Y is specified, RESUME makes a conditional attempt to acquire an SRB. If an SRB is available, it is scheduled to complete the RESUME function asynchronously. If ASYNC = N is specified explicitly or as a default, and the RESUME cannot immediately complete the function, it places return code 04 in register 15 and returns to the caller.

If MODE = UNCOND is specified, the action RESUME takes also depends on the ASYNC option. If ASYNC = Y is specified, RESUME makes an unconditional request for an SRB and completes the RESUME function asynchronously. If ASYNC = N is specified explicitly or as a default, RESUME unconditionally obtains the CML lock of the ASCB whose TCB or RB is to be resumed. The TCB or RB is resumed before control returns to the caller.

,ASYNC = Y
,ASYNC = N

specifies whether the RESUME is to be completed asynchronously (Y) or not (N).

,ASCB = *ascbaddr*

specifies the address of the ASCB whose TCB or RB is to be resumed. The caller must establish current addressability to the address space before calling RESUME. If this option is not specified, the home address space is assumed. This option must be specified if ASYNC = Y is specified.

Note: The ASCB resides in storage below 16 megabytes.

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

Hexadecimal Code	Meaning
00	A normal, synchronous RESUME completed the function.
04	For MODE = COND and ASYNC = N, the RESUME cannot complete the function. For MODE = COND or MODE = UNCOND and ASYNC = Y, an SRB is completing the function asynchronously.
08	For MODE = COND and ASYNC = Y the SRB pool is empty and RESUME cannot complete the function.

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 - 3	Unchanged
4 - 5	Used as work registers by the macro
6 - 10	Unchanged
11 - 14	Used as work registers by the macro
15	Return code if RETURN = Y was specified; otherwise, used as a work register by the macro.

Example

Operation: Resume execution of the task specified in the address labeled CURRTCB. Use the request block address in register 5. Pass control back to the task (the issuer is currently in SRB mode, and this step terminates SRB mode processing).

```
RESUME  TCB=CURRTCB,RB=(5),RETURN=N
```

RISGNL — Issue Remote Immediate Signal

The RISGNL macro uses the emergency signal (EMS) order code of the signal processor (SIGP) instruction to invoke the execution of a specified software program on a specific processor in a multiprocessing configuration. The program may be requested to execute in parallel or serially with the function requesting the program. The specified software program (receiving routine) gets control disabled, in key 0, and supervisor state. The receiving routine cannot enable, request locks, or issue SVCs. In addition, the receiving routine must return via the address in register 14. The RISGNL macro can be invoked by programs executing in cross memory mode.

Additional SIGP order codes are available via the DSGNL macro. See *Principles of Operation* for an explanation of the order codes.

The RISGNL macro is written as follows:

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

PARALLEL SERIAL	
,CPU=*PCCA addr*	*PCCA addr:* RX-type address, or register (1).
,EP=*entry name addr*	*entry name addr:* RX-type address, or register (12).
,PARM=*parm addr*	*parm addr:* RX-type address, or register (11).

The parameters are explained as follows:

PARALLEL
SERIAL
specifies that control is to be returned to the caller when the specified receiving routine has been given control (PARALLEL) or has completed execution (SERIAL) on the designated processor.

,CPU=*PCCA addr*
specifies the address of the physical configuration communication area (PCCA) of the processor on which the function is to be performed.

Note: The PCCA must reside in 24-bit addressable storage.

,EP=*entry name addr*
specifies the address of the receiving routine to be executed on the specified processor. The receiving routine will get control in the same addressing mode as the macro issuer.

,PARM=*parm addr*
specifies the address of a user-defined fullword parameter to be passed to the receiving routine. When the receiving routine receives control, general purpose register one points to a fullword parameter.

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

Hexadecimal Code	Meaning
00	Specified receiving routine has been given control or has completed execution, as requested.
04	Function not initiated because addressed processor not online. If it appeared to be online, it is no longer in the configuration.
14	Processor alive bit was turned off during the remote immediate window spin routine.

Example 1

Operation: The routine whose address is in register 12 is to be given control on the processor whose PCCA address is in register 1. Return control to the caller when the specified receiving routine has been given control.

```
RISGNL PARALLEL,CPU=(1),EP=(12)
```

Example 2

Operation: The routine whose address is in register 12 is to be given control on the processor whose PCCA address is in register 1. The routine will complete before the caller of RISGNL receives control again. Register 11 contains the address of a parameter to be passed to the receiving routine.

```
RISGNL SERIAL,CPU=(1),EP=(12),PARM=(11)
```

SCHEDULE — Schedule System Services for Asynchronous Execution

The SCHEDULE macro schedules system services for asynchronous execution. These services may be scheduled for execution in any address space and may be scheduled at either global or local priorities.

Services scheduled at a global priority have a priority that is greater than, and independent of, any address space priority. Services scheduled at a local priority have the priority of the specific address space they execute in, but still have a priority greater than that of any task within the address space. To use SCHEDULE you must be in supervisor state, PSW key zero.

The addressing mode of the SRB routine is specified in the SRBEP field of the SRB control block. The user is required to set the correct AMODE. If bit 0 of the SRBEP field is set to 1, the SRB gets control in 31-bit addressing mode; if bit 0 is set to 0, the SRB routine gets control in 24-bit addressing mode. The addressing mode of the SRB's FRR is specified in the SRBFRRA field of the SRB control block. The user is required to set the correct AMODE. If bit 0 of the SRBFRRA field is set to 1, the FRR (and its retry routine) get control in 31-bit addressing mode. If bit 0 of the SRBFRRA field is set to 0, the FRR (and its retry routine) get control in 24-bit addressing mode.

Programs executing in primary ASC mode or access register ASC mode, including programs that are in cross memory mode, can issue the SCHEDULE macro.

For information on using this macro on an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.

The SCHEDULE macro is written as follows:

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

SRB=*SRB addr*	*SRB addr:* RX-type address, or register (1) or (2) - (12).
,SCOPE=LOCAL ,SCOPE=GLOBAL	**Default:** SCOPE=LOCAL
,LLOCK=YES ,LLOCK=NO	**Default:** LLOCK=NO
,FRR=YES ,FRR=NO	**Default:** FRR=NO
,DISABLED	

The parameters are explained as follows:

SRB=*SRB addr*
specifies the address of the service request block (SRB).

,SCOPE=LOCAL
,SCOPE=GLOBAL
specifies whether the service is to be scheduled at a local or global priority.

,LLOCK = YES
,LLOCK = NO
specifies whether the SRB is to receive control with the LOCAL lock held.

Note: CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

,FRR = YES
,FRR = NO
specifies whether the SRB is to receive control with recovery established. If FRR = YES is specified, the user must include in the SRB field (SRBFRRA) the FRR exit address. When the SRB receives control, the FRR will have been added to the FRR stack. When FRR = YES is specified, the 24 byte FRR parameter area address will be passed to the SRB routine in register 2.

,DISABLED
specifies that the calling program is running disabled. DISABLED should be specified only when the calling program is physically disabled for interrupts.

Example 1
Operation: Schedule an SRB at a global priority.

```
SCHEDULE SRB=(1),SCOPE=GLOBAL
```

Example 2
Operation: Schedule an SRB at a local priority.

```
SCHEDULE SRB=(1),SCOPE=LOCAL
```

Example 3
Operation: Schedule an SRB at a global priority specifying that the SRB is to receive control with the LOCAL lock held and recovery established. The issuer of the SCHEDULE macro is disabled.

```
SCHEDULE SRB=(1),SCOPE=GLOBAL,LLOCK=YES,FRR=YES,DISABLED
```

SCHEDXIT — Schedule an Exit Routine for Execution

The SCHEDXIT macro schedules an asynchronous exit routine for execution under a specific task. To use asynchronous exit routines, the caller must complete a three-stage process that consists of:

1. Identifying the exit routine to the system by initializing an interrupt request block (IRB)
2. Scheduling the exit routine for execution by initializing an interrupt queue element (IQE)
3. Adding the exit routine's IRB block to the specified task's dispatching queue.

Before using this macro, read the description of the three-stage process for asynchronous exit routines in *SPL: Application Development Guide*.

To schedule an exit routine for execution, the caller must invoke the exit effector through one of two methods: using SCHEDXIT, or branching directly to the exit effector. The method that the caller should use depends primarily on where the IQE resides:

- When the IQE resides in 31-bit storage, the caller must use SCHEDXIT.
- When the IQE resides in 24-bit storage, the caller can use SCHEDXIT only if the IQE address passed is a clean 31-bit address (that is, the high-order byte of the address is zero). Otherwise, the caller must use branch entry to the exit effector to schedule the routine for execution.

The requirements for the caller are:

Authorization:	Supervisor state with PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	31-bit
ASC mode:	Primary
Serialization:	Must hold the local lock
Control parameters:	None

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	True (non-complemented) IQE address
2 - 13	Unchanged
14 - 15	Used as work registers by the macro

The standard form of the SCHEDXIT macro is written as:

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

IQE=*iqe-address*	*iqe-address:* RX-type address or register (2) - (12).

The parameter is explained as follows:

IQE= *iqe-address*
specifies the address of the interrupt queue element (IQE) that defines the task under which the exit routine will execute.

SDUMP and SDUMPX — Dump Virtual Storage

The SDUMP macro invokes SVC dump to provide a fast unformatted dump of virtual storage to a data set. It is intended for use by authorized routines that encounter errors.

If your program is in primary ASC mode, you can use either SDUMP or SDUMPX. If your program runs in access register (AR) mode, use SDUMPX instead of SDUMP. SDUMPX provides all of the function of SDUMP but generates code and addresses that are appropriate for AR mode. Make sure that the SYSSTATE ASCENV=AR macro has been issued to let SDUMP know that the caller is in AR mode.

Descriptions of SDUMP and SDUMPX in this book are:

- The standard form of the SDUMP macro

  Includes general information about the SDUMP and SDUMPX macros, with some specific information about the SDUMP macro. The syntax of the SDUMP macro is presented, and all SDUMP parameters are explained.

- The standard form of the SDUMPX macro

  Includes information specific to the SDUMPX macro. The syntax of the SDUMPX macro is presented and parameters that are valid only on the SDUMPX macro are described.

- The list form of the SDUMP and SDUMPX macros

  Comments in the syntax identify parameters that are not valid for certain ASC modes.

- The execute form of the SDUMP and SDUMPX macros

  Comments in the syntax identify parameters that are not valid for certain ASC modes.

The SDUMP macro cannot dump data space storage. To dump data space storage, issue SDUMPX and include either the LISTD or SUMLISTL parameter.

For information about how to select this macro for an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.

Except for the DCB, all input parameters to this macro can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.

SVC dump is available only to authorized programs. Issuers of SDUMP with entry by SVC must be authorized via APF or have a PSW key 0-7. Branch entry callers must be key zero, supervisor state, and must be in SRB mode, or own a lock, or be disabled (with supervisor bit on) or be in enabled unlocked task FRR mode.

The caller can initiate an SVC dump in an address space other than the primary. A branch entry is available for callers who wish a dump of their own or another address space, but cannot issue an SVC.

When SVC dump is entered, the specified parameter list and all areas the list points to (except the DCB and ECB) must be in the currently addressable primary address space. If the caller is in access register mode, the parameter list address must be qualified by an ALET of zero. Both the DCB and ECB, if specified, are assumed to be addressable in the home address space.

If options requiring address constants (ADCONs) are not specified, the standard form of the SDUMP macro produces reentrant code for routines that reside in the link pack area. The following parameters do not require ADCONs and produce reentrant code for routines that reside in the link pack area:

HDRAD
IDAD
SDATA
TYPE
HDR
ID
BRANCH
SUSPEND
QUIESCE
BUFFER

SVC dump allows programs in page-protected storage (such as the nucleus, PLPA, and MLPA) to issue the standard form of the SDUMP macro without causing a protection exception.

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

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: from 1 to 100 characters. *dump title addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: A-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note**: XMEM and XMEME are mutually exclusive codes.
,PLISTVER=1 ,PLISTVER=2	*decimal digit 1*: Use up to a 68-byte parameter list. *decimal digit 2*: Use 136-byte parameter list.
,SYMREC=*symptom record addr*	*addr*: RX-type address, or register (2) - (12).
,ID=*'identifier'* ,IDAD=*identifier addr.*	*identifier*: from 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,ECB=*ecb addr*	*ecb addr*: A-type address, or register (2) - (12). **Note**: If ECB is specified, ASID or ASIDLST, BRANCH=YES, and LISTA must also be specified.
,SRB=*srb addr*	*srb addr*: A-type address, or register (2) - (12).

,SDATA=*(data code)*	*data code*: any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing the SDUMP macro results in the ALLPSA, SQA, IO, and SUMDUMP storage areas being dumped unless excluded by the NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP parameter. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS does not need to be specified. All SVC dumps will include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)*	*strt addr*: A-type address, or register (2) - (12). *end addr*: A-type address, or register (2) - (12).
,LIST=*list addr* ,LISTA=*list addr*	*list addr*: A-type address, or register (2) - (12). **Note**: One or more pairs of addresses may be specified, separated by commas.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: RX-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: RX-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO **Note**: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO
,SUMLIST=*list addr* ,SUMLSTA=*list addr*	*list addr*: RX-type address, or register (2) - (12).

The parameters are explained as follows:

HDR= *'dump title'*
HDRAD= *dump title addr*

specifies the title or address of the title to be used for the dump. If HDR is specified, the title must be 1-100 characters enclosed in apostrophes, although the apostrophes do not appear in the actual title. If HDRAD is specified, the first byte at the indicated address specifies the length of the title in bytes.

If these keywords are specified with BRANCH=YES or ASID/ASIDLST (that is, causing a scheduled dump), the title is moved to SVC dump storage before control returns to the caller. There is no requirement to synchronize with the completion of the dump.

,DCB= *dcb addr*

specifies the address of a previously opened data control block for the data set that is to contain the dump. If this parameter is omitted, one of the SYS1.DUMP data sets is used. The data control block must be addressable from all the address spaces in which the SVC dump routine executes. The control blocks built by OPEN must also be addressable from the address spaces. The DCB must support EXCP.

The DCB must reference device types supported by SVC dump. Eligible device types are unlabeled 9-track 2400-series tape devices (or tape devices compatible with the 2400-series) and any direct access devices supported by the system that have a track size of at least 4160 bytes. (4160 bytes equals 1 SVC dump output record.) The IBM 3850 Mass Storage System is not supported as a dump data set.

SVC dump does not close the dump data set. The caller should use the CLOSE macro to close the data set and cause an end-of-file mark or a tape mark to be placed after the dump data. SVC dump sets up the DCB so that CLOSE works correctly and positions the end-of-file mark or tape mark at the correct place on the data set. For tape data sets, the caller can write a tape mark to separate multiple dumps without using the CLOSE macro.

Note: The DCB resides in storage below 16 megabytes.

,ASID = *ASID addr*
,ASIDLST = *list addr*

specifies the address of a halfword or a list of halfwords containing the hexadecimal address space identifier of an address space to be dumped. If register notation is used, the low order halfword of the register contains the address space identifier of the address space to be dumped. If both parameters are omitted, the current address space will be dumped. If 0 is specified for the address space identifier, a dump is scheduled for the home address space of the issuer of the SDUMP macro. No private area storage will be included in the dump for the specified address space(s) if either of the following events occurred:

- No SDATA parameters were specified that apply to the private area of the requested address space(s).
- The CHNGDUMP operator command was used to set an overriding parameter in the system dump options list that limits SVC dumps to areas outside of the private area.

The ASID list can contain a maximum of 15 address space identifiers. The high order bit of the halfword containing the last identifier of the list must be set to 1, and all other high order bits must be set to 0.

,TYPE = XMEM
,TYPE = XMEME
,TYPE = NOLOCAL
,TYPE = FAILRC

specifies that the caller's cross memory mode is to be used to decide the address spaces to dump (XMEM or XMEME) or that the caller cannot allow SDUMP to obtain a local lock (NOLOCAL) or that SVC dump should return a reason code with the return code to the DUMP command processor when the requested dump was not taken (FAILRC).

XMEM requests SVC dump to use the caller's cross memory mode at the time the SDUMP macro is executed.

XMEME requests SVC dump to use the caller's cross memory mode at the time of the error for which the dump is being taken.

The home address space is dumped for both keywords. The relevant primary and secondary address spaces are also dumped if they are unique. If a cross memory local lock was held, the address space whose local lock is held is also dumped.

NOLOCAL indicates that the caller is in an environment that cannot tolerate SDUMP obtaining a local lock. This option has meaning only when BRANCH = YES is specified and the caller is enabled and unlocked (for example, in an enabled unlocked task FRR or in SRB or cross memory mode).

FAILRC requests that the caller receive special information from SVC dump whenever the dump fails. Some information is already placed in SDWASDRC as a result of the SVC dump failure. When the caller receives control again after a dump failure (return code 8) and the caller has specified TYPE = FAILRC, the reason code is combined with the return code and passed to the caller in either register 15 or the ECB. The reason code is in byte 3; the return code is in byte 4. When the return code is in the ECB, the

POST flag is set on. DUMP passes back a return code in register 15 and places the reason code in the SDWA. The reason code explains why the dump failed.

,PLISTVER = *1/2*
specifies the length of the parameter list used. When PLISTVER = 1 is specified, SDUMP uses a parameter list of up to 68 bytes. PLISTVER = 2 specifies a 136-byte parameter list. If SYMREC, ID, IDAD, PSWREGS, or SDATA = DEFAULTS, NODEFAULTS, or IO is specified, SDUMP defaults to use PLISTVER = 2.

,SYMREC = *symptom record addr*
specifies the address of a valid symptom record for DAE to use for dump suppression. DAE suppresses the SVC dump if the primary symptom string found in the symptom record matches previously known symptoms and suppression has been enabled by the installation.

The caller must build the symptom record and fill in at least the 'SR' identifier and the primary symptom string, which should uniquely identify the error.

SVC dump issues an abend with a completion code of '233'X then returns to the caller with a return code of 8 if the symptom record identifier is not 'SR' or if the first byte of the symptom record and the last byte of the secondary symptom string are not addressable.

SVC dump will not include the symptom record in the dump. The caller can use the SUMLIST keyword to include the symptom record in the dump.

See the dump analysis and elimination (DAE) section in *SPL: Application Development Guide* for more information on symptom strings and how to build them.

The ADSR macro maps the symptom record. See *Data Areas* for a macro mapping of the ADSR.

,ID = '*identifier*'
,IDAD = *identifier addr*
specifies an identifier that is included in dump message IEA911E, which is issued at the completion of the dump. The identifier must be from one to 50 printable characters. If ID is specified, the identifier must be enclosed in apostrophes, although the apostrophes do not appear in the actual identifier. If IDAD is specified, the first byte at the indicated address specifies the length of the identifier in bytes. If the length of the identifier is greater than 50, SVC dump issues an abend with a completion code of '233'X then returns to the caller with a return code of 8. If the length of the identifier is zero, SVC dump will continue processing as if the ID or IDAD parameter was not specified.

,PSWREGS = *list addr*
specifies a psw or register area to be passed to SVC dump. This area may contain a PSW, control registers 3 and 4, all the general purpose registers (GPRs), and all the access registers (ARs). When PSWREGS is specified, SVC dump will include the following information in the summary dump portion of the dump:

- The PSWREGS parameter list
- If the PSW is provided, 4K of storage before and 4K after the PSW address from the primary address space.
- 4K of storage before and 4K of storage after each of the GPRs from the primary and secondary address spaces.
- If the ARs are provided, they also are used to include 4K of storage before and 4K of storage after each of the GPRs. GPRs will be used to locate storage; ARs (if provided along with a PSW in AR mode) will be used to identify the source address space or data space.

Note: If the control registers are provided, they will be used to determine the primary and secondary address spaces. If no control registers are provided, then the storage will be dumped from the caller's primary and secondary address spaces.

The PSWREGS parameter allows programs running in a non-abend environment, where there is no SDWA, to request SVC dump and dump suppression services similar to those available in an abend environment, where an SDWA is present.

The parameter list for the PSWREGS parameter must reside in the address space currently addressable by SVC dump. The caller must provide at least the length and the mask field. Each bit in the mask refers to a data area. If a bit is set, SVC dump expects that the data area is supplied. If a mask bit is off and any lower-order mask bit is on, the corresponding storage area must be included in the parameter list. If a mask bit is off, but no lower-order mask bits are set, then the storage area need not be supplied.

The following diagram describes the parameter list:

Figure 24. PSWREGS Parameter List

Offset in Hex	Length	Field Description
00	2	The total length of the PSWREGS parameter list
02	2	Bit mask describing data areas included in the psw/register area
	1...	Bit 1: On - The PSW is included in the psw/register area
	.1..	Bit 2: On - Control registers 3 & 4 are included in the psw/register area
	..1.	Bit 3: On - General purpose registers are included in the psw/register area
	...1	Bit 4: On - Access registers are included in the psw/register area. Bits 5 - 16: Initialize these bits to zero.
04	8	PSW: Data only supplied if the PSW mask bit is set
0C	8	Control registers 3 and 4: Data only supplied if mask bit is set.
14	64	General purpose registers 0 - 15: Data only supplied if mask bit is set.
54	64	Access registers 0 - 15: Data only supplied if mask bit is set.

,ECB=*ecb addr*

If an A-type operand is specified, *ecb addr* specifies the address of a fullword containing the address of an event control block that is posted on completion of a scheduled dump. If a register operand is used, the register must contain the actual address of the event control block. If this parameter is omitted, the caller is not notified of the completion of the scheduled dump. The fullword and the event control block must be addressable from the home address space. The fullword address that points to the event control block must be a valid 24-bit or 31-bit address.

Note: The ECB will be posted only if a scheduled dump is taken and the return code from SDUMP is 0. Refer to *SPL: Application Development Guide* for a description of a scheduled dump.

,SRB=*srb addr*

specifies that the system is to dispatch the SRB that *srb addr* refers to when dumping completes. The SRB parameter is valid only when you specify ASID, ASIDLST, LISTA, TYPE=XMEM, TYPE=XMEME, SUBPLST, or BRANCH=YES. Thus, SRB= is valid only when you request a scheduled (asynchronous) dump. When you specify the ECB parameter, SRB is not valid.

When you obtain storage for the SRB, also obtain 24-bytes for SDUMP's return code and reason code. If, in addition to the return and reason codes, you want the name of the dump data set passed to the SRB routine, obtain an area of 84 bytes. Store the address of the 24-byte or 84-byte area into field SRBPARM. Use macro IHASDST to map the area.

Note: If SDUMP cannot initiate its asynchronous processing for a scheduled dump, it will not schedule the SRB specified by SRB=*srb addr*.

,SDATA=*(data code)*

specifies the system control program information to be dumped:

ALLNUC - The DAT-ON and DAT-OFF nuclei. The read only (page-protected) area of the nucleus and the DAT-OFF nucleus will not be included in the dump unless this keyword is specified.

ALLPSA - All of the prefixed storage areas in the system.

CSA - The common service area subpools (subpools 227, 228, 231 and 241).

GRSQ - Global resource serialization control blocks are included in the dump.

LPA - The active link pack area modules and SVCs for each address space being dumped.

LSQA - The local system queue area for each address space being dumped (subpools 203-205, 213-215, 223-225, 233-235, and 253-255).

NOALLPSA or NOALL
The prefixed storage area for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

NOSQA - The system queue area is not dumped.

NOSUMDUMP or NOSUM
A summary dump is not included in the SVC dump.

NUC - The non-page protected areas of the DAT-ON nucleus. (The ALLNUC parameter must be specified to obtain the entire nucleus, including the page-protected areas of the DAT-ON nucleus and the DAT-OFF nucleus.)

PSA - The prefixed storage area for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

RGN - The allocated pages in the private area of each address space being dumped. This includes the following areas:

Subpools	Storage
0-127, 251, 252	All virtual storage in the address space allocated to these subpools, that resides below and above the 16 megabytes line
203-205, 213-215, 223-225, 233-235, 253-255	All virtual storage allocated to the LSQA and ELSQA
236 and 237	All virtual storage allocated to the SWA and ESWA

SVC dump does not dump all the obtained storage in an address space if the RGN option of SDATA is specified. This reduces the number of page faults that occur during SVC dump processing, decreases the time required to take a dump, and reduces the size of dumps on tape or DASD.

For storage that is not related to data-in-virtual, only obtained pages that have something stored into them are dumped. This eliminates the pages of storage that are in a freshly obtained state.

For storage that is related to data-in-virtual, pages that are in central storage are dumped, as well as pages that have been changed since the last DIV macro (that specified the SAVE service) executed.

When the RGN option of SDATA is coded on the SDUMP macro, SVC DUMP determines which category a page belongs to. The categories are:

1. A copy of the page cannot be found in virtual storage.
2. All copies in virtual storage are at the same level as the copy on permanent storage.
3. At least one copy of the page has been found in virtual storage that is at a later level than the copy on permanent storage.
4. The system cannot determine the status of the page.

SVC dump does not dump pages in the first category. It dumps pages in category 2 that are in central storage and all of category 3. If the page is in category 4, SVC dump references the page again. If the

page is still in category 4, it is not dumped. If the page is no longer in category 4, it is treated like any other page in its category.

SQA - The entire system queue area.

SUMDUMP or SUM
A summary dump is to be included with the SVC dump output. The trace table is included in the non-summary portion of the dump if this option is specified or used as a default.

The type of summary dump depends on how you specify the BRANCH and SUSPEND parameters:

- If you specify BRANCH=YES and SUSPEND=NO, you get a disabled summary dump.
- If you specify BRANCH=YES and SUSPEND=YES, you get a suspend summary dump.
- If you specify BRANCH=NO, you get an enabled summary dump.

For a description of the dump contents, see *Diagnosis: Using Dumps and Traces*.

SWA - The scheduler work area subpools for each address space being dumped (subpools 236 and 237). This includes all virtual storage allocated above and below the 16 megabytes line.

TRT - The system trace table, the GTF trace records, and master trace data if these types of traces are active.

DEFAULTS or DEFS
The following default SDATA options are included in the SVC dump:

- ALLPSA
- SQA
- SUMDUMP
- IO
- Any default SDATA options specified by the CHNGDUMP command when CHNGDUMP is in ADD mode.

Notes:

1. DEFAULTS does not need to be specified. All SVC dumps include the default SDATA options unless NODEFAULTS is specified.
2. DEFAULTS and NODEFAULTS are mutually exclusive.

NODEFAULTS or NODEFS
The SDATA defaults are NOT included in the SVC dump. Specifying NODEFAULTS reduces the size of an SVC dump by excluding the following default SDATA options:

- ALLPSA
- SQA
- SUMDUMP
- IO
- Any default SDATA options specified by the CHNGDUMP command when CHNGDUMP is in ADD mode.

If a data area relating to an SDATA option is required in the dump, the programmer can code that SDATA option on the SDUMP macro invocation. All keywords and SDATA options are valid when NODEFS is coded.

The following considerations relate to the NODEFAULTS option:

- When NODEFAULTS is not coded, SVC dump includes all of the default areas.
- DEFAULTS and NODEFAULTS are mutually exclusive.
- When NODEFAULTS is specified, the dump will still contain some default system areas that are included in all dumps.

IPCS requires that these areas be included in the dump so that the dump reader can complete some basic diagnostic procedures.

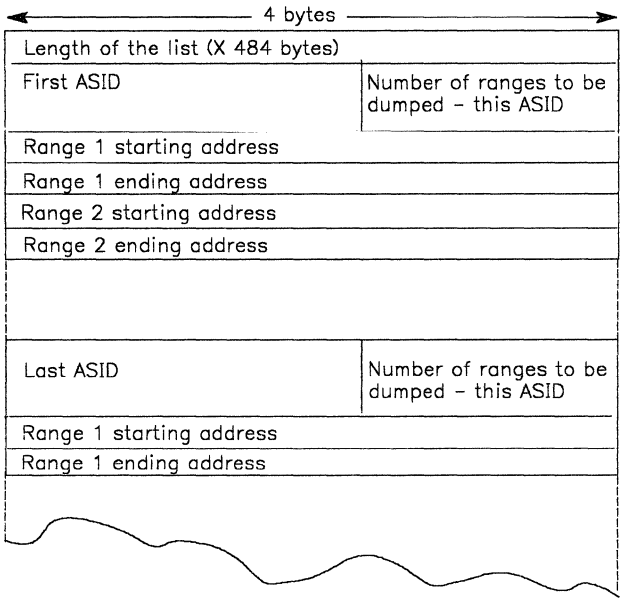
IO - The IO data areas are included in the SVC dump. This is an SDATA default option and does not need to be specified unless NODEFAULTS is specified. The NODEFAULTS option excludes the IO data areas from the SVC dump. If a programmer wants to use the NODEFAULTS option, but still wants the IO data areas dumped, the IO SDATA option should be coded.

,STORAGE = *(strt addr,end addr)*
,LIST = *list addr*
,LISTA = *listaddr*

specifies one or more pairs of starting and ending addresses (STORAGE), a list of starting and ending addresses (LIST), or a list of ASIDs and storage ranges (LISTA). Each starting address must be less than its corresponding ending address.

When LIST or STORAGE is specified, the list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high order bit of the fullword containing the last ending address of the list must be set to 1; all other high order bits must be set to 0.

When LISTA is specified, the first fullword of the storage list contains the number of bytes (including the first word) in the list. LISTA specifies a list of ASIDs and storage ranges as follows:



Note: If LISTA or SUBPLST is specified for a scheduled dump request and if the list does not exceed 484 bytes in size, SVC dump will move the list to SVC dump storage. The caller can free or reuse this space on return from SVC dump. No synchronization with SVC dump completion is required. If the list exceeds 484 bytes, SVC dump will not move the list and synchronization with SVC dump completion is required.

,SUBPLST=*subpool id list address*

specifies a list of ASIDs with associated subpool ids corresponding to subpools of virtual storage that are to be included in the SVC dump.

The first fullword of the list contains the number of bytes (including the first word) in the list. The list can contain a maximum of 200 bytes consisting of unique ASIDs and subpool ids. If the list contains duplicate ASIDs or subpool ids, the length can exceed 200 bytes because SDUMP stores the unique subpool ids in a 200-byte work area.

The structure of the list for each ASID follows:

- The first word contains the ASID in bits 0-15 and the number of subpools associated with this ASID (n) in bits 16-31. If 0 is specified as the ASID, the caller's home ASID is used.
- The next n halfwords contain the subpool ids (right justified) corresponding to the virtual storage to be included in the SVC dump. The manner in which these subpools are dumped depends on whether they are private or common area subpools.
  - If a private area subpool (related to a TCB) is specified, all virtual storage associated with this subpool, for all TCBs in the specified address space, is dumped.
  - If a common area subpool is specified, all of the virtual storage allocated in the subpool is dumped.

SVC dump does not dump all the obtained storage in an address space if the SUBPLST list keyword for private subpools is coded. This reduces the number of page faults that occur during SVC dump processing and the time required to take a dump. It also reduces the size of dumps on tape or DASD.

For storage that is not related to data-in-virtual, only obtained pages that have something stored into them are dumped. This eliminates the pages of storage that are in a freshly obtained state.

For storage that is related to data-in-virtual, only pages that are in central storage are dumped, as well as pages that have been changed since the last data-in-virtual SAVE operation.

When SUBPLST for private subpools is coded, SVC dump determines which category a page belongs to. The categories are:

1. A copy of the page cannot be found in virtual storage.
2. All copies in virtual storage are at the same level as the copy on permanent storage.
3. At least one copy of the page has been found in virtual storage that is at a later level than the copy on permanent storage.
4. The system cannot determine the status of the page.

SVC dump does not dump pages in the first category. It dumps pages in category 2 that are in central storage and all of category 3. If the page falls into category 4, SVC dump references the page again. If the page is still in category 4, it is not dumped. If the page is no longer in category 4, it is treated like other page in its category.

Notes:

1. SVC dump ignores unassigned subpool ids and ASIDs.
2. If an invalid subpool or ASID (ASID greater than ASVTMAX) is specified, the caller receives a 233 ABEND and SDUMP processing terminates the dump.
3. If all ASIDs specified in SUBPLST are the current ASID, SUBPLST does not force a scheduled dump. However, if any of the ASIDs are different, a scheduled (or asynchronous) dump results.
4. SDUMP callers executing in key 0 and supervisor state, who request storage from subpool 0 via GETMAIN obtain that storage from subpool 252 instead. Therefore, when these callers want to dump this storage, they must specify subpool 252 rather than subpool 0.

,KEYLIST = *storage key list addr*

specifies the address of a list of storage keys associated with the virtual storage to be dumped. If the key of a subpool specified in SUBPLST does not match a key in this list, the data in the subpool is not dumped. SUBPLST must be specified if the KEYLIST option is used. If KEYLIST is not specified, all virtual storage (regardless of key) associated with the requested subpools will be included in the dump.

The list contains one-byte entries and starts on a halfword boundary. The first byte indicates the length of the list (including this byte). The list has a maximum length of 16 bytes so that up to 15 keys can be specified. Callers should specify each key in the leftmost four bits of each byte, except the length byte.

Callers who want to dump the storage corresponding to all 16 keys should not specify this parameter.

If the KEYLIST option is specified, only the storage with keys matching the keys in the list is dumped.

,BUFFER = NO
,BUFFER = YES

specifies that the contents of the SQA buffer is (YES) or is not (NO) to be included in the dump. (The SQA buffer does not include the SDUMP parameter list or any data pointed to by the parameter list.) Callers who specify BUFFER = YES on the SDUMP or SDUMPX macro will obtain a dump of a 4K buffer reserved in the SQA for the callers of SVC dump. You can reserve the buffer by setting the high-order bit of the CVTSDBF field in the Communications Vector Table (CVT). Once you have reserved the buffer, you can fill it with data before issuing SDUMP or SDUMPX. Programs that are involved with volatile data, data that might change before SDUMP or SDUMPX can dump it, should use this buffer.

The CVTSDBF field of the CVT points to the buffer. Before using the buffer, use compare and swap logic to check the high-order bit of CVTSDBF. If the bit is on (B'1'), the buffer is in use, and you should continue processing as though a dump could not be taken. If the bit is off (B'0'), set the bit to B'1'. You can then fill the buffer and issue SDUMP.

,QUIESCE = YES
,QUIESCE = NO

specifies that the system is to be set non-dispatchable until the contents of the SQA and the CSA are dumped (YES), or that the system is to be left dispatchable (NO). If the SDATA parameter does not specify SQA or CSA, the QUIESCE = YES request is ignored.

Note: Summary dumps (SUMDUMP) for branch entries (BRANCH = YES) always cause the system to be set non-dispatchable until the summary dump is written.

,BRANCH = NO
,BRANCH = YES

specifies that a branch entry is to be used for interfacing with SVC dump to schedule a dump (YES), or that an SVC instruction is to be generated for interfacing with SVC DUMP (NO). For BRANCH = NO, the caller cannot be in cross memory mode, and must be in task mode with no locks held. For BRANCH = YES, the caller can be in either cross memory mode or non-cross memory mode and must be in PSW key 0, supervisor state, and one of the following:

SRB mode
Holding any lock

If BRANCH = YES is specified and the caller has not specified at least one of the following keywords: ASID, ASIDLST, TYPE = XMEM, TYPE = XMEME, or LISTA, the dump is scheduled to the current home address space.

Routines that issue SDUMP with BRANCH = YES must provide a 72-byte save area pointed to by register 13.

For BRANCH=YES entry by reentrant recovery routines, SDUMP processing moves the data supplied by the following parameters to a system area:

HDR
HDRAD
ID
IDAD
ASIDLIST
STORAGE
LIST
LISTA
SUBPLST
KEYLIST

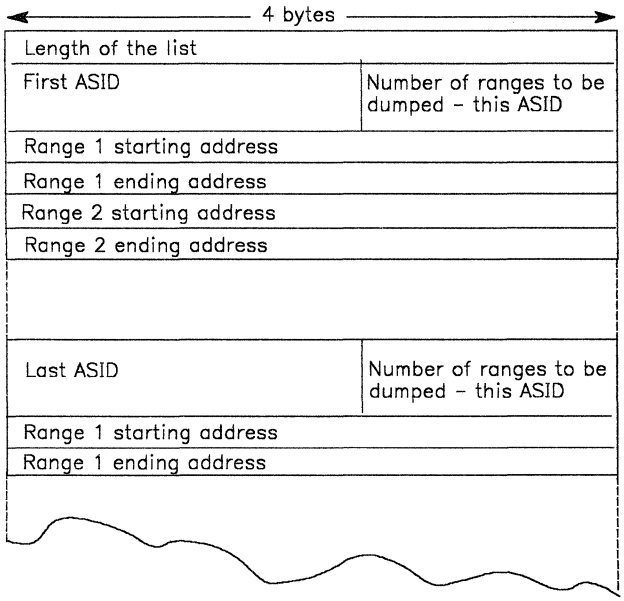
This enables the recovery routine to free its storage on return from SDUMP although the dump has not completed.

,SUSPEND=NO
,SUSPEND=YES
specifies that a suspend summary dump is requested (YES) or not requested (NO). SUSPEND=YES must be used together with the BRANCH=YES and SDATA=SUMDUMP parameters. This keyword should be used by routines that can experience page faults but that want to save volatile system dump information in a virtual storage buffer.

,SUMLIST=*list addr*
,SUMLSTA=*list addr*
specifies a list of starting and ending addresses of areas to be included in a summary dump (SUMLIST) or specifies a combined list of ASIDs and storage ranges (SUMLSTA). SUMDUMP must be specified as an SDATA parameter and each starting address must be less than its corresponding ending address.

For SUMLIST, the storage list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high order bit of the fullword containing the last ending address of the list must be set to 1, and all other high order bits must be set to 0.

For SUMLSTA, the first fullword of the list contains the number of bytes (including the first word) in the list. SUMLSTA specifies a list of ASIDs and storage ranges as follows:



Restriction:

- The maximum number of ASIDs that the combined TYPE=XMEM, TYPE=XMEME, LISTA, ASIDLST, ASID, and SUBPLST parameters can specify is fifteen.

  Note: There is no restriction on the number of ASIDs that the SUMLSTA can specify.

When BRANCH=YES and SUSPEND=NO are also specified, the list must be addressable using the addressability established when SVC dump was entered. The lists themselves and all ranges specified must reference paged-in data. Paged-out data is not dumped by summary dump.

When BRANCH=YES and SUSPEND=YES are also specified, the lists must be addressable using the addressability established when SVC dump was entered. The lists and referenced data can either be in paged in or paged out areas. The maximum amount of summary dump data with this type of dump is 1M.

When BRANCH=NO is also specified, the lists must be addressable in all address spaces in which the dump will be taken (those address spaces specified by ASID, ASIDLST, LISTA, or TYPE=XMEM, TYPE=XMEME, or SUBPLST). The lists and referenced data can be in paged-in or paged-out areas. The maximum amount of summary dump data possible with this type of dump is dependent only on the size of the dump data set.

Each ASID specified with SUMLSTA must represent a valid, swapped-in address space in order for the data to be dumped.

Programming Notes:
The total number of distinct ASIDs that can be specified by TYPE=XMEM, TYPE=XMEME, LISTA, ASID, SUBPLST and ASIDLST is fifteen. If more than fifteen are requested, only the first fifteen are processed. There is no restriction on the number of ASIDs specified by the SUMLSTA parameter, nor do SUMLSTA ASIDs contribute toward the fifteen ASID limit.

If BRANCH=NO was specified and no ASIDs other than the current ASID were requested, register 15 contains one of the following return codes when control is returned:

Hexadecimal Code	Meaning
00	A complete dump was taken.
04	A partial dump was taken because the dump data set did not have sufficient space.
08	The system was unable to take a dump.

If BRANCH=YES or any ASID other than the current ASID was requested, register 15 contains one of the following return codes when control is returned:

Hexadecimal Code	Meaning
00	A dump was scheduled. If an ECB was supplied, it will be posted on completion of the dump.
08	The system was unable to schedule a dump.

If an ECB was supplied, one of the following codes is returned in the ECB:

Hexadecimal Code	Meaning
00	A complete dump was taken.
04	A partial dump was taken.
08	The system was unable to take a dump.

Notes:

1. The ECB will not be posted unless the return code from SDUMP is 0.
2. When a return code of 8 is received, a reason code is returned. The reason code is in the following locations:
   - In the SDWASDRC field of the SDWA.
   - In either the ECB or register 15, provided that the FAILRC parameter is specified.

- In the SDSTATUS field. This field is pointed to by the SRBPARM field that is in the SRB parameter list. The parameter list is passed to SDUMP by using the SRB keyword.

The reason codes are as follows:

Figure 25 (Page 1 of 2). SDUMP Reason Codes

Hexadecimal Reason Code	Meaning
0	No SVC dump was requested.
1	An SVC dump was successfully started.
2	An SVC dump was suppressed because another SVC dump was in progress.
3	An SVC dump was suppressed by a request by the installation (for example: DUMP=NO at IPL or CHNGDUMP SET,NODUMP).
4	An SVC dump was suppressed by a SLIP NODUMP command.
5	An SVC dump was suppressed because a SYS1.DUMP data set was not available. (Only for synchronous dumps.)
6	An SVC dump was suppressed because an I/O error occurred during the initialization of the SYS1.DUMP data set. (Only for synchronous dumps.)
8	An SVC dump was suppressed because an SRB could not be scheduled to activate the dump tasks in the requested address spaces.
9	An SVC dump was suppressed because a terminating error occurred in SDUMP(X) before the first dump record was written.
A	An SVC dump was suppressed because a status stop SRB condition was detected. (This prevents dump I/O from completing.)
B	An SVC dump was suppressed by DAE.
15	The parameter list address is zero.
16	The parameter list is not a valid SVC dump or SNAP parameter list.
17	The caller-supplied data set is not supported.
18	The start address is greater than the end address in a storage list.
19	The caller-supplied header is longer than 100 characters.
1A	The 4K buffer was requested, but not locked.
1B	A storage list overlaps the 4K buffer.
1C	The caller-supplied DCB is invalid.
1E	An ASID in the ASID list is syntactically invalid.
22	The 4K buffer was requested with an SDUMP already in progress.
25	An invalid subpool ID was specified in the subpool list.
28	Part of the parameter list is inaccessible.
29	The caller-supplied DCB is inaccessible.
2A	The caller-supplied storage list is inaccessible.
2B	The caller-supplied header data is inaccessible.
2C	The caller-supplied ECB is inaccessible.
2D	The caller's ASID list is inaccessible.
2E	The caller's SUMLIST/A is inaccessible.
2F	The caller's SUBPOOL list is inaccessible.
30	The caller's storage key list is inaccessible.
31	Copies of the SLIP register and PSW are inaccessible.
32	The caller-supplied SRB is inaccessible.
33	The version number in the parameter list is not valid.
34	The caller's LISTD is inaccessible.

Figure 25 (Page 2 of 2). SDUMP Reason Codes

Hexadecimal Reason Code	Meaning
35	The caller's SUMLSTL is inaccessible.
36	The parameter list contains conflicting parameters.
37	The ID is longer than 50 characters.
38	The ID is not addressable.
39	The psw/register area is an incorrect length.
3A	The PSWREGS area is not addressable.
3B	The symptom record is invalid.
3C	The symptom record is not addressable.
3D	The DEB for the caller-supplied DCB is inaccessible.
FF	An SVC dump was suppressed for some other unspecified reason.

Example 1

Operation: This example shows how SVC dump can be branch entered to initiate a dump in an address space by callers who cannot issue an SVC. Areas to be dumped are requested via three parameters (BUFFER, SDATA, and STORAGE). The dump has the title indicated in the HDR parameter, the caller requests to be notified of the completion of the scheduled dump via the ECB parameter, and the dump is going to a private data set (indicated by the DCB option).

```
SDUMP HDR='USER DATA FOR TEST A',DCB=TESTADCB,BUFFER=YES,  X
  ASID=TSTAASID,ECB=(8),QUIESCE=YES,BRANCH=YES,           X
  STORAGE=(A,B,C,D,(9),E),SDATA=(ALLPSA,SQA,LSQA)
```

Example 2

Operation: This example shows how SVC dump can be invoked via a branch entry to initiate a dump of several address spaces by callers who cannot issue an SVC. Areas to be dumped are requested via four parameters (BUFFER, SDATA, LIST, and SUMLIST). The address spaces to be dumped are described by the ASIDLST parameter. Note that areas specified by SUMLIST only apply to the current address space. The LIST addressed by the LIST keyword must be addressable from any address space. The dump has the title indicated in the HDR parameter, and the caller requests to be notified of the completion of the scheduled dump via the ECB parameter.

```
SDUMP HDR='USER DATA FOR TEST B',                          X
  BUFFER=YES,ASIDLST=TSTALIST,ECB=(8),                     X
  QUIESCE=YES,BRANCH=YES,LIST=(9),                         X
  SDATA=(ALLPSA,NUC,SQA,SUMDUMP),                          X
  SUMLIST=TSTSLIST
  .
  .
  .
TSTALIST DC X'0000000A800B'
TSTSLIST DC X'0000000080400000'
```

SDUMPX — Dump Virtual Storage

The SDUMPX macro provides a dumping capability for system routines that are running in access register mode. This macro is similar to the SDUMP macro, except that it generates code and addresses that are appropriate for access register mode. All parameters on the SDUMP macro are valid for the SDUMPX macro. The LISTD and SUMLSTL parameters, however, are valid only on the SDUMPX macro. These two parameters are described in this section.

If you are in access register mode, before you issue the SDUMPX macro, issue the SYSSTATE ASCENV=AR macro to tell the SDUMPX macro to generate code appropriate for access register mode.

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

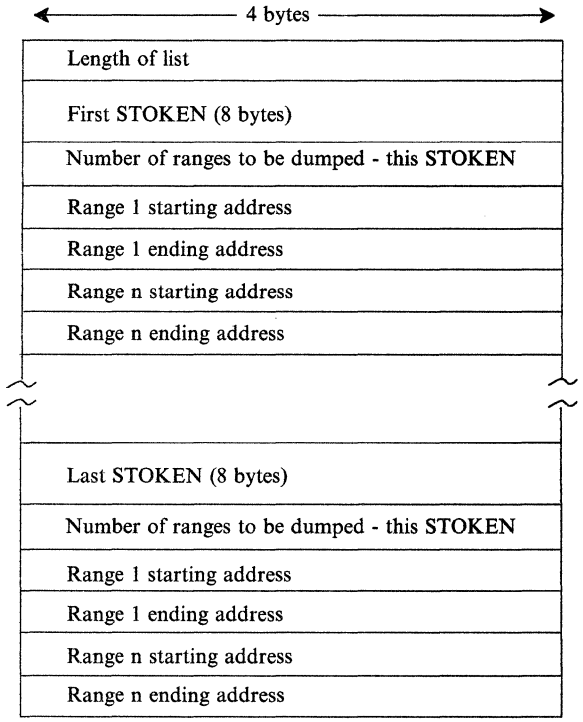
name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede SDUMP or SDUMPX.
SDUMPX	
b	One or more blanks must follow SDUMP or SDUMPX.

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: from 1 to 100 characters. *dump title addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: A-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note**: XMEM and XMEME are mutually exclusive codes.
,PLISTVER=1 ,PLISTVER=2	*decimal digit 1*: Use up to 104-byte parameter list. *decimal digit 2*: Use 136-byte parameter list.
,SYMREC=*symptom record addr*	*addr*: RX-type address, or register (2) - (12).
,ID=*'identifier'* ,IDAD=*identifier addr*	*identifier*: from 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,ECB=*ecb addr*	*ecb addr*: A-type address, or register (2) - (12). **Note**: If ECB is specified, ASID or ASIDLST must also be specified.
,SRB=*srb addr*	*srb addr*: A-type address, or register (2) - (12).

,SDATA=*(data code)*	*data code*: any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing the SDUMP macro results in the ALLPSA, SQA, IO, and SUMDUMP storage areas being dumped unless excluded by the NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS does not need to be specified. All SVC dumps will include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)*	*strt addr*: A-type address, or register (2) - (12). *end addr*: A-type address, or register (2) - (12).
,LIST=*list addr* ,LISTA=*list addr* ,LISTD=*list addr*	*list addr*: A-type address, or register (2) - (12). **Note**: One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)*
,SUBPLST=*subpool id list addr*	*subpool id list addr*: RX-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: RX-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO **Note**: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO
,SUMLIST=*list addr* ,SUMLSTA=*list addr* ,SUMLSTL=*list addr*	*list addr*: RX-type address or register (2) - (12).

The parameters for the SDUMPX macro are explained under the standard form of the SDUMP macro, with the following exceptions:

,LISTD= *list addr*
specifies a list of address ranges, qualified by STOKENs, of areas to be included in the SVC dump. Specify the STOKENs and address ranges as follows:



The first fullword of the list contains the number of bytes (including the first word) in the list.

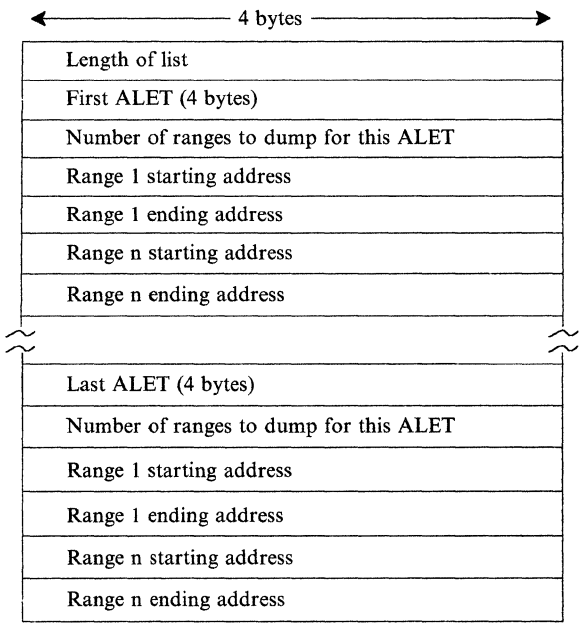
STOKEN refers to any address/data space. SVC dump does not dump data space storage that has not been referenced.

LISTD causes a scheduled dump when the caller performs one of the following actions:

- Requests a SCOPE=SINGLE data space that is owned by a task in an address space other than the caller's primary address space.
- Requests an address space other than the primary
- Uses the ECB or SRB parameter

,SUMLSTL = *list addr*

specifies a list of address ranges, qualified by ALETs, of areas to be included in a summary dump. Specify the ALETs and address ranges as follows:



The first fullword of the list contains the number of bytes (including the first word) in the list.

ALET refers to entries in either a DU-AL or a PASN-AL, and associated with any address/data space that the caller has addressability to. SVC dump does not dump data space storage that has not been referenced.

SDUMP and SDUMPX (List Form)

Use the list form of the SDUMP or SDUMPX macro to construct a control program parameter list. You can specify any number of storage addresses using the STORAGE parameter. Therefore, the number of starting and ending address pairs in the list form of SDUMP or SDUMPX must be equal to the maximum number of addresses specified in the execute form of the macro.

The list form of the SDUMP or SDUMPX macro is written as follows:

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: from 1 to 100 characters. *dump title addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.
,PLISTVER=1 ,PLISTVER=2	*decimal digit 1*: Use up to 68-byte parameter list (up to 104-byte for SDUMPX). *decimal digit 2*: Use 136-byte parameter list.
,SYMREC=*symptom record addr*	*addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*	*identifier*: from 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,SDATA=*(data code)*	*data code*: any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing the SDUMP macro results in the ALLPSA, SQA, IO, and SUMDUMP storage areas being dumped unless excluded by the NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP parameter. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS does not need to be specified. All SVC dumps will include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)*	*strt addr*: A-type address. *end addr*: A-type address.
,LIST=*list addr* ,LISTA=*list addr* ,LISTD=*list addr*	*list addr*: A-type address. **Notes:** 1. One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)* 2. LISTD is valid on SDUMPX macro only.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: A-type address, or register (2) - (12).

,KEYLIST=*storage key list addr*	*storage key list addr*: A-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO
,TYPE=*(type code)*	*type code*: Any combination of the following, separated by commas: XMEM or XMEME, NOLOCAL.

,MF=L

The parameters are explained under the standard form of the SDUMP or SDUMPX macro, with the following exception:

,MF=L
specifies the list form of the SDUMP or SDUMPX macro.

Note: If SYMREC, ID, IDAD, PSWREGS, SDATA=NODEFS, SDATA=DEFS or SDATA=IO is not used on the list form of the macro, but may be coded on the execute form, use PLISTVER=2 when specifying MF=L to generate a 136-byte parameter list.

SDUMP and SDUMPX (Execute Form)

A remote control program parameter list is referred to and can be modified by the execute form of the SDUMP or SDUMPX macro.

If you code one or more of the SDATA parameters on the execute form of the macro, any SDATA parameters coded on the list form are lost.

The execute form of the SDUMP or SDUMPX macro is written as follows:

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: from 1 to 100 characters. *dump title addr*: RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: RX-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: RX-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: any of the following, separated by commas: XMEM or XMEME, NOLOCAL
,PLISTVER=1 ,PLISTVER=2	*decimal digit 1*: Use up to 68-byte parameter list (up to 104-byte for SDUMPX). *decimal digit 2*: Use 136-byte parameter list.
,SYMREC=*symptom record addr*	*addr*: RX-type address, or register (2) - (12).
,ID=*'identifier'* ,IDAD=*identifier addr*	*identifier*: from 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,ECB=*ecb addr*	*ecb addr*: RX-type address, or register (2) - (12).
,SDATA=*(data code)*	*data code*: any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing the SDUMP macro results in the ALLPSA, SQA, IO, and SUMDUMP storage areas being dumped unless excluded by the NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP parameter. 2. The PSA and IO options are not required unless NODEFS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS does not need to be specified. All SVC dumps will include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)*	*strt addr*: RX-type address, or register (2) - (12). *end addr*: RX-type address, registers (2) - (12).

,LIST=*list addr* ,LISTA=*list addr* ,LISTD=*list addr*	*list addr*: RX-type address, or register (2) - (12). **Notes:** 1. One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)* 2. LISTD is valid on SDUMPX macro only.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: RX-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: RX-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	
,QUIESCE=YES ,QUIESCE=NO	
,BRANCH=NO ,BRANCH=YES	**Note**: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO
,SUMLIST=*list addr* ,SUMLSTA=*list addr* ,SUMLSTL=*list addr*	*list addr*: RX-type address or register (2) - (12). **Note:** SUMLSTL is valid on SDUMPX only.
,MF=(*E,ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the SDUMP or SDUMPX macro, with the following exception:

,MF=(E, *ctrl addr)*
specifies the execute form of the SDUMP or SDUMPX macro using a remote control program parameter list.

Example 1

Operation: The execute form is used to change SDATA areas, BUFFER, and QUIESCE options in the SDUMP or SDUMPX parameter list. The list form of SDUMP was previously used to create the basic SDUMP parameter list located by register 1.

```
SDUMP SDATA=(SQA,LPA),BUFFER=NO,QUIESCE=NO,MF=(E,(1))
```

Example 2

Operation: This example shows a dump request from SUBSYSTEM1. This dump will be suppressed if the symptoms in the symptom record match a previous dump's symptoms, and if the installation has enabled dump suppression. The dump will not include the SDATA options specified on CHNGDUMP or the ALLPSA or SQA data areas. The dump will include the IO data areas and a summary dump which will contain the psw/register data.

```
SDUMP ID='SUBSYSTEM1',SYMREC=(8),SDATA=(NODEFS,10),PSWREGS=(9)
```

SETFRR — Set Up Functional Recovery Routines

The SETFRR macro gives authorized programs the ability to define their recovery in the FRR (functional recovery routine) LIFO stack, which is used during processing of the system recovery manager. Any program function can use SETFRR to define its own unique recovery environment.

The SETFRR macro can be used to add, delete, or replace FRRs in the LIFO stack, or to purge all FRRs in the stack. The macro also optionally returns to the user the address of a parameter area that is eventually passed to the FRR when an error occurs. The parameter area can be used to keep information that might be useful to the FRR. The exit and retry routines execute in the same addressing mode as the SETFRR macro expansion and service routine. This is the addressing mode of the issuer of the macro.

The issuer of the SETFRR macro can be in any cross memory mode but must be in supervisor state key zero. When the EUT=YES parameter is not specified, the caller must also be either locked, disabled, or in SRB mode.

Callers in AR mode can issue the SETFRR macro. Before issuing the SETFRR macro in AR mode, an AR mode caller must issue SYSSTATE ASCENV=AR to tell the SETFRR macro to generate code appropriate for AR mode. For AR mode callers, the SETFRR parameter list can be located in any address space.

All SETFRR users must include the DSECTs for the FRR stack (via the IHAFRRS mapping macro) and the PSA (via the IHAPSA mapping macro) before using the SETFRR macro.

Support of the SETFRR macro sets the high-order bit of the recovery exit routine address to the addressing mode of the issuer. This bit setting determines the addressing mode of both the recovery exit routine and the retry routine.

SPL: Application Development Guide describes the interface to an FRR under "System Environment"; guidelines for writing an FRR appear in *SPL: Application Development Guide* under "Recovery Routine Guidelines".

Note: FRRs need not restore registers upon return.

The SETFRR macro is written as follows:

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

b                                One or more blanks must precede SETFRR.

SETFRR

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

```
A,FRRAD=FRR addr                 FRR addr: A-type address, or register (2) - (12).
R,FRRAD=FRR addr
D
P

,WRKREGS=(reg1,reg2)             reg1: decimal digits 1-15.
                                 reg2: decimal digits 1-15.

   ,PARMAD=parm area addr        parm area addr: A-type address, or register (2) - (12).
                                 Note: This parameter may only be specified with A or R above.

   ,CANCEL=YES                   Default: CANCEL=YES
   ,CANCEL=NO

   ,EUT=YES

   ,MODE=
     (                           Default: MODE=HOME
     FULLXM
     PRIMARY
     HOME
     ,
     GLOBAL
     LOCAL
     GLOBAL,LOCAL
     LOCAL,GLOBAL
     )

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

The explanation of the parameter is as follows:

A,FRRAD= *FRRAD addr*
R,FRRAD= *FRRAD addr*
D
P

specifies the operation to be performed on the FRR LIFO stack:

A an FRR address is to be added to the stack.
R the FRR address last added to the stack is to be replaced by another FRR address.
D the FRR address last added to the stack is to be deleted.
P all entries in the stack are to be purged.

FRRAD specifies the address of a fullword containing the FRR address that is to be added or replaced. The parameter specifies the FRR address in a register or specifies the address of a storage location containing the FRR address.

,WRKREGS= *(reg1,reg2)*

specifies two unique general purpose registers to be used as work registers in the code generated by the SETFRR macro expansion.

,PARMAD= *parm area addr*

specifies the address of a fullword to receive the address of the 24-byte parameter area provided by the system to the issuer of SETFRR. If a register is specified, the address of the 24-byte parameter area is placed in the register. This parameter area is

associated with the FRR address that has either been added to or has replaced an FRR address on the stack. This parameter area is passed to the FRR when an error occurs.

,CANCEL = YES
,CANCEL = NO

specifies whether you want to allow the recovery routine to be interrupted by cancel or detach processing.

To allow a recovery routine to be interrupted, specify CANCEL = YES.

To prevent a recovery routine from being interrupted, specify CANCEL = NO. If a cancel or detach is attempted against a recovery routine for which you have specified CANCEL = NO, MVS defers cancel and detach processing until the recovery routine returns control to the system.

Usage Notes:

1. If a recovery routine that runs under the CANCEL = NO option can be called by an unauthorized program running under the same task, IBM recommends that you specify ASYNCH = NO for each ESTAE(X) macro that the recovery routine issues. This also includes any ESTAE(X) macros issued by programs that the recovery routine calls.
2. If a recovery routine running under the CANCEL = NO option calls an unauthorized program, cancel and detach processing is also deferred for the called program.

,EUT = YES

used only with A and R, specifies that the new FRR can be used in any environment. EUT = YES is used by routines that are not certain of their environment; for example, a routine that can be called by an SRB or by a task that is executing enabled and might not hold any locks. While the FRR remains in effect, no SVCs can be issued, no new asynchronous exits are dispatched, and no vector instructions can be executed.

,MODE = *options*

specifies the environment in which the FRR is to get control and also, optionally, identifies the FRRs that free critical resources. The normal or expected addressing environment is identified by FULLXM, PRIMARY, or HOME. The restricted or critical resource freeing addressing environment is identified by LOCAL, GLOBAL, or both. Parentheses are not needed if only one option is chosen.

FULLXM

specifies that the FRR exit must be entered in the same cross memory environment that existed when the SETFRR was issued.

PRIMARY

specifies that the FRR exit must be entered in primary addressing mode with both the PASID and SASID the same as the PASID that existed when the SETFRR was issued, the home address space must be unchanged, and the PSW key mask must be the same as when the SETFRR was issued.

HOME

specifies that the FRR exit must be entered in primary addressing mode with PASID = SASID = HASID, and the PSW key mask either the same as that at the time of the error for SRB mode, or the task storage protect key for TCB mode.

If neither FULLXM, PRIMARY, nor HOME is coded, HOME is the default.

GLOBAL

specifies that the FRR frees a critical global resource. If the FRR cannot be entered in its normal addressing environment (for example, if the secondary address space is no longer valid), it must be entered in GLOBAL restricted addressing environment to free critical resources. To enter the FRR, a global spin lock must be held.

If it cannot be entered either as an FRR or as a resource manager, the FRR is skipped.

LOCAL
specifies that the FRR frees a critical local resource. If the FRR cannot be entered in its normal addressing environment then it must be entered in LOCAL restricted addressing environment to free resources.

In order for the FRR to be entered in LOCAL restricted addressing environment, a local lock must be held.

If it cannot be entered either as an FRR or as a resource manager, the FRR is skipped.

,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.

Example 1

Operation: Add an FRR to the FRR stack and return the address of the parameter list to the issuer of the SETFRR. The FRR address contained in register (R5) is placed on the FRR stack in the next available FRR entry. On return, register (R2) contains the address of the parameter list associated with this FRR entry. Registers R3 and R4 are work registers used in the code generated by SETFRR in performing its operations.

```
SETFRR A,FRRAD=(R5),PARMAD=(R2),WRKREGS=(R3,R4)
```

Example 2

Operation: Delete the last FRR added to the FRR stack.

```
SETFRR D,WRKREGS=(1,6)
```

SETLOCK — Control Access to Serially Reusable Resources

The SETLOCK macro is used to control access to serially reusable resources. Each kind of serially reusable resource is assigned a separate lock. To use SETLOCK, you must be executing in supervisor state with PSW key zero. The SETLOCK macro can be used by programs executing in cross memory mode or in access register mode. Before you invoke the SETLOCK macro in access register mode, issue SYSSTATE ASCENV=AR, to tell the SETLOCK macro to generate code appropriate for access register mode. A DSECT for the PSA (via the IHAPSA mapping macro) must be included in the CSECT using the SETLOCK macro.

SETLOCK can be used to:

- Obtain a specified lock
- Release a specified lock
- Test a specified lock or to determine if the lock is held on the requestor's processor

For information on using this macro on an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.

Locks are discussed under "Locking" in *SPL: Application Development Guide*. CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

Notes:

1. In MVS/SP Version 3 and later versions, a locked routine is not allowed to issue an SVC, or invoke a routine that would issue an SVC on the locked routine's behalf.
2. The contents of access registers 0, 1, 14, and 15 are volatile across the macro invocation.

OBTAIN Option

The OBTAIN option of SETLOCK macro is written as follows:

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

OBTAIN	
,TYPE=CPU ,TYPE=CMS ,TYPE=CML,ASCB=(11) ,TYPE=LOCAL	
,MODE=COND ,MODE=UNCOND	**Note:** MODE cannot be specified with TYPE=CPU.
,REGS=SAVE ,REGS=USE	**Note:** Registers 11-14 will be destroyed if this parameter is omitted.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

OBTAIN
specifies that the designated lock is to be obtained on the caller's behalf.

,TYPE=CPU
,TYPE=CMS
,TYPE=CML, ASCB=(11)
,TYPE=LOCAL
specifies the type of lock that is to be obtained on the caller's behalf.

The types available are:

CPU is the processor lock. It is a pseudo spin lock providing MVS-recognized disablement. There is one CPU lock per processor and no processor can request another processor's lock. The lock is always available. Users not holding a spin lock can obtain the CPU lock to become disabled for I/O and external interruptions.

CMS is the cross memory services lock. It is a global suspend lock used to serialize functions between address spaces where this serialization is not provided by one or more of the global spin locks. The caller must hold the CML or LOCAL lock to obtain the CMS lock.

CML is the cross memory local lock. It is a local level suspend type lock used to serialize resources in an address space other than the home address space.

The requestor of a CML lock must have authority to access the specified address space before requesting the lock. To establish authority, the requestor sets the primary or secondary address space to the one specified by the ASCB=(11) parameter. This address space must be non-swappable before the SETLOCK request.

LOCAL is the lock that serializes resources in the home address space pointed to by PSAAOLD. It is a local level suspend lock.

,ASCB=(11)

specifies that the address of the ASCB whose local lock is requested has been loaded into register 11 before the SETLOCK request. This parameter must be specified if TYPE=CML is specified and is valid only for CML lock requests

Note: If the requestor specifies OBTAIN, TYPE=CML and the ASCB=(11) parameter points to the home address space, the request is treated as though the LOCAL lock were being obtained.

The return registers are:

11 Unchanged if ASCB is specified, otherwise used as a work register by the macro.
12 Used as a work register by the macro.
13 Return code.
14 Return address.

,MODE=COND
,MODE=UNCOND

specifies whether the lock is to be conditionally or unconditionally obtained.

COND specifies that the lock is to be conditionally obtained. That is, if the lock is not owned on another processor, it is acquired on the caller's behalf. If the lock is already held, control is returned indicating that the caller holds the lock or that another unit of work on another processor owns the lock.

UNCOND specifies that the lock is to be unconditionally obtained. That is, if the lock is not owned on another processor, it is acquired on the caller's behalf. If the lock is already held by the caller, control is returned to the calling program indicating that it already owns the lock. If the lock is held on another processor, the caller's processor spins on the lock until it is released or suspends the SETLOCK caller until the lock is available.

,REGS = SAVE
,REGS = USE

specifies the use of general purpose registers 11 through 14. If you omit REGS, the SETLOCK service uses general purpose registers 0, 1, 14, and 15.

SAVE specifies that register contents are to be saved. Registers 11 through 14 are saved in the area pointed to by register 13, and are restored upon completion of the SETLOCK request. The save area consists of at least 5 words (These words hold the contents of the four registers and the return code that is to be placed in register 15).

Note: The save area used for the REGS = SAVE parameter must be a different area than the standard linkage save area used by the program.

USE specifies that registers 14, 15, 0, and 1 are to be used by the macro service. Registers 11, 12, and 13 are saved in registers 15, 0, and 1, respectively, and are restored upon completion of the SETLOCK request. Register 14 is used as a link register; register 15 contains the return code.

Note: If neither SAVE nor USE is specified, registers 11-14 are destroyed and register 13 contains the return code.

,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.

When control is returned, register 15 (register 13, if neither SAVE nor USE is specified) contains one of the following return codes:

Hexadecimal Code	Meaning
00	The lock was successfully obtained.
04	The lock was already held by the caller. The lockword id matches the caller's id.
08	The conditional obtain process was unsuccessful. The lockword id does not match the caller's id. This means that the lock is owned by another processor. In the case of a shared/exclusive lock, this return code means that the lock was not immediately available with the scope requested.
10	A level error was detected. This return code is supplied on a conditional obtain only. A level error detected on an unconditional obtain results in an abnormal termination.

Notes:

1. See the topic "Locking" in *SPL: Application Development Guide* for a description of the types of level errors that the lock manager can and cannot detect.
2. For an unconditional request, if the caller holds the lockword on a different level, the lock manager abnormally terminates the caller with a 073 ABEND.

RELEASE Option

The RELEASE option of the SETLOCK macro is written as follows:

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

ƀ                                One or more blanks must precede SETLOCK.

SETLOCK

ƀ                                One or more blanks must follow SETLOCK.

RELEASE

,TYPE=CPU
,TYPE=CMS
,TYPE=LOCAL
,TYPE=ALL
,TYPE=CML,ASCB=(11)

   ,REGS=SAVE
   ,REGS=USE

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

The parameters are explained under the OBTAIN option of the SETLOCK macro, with the following exceptions:

RELEASE
specifies that the lock is to be released.

,TYPE=ALL
specifies the type of lock to be released.

ALL indicates that all locks currently held on the processor are to be released.

Notes:

1. For information on using this macro on an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.
2. If you specify RELEASE, TYPE=CML and the ASCB=(11) parameter specifies the home address space and the lock you are holding is home's local lock, then SETLOCK processing treats the CML release request as a RELEASE, TYPE=LOCAL.

When control is returned, register 15 (register 13 if neither SAVE nor USE is specified) contains one of the following return codes:

Hexadecimal Code	Meaning
00	The lock was successfully released.
04	The lock was not owned. The lock was free when the release request was issued.
08	The release process was unsuccessful. The lockword id does not match the caller's id. This means that the lock was owned by a different processor.
0C	The release process was unsuccessful. The caller does not own the specified local or CML lock. This return code applies to LOCAL or CML release only.

Example

Operation: Release the local lock.

```
SETLOCK RELEASE,TYPE=LOCAL,RELATED=(TCBRQ,MOD1(NAME1),   X
  MOD2(NAME2))
```

TEST Option

The TEST option of the SETLOCK macro is written as follows:

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

TEST	
,TYPE=CPU ,TYPE=CMS ,TYPE=LOCAL ,TYPE=ALL ,TYPE=CML ,TYPE=ALOCAL	
,ASCB=*(11)*	**Note:** ASCB can only be specified with TYPE=CML.
,LOCKHLD=*(reg)*	*reg:* any valid register value **Note:** LOCKHLD can only be specified with TYPE=CML, TYPE=ALOCAL, TYPE=CPU
,BRANCH=(HELD,*addr*) ,BRANCH=(NOTHELD,*addr*)	*addr:* RX-address.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under the OBTAIN or RELEASE option of the SETLOCK macro, with the following exceptions:

TEST
specifies that the designated lock is to be checked to determine if it is currently held on the requesting processor.

,TYPE=CML
,TYPE=ALOCAL
The types are:

CML — specifies that the requestor wishes to determine whether a CML lock is held. The ASCB=(11) parameter or the LOCKHLD=*(reg)* parameter must be specified with TYPE=CML.

ALOCAL — specifies that the requestor wishes to determine whether a local lock is held, either home's LOCAL or a CML. The LOCKHELD=*(reg)* parameter may be specified with TYPE=ALOCAL. ASCB may not be specified with TYPE=ALOCAL.

,ASCB=(11)
specifies that the register 11 contains the ASCB address that is to be checked to determine if the requestor owns its local lock as a CML lock. This parameter is only valid with TYPE=CML.

,LOCKHLD=*(reg)*
specifies that the a designated register is to be used as a work register by the macro. This parameter is valid only for TYPE=CML and TYPE=CPU.

If TYPE=CML is specified, and if a CML lock is held, this register will contain the ASCB address of the CML locked address space.

If TYPE = CPU is specified, this register will be loaded with the current CPU lock use count for this processor.

,BRANCH = (HELD,*addr*)
,BRANCH = (NOTHELD,*addr*)
specifies that the return code setting of the macro is to be suppressed and replaced by a direct branch to the specified address or the specified label.

If (HELD,*addr*) is specified, the address is branched to if the specified lock is held on the requesting processor.

If (NOTHELD,*addr*) is specified, the address is branched to if the specified lock is not currently held on the requesting processor.

When control is returned, register 15 contains one of the following return codes (if the BRANCH = parameter was omitted):

Hexadecimal Code	Meaning
00	The lock was held by the requestor, or at least one lock was held (if TYPE = CMS or TYPE = ALL was specified).
04	The lock was not held by anybody, or no lock was held (if TYPE = CMS or TYPE = ALL was specified).

Note: TYPE = CMS is used to determine if at least one cross memory services lock is held, but cannot be used to determine which one or if all are held.

Example 1
Operation: If a local lock is not held, branch to DSRLLINT, otherwise, execute the next sequential instruction.

```
SETLOCK TEST,TYPE=LOCAL,BRANCH=(NOTHELD,DSRLLINT)
```

Example 2
Operation: Put the current CPU lock use count for this processor into register 3.

```
SETLOCK TEST,TYPE=CPU,LOCKHLD=(3)
```

SETRP — Set Return Parameters

The SETRP macro is used within a recovery routine to indicate the various requests. It may be used only if a system diagnostic work area (SDWA) was passed to the recovery routine. The macro is valid for functional recovery routines (FRRs) and ESTAE type recovery routines.

If you are executing in 31-bit addressing mode, you must use the MVS/ESA version of this macro.

The SETRP macro is also described in *Application Development Macro Reference* with the exception of the RECORD, RECPARM, FRELOCK, CPU, SERIAL, and RETRY parameters. These parameters are restricted in use to programs executing as FRRs in supervisor state or key 0-7 and, therefore, are only described here.

Note: This macro requires that the IHASDWA mapping macro be assembled as a DSECT in the caller's program. The SDWA is addressable when the recovery routine is entered; when the SETRP macro is issued, the same address space must be addressable.

Programs in primary, secondary, and access register (AR) mode can issue SETRP. If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before you issue SETRP. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode. (SETRP defaults to primary/secondary mode.) There is one exception to using SETRP: If your program is in secondary mode, do not use the DUMPOPX parameter.

For more information on how to use SETRP, see the chapter on recovery routines in *SPL: Application Development Guide*.

The SETRP macro is written as follows:

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

WKAREA=*(reg)*	*reg:* decimal digits 1-12. **Default:** WKAREA=(1)
,REGS=*(reg1)* ,REGS=*(reg1,reg2)*	*reg1:* decimal digits 0-12, 14, 15. *reg2:* decimal digits 0-12, 14, 15. **Note:** If *reg1* and *reg2* are both specified, order is 14, 15, 0-12.
,DUMP=IGNORE ,DUMP=YES ,DUMP=NO	**Default:** DUMP=IGNORE
,DUMPOPT=*parm list addr* ,DUMPOPX=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12). **Note:** Specify these parameters only if you specify DUMP=YES.
,RC=0 ,RC=4 ,RC=16	**Default:** RC=0
,RETADDR=*retry addr*	*retry addr:* RX-type address, or register (2) - (12). **Note:** This parameter may be specified only if RC=4 is specified above.

,RETREGS=NO ,RETREGS=YES ,RETREGS=YES,RUB=*info addr*	*info addr:* RX-type address, or register (2) - (12). **Default:** RETREGS=NO **Note:** This parameter may be specified only if RC=4 is specified above. If RETREGS=YES is specified for a FRR, all registers are restored from SDWASRSV with the exception of register 15. Register 15 always contains the entry point of the retry routine.
,FRESDWA=NO ,FRESDWA=YES	**Default:** FRESDWA=NO **Note:** This parameter may be specified only if RC=4 is specified above.
,COMPCOD=*code* ,COMPCOD=*(code,* USER) ,COMPCOD=*(code*,SYSTEM)	*code:* symbol, decimal digit, or register (2) - (12). **Default:** COMPCOD=*(code*,USER)
,FRELOCK=*(locks)*	*locks:* any combination of the following, separated by commas: CPU CMS LOCAL CML(*cmlascb*) *cmlascb:* RX-type address or register (2) - (12).
,REASON=*code*	*code:* symbol, decimal or hexadecimal number, or register (2) - (12).
,CPU=*reg*	*reg:* decimal digits 2-12.
,RECORD=IGNORE ,RECORD=YES ,RECORD=NO	**Default:** RECORD=IGNORE
,RECPARM=*record list addr*	*record list addr:* RX=type address, or register (2) - (12). **Note:** This parameter may be specified only if RECORD=IGNORE or RECORD=YES is specified above.
,SERIAL=YES ,SERIAL=NO	
,RETRY=FRR ,RETRY=ERROR	**Default:** RETRY=FRR
,RETRY15=NO ,RETRY15=YES	**Default:** RETRY15=NO
,REMREC=NO ,REMREC=YES	**Default:** REMREC=NO
,FRLKRTY=NO ,FRLKRTY=YES	**Default:** FRLKRTY=NO

The parameters are explained below:

,WKAREA=(*reg*)
specifies the address of the SDWA passed to the recovery exit. If this parameter is omitted, the address of the SDWA must be in register 1.

,REGS=(*reg* 1)
,REGS=(*reg* 2)
specifies the register or range of registers to be restored from the save area pointed to by the address in register 13. If REGS is specified, a branch on register 14 instruction will also be generated to return control to the system. If REGS is not specified, the user must code his own return.

,DUMP = IGNORE
,DUMP = YES
,DUMP = NO

specifies that the dump option fields will not be changed (IGNORE), will be zeroed (NO), or will be merged with dump options specified in previous dump requests, if any (YES). If IGNORE is specified, a previous exit had requested a dump or a dump had been requested via the ABEND macro, and the previous request will remain intact. If NO is specified, no dump will be taken.

,DUMPOPT = *parm list addr*
,DUMPOPX = *parm list addr*

specifies the address of a parameter list of dump options. You can create the parameter list through the list form of the SNAP or SNAPX macro, or you can create a compatible list. DUMPOPT specifies the address of a parameter list that the SNAP macro creates. DUMPOPX specifies the address of a parameter list that the SNAPX macro creates. A program in secondary mode cannot use the DUMPOPX parameter.

If the specified dump options include subpools for storage areas to be dumped, up to seven subpools can be dumped. Subpool areas are accumulated and wrapped, so that the eighth subpool area specified replaces the first. The TCB, DCB, and STRHDR options available on SNAP or SNAPX are ignored if they appear in the parameter list. The TCB used will be the one for the task that suffered the error. The DCB used will be one created by the system, and either SYSABEND, SYSMDUMP, or SYSUDUMP will be used as a DDNAME.

,REASON = *code*

specifies the reason code that the user wishes to pass to subsequent recovery exits. The value range for *code* is any 32-bit hexadecimal number or 31-bit decimal number. See *Application Development Macro Reference* for information about how a user can change this code.

,RC = 0
,RC = 4
,RC = 16

specifies the return code the recovery routine sends to the system to indicate what further action is required:

Hexadecimal Code	Meaning
0	Continue with termination, causes entry into previously-specified recovery routine, if any.
4	Retry using the retry address specified.
16	Suppress further ESTAI/STAI processing (for ESTAI only)

,RETADDR = *retry addr*

specifies the address of the retry routine to which control is to be given.

,RETREGS = NO
,RETREGS = YES
,RETREGS = YES ,RUB = *reg info addr*

specifies the contents of the registers to be restored on entry to the retry routine. RETREGS = NO (the default) indicates that you do not want the system to restore any register contents from the SDWA. If YES is specified, the contents of the SDWASRSV field will be used to initialize registers 0-14 when an FRR requests retry and registers 0-15 when an ESTAE requests retry. For ESTAE exits, this field contains the registers at the last interruption of the RB level at which retry will occur. For ESTAI exits, the contents of SDWASRSV must be set by the user either before SETRP is issued or by use of the RUB parameter; any field not set will cause the corresponding register to contain 0 on entry to the retry routine.

RUB specifies the address of an area that contains register update information. The system will move the data specified in this area into the SDWA and into the general purpose registers before entry to the retry routine.

The maximum length of the RUB is 66 bytes:

- The first two bytes represent the registers to be updated, register 0 corresponding to bit 0, register 1 corresponding to bit 1, and so on. The user indicates which of the registers are to be stored in the SDWA by setting the corresponding bits in these two bytes.
- The remaining 64 bytes contain the update information for the registers, in the order 0-15. If all 16 registers are being updated, this field consists of 64 bytes. If only one register is being updated, this field consists of only 4 bytes for that one register.

For example, if only registers 4, 6, and 9 are being updated:

- Bits 4, 6, and 9 of the first two bytes are set.
- The remaining field consists of 12 bytes for registers 4, 6, and 9; the first 4 bytes are for register 4, followed by 4 bytes for register 6, and 4 final bytes for register 9.

,FRESDWA = NO
,FRESDWA = YES
specifies that the entire SDWA be freed (YES) or not be freed (NO) before entry into the retry routine.

,COMPCOD = *comp code*
,COMPCOD = *(comp code*,**USER)**
,COMPCOD = *(comp code*,**SYSTEM)**
specifies the user or system completion code that the user wants to pass to subsequent recovery exits.

,FRELOCK = *(locks)*
specifies the locks to be freed and the corresponding lockwords that are placed in the SDWA:

CPU	Processor lock
CMS	Cross memory services lock
LOCAL	Storage lock of the storage the caller is executing in
CML(*cmlascb*)	Cross memory local lock, where *cmlascb* indicates the ASCB address of the address space for which the local lock is to be freed

Notes:

1. If FRLKRTY = NO is specified or taken as a default, the specified locks are freed only on percolation, not on retry. Specifying FRLKRTY = YES allows the locks listed in FRELOCK to be freed on retry.
2. Certain MVS services may require you to free one or more of the other locks shown in the syntax diagram. If a specific MVS service calls for one of these lock types, code SETRP as shown in the diagram.

,CPU = (*reg***)**
specifies the register that contains the logical processor identification of the processor holding the resource that this processor is waiting for.

,RECORD = IGNORE
,RECORD = YES
,RECORD = NO
specifies that the entire SDWA (fixed, base, variable areas, and extensions) is to be written on SYS1.LOGREC (YES), is not to be written on SYS1.LOGREC (NO), or is to be written as indicated prior to the SETRP macro (IGNORE).

,RECPARM = *record list addr*

specifies the address of a user-supplied record parameter list used to update the SDWA with recording information. The parameter list consists of three 8-byte fields:

- The first field contains the load module name.
- The second field contains the CSECT name (assembly module name).
- The third field contains the recovery routine name (assembly module name). If the recovery routine label is not the same as the assembly module name, the label can be placed in the SDWARRL field.

The three fields are left-justified, and padded with blanks.

,SERIAL = YES
,SERIAL = NO

specifies whether the percolation from an SRB mode FRR to a related task recovery routine (ESTAE or FRR) is to be serialized (YES) or not serialized (NO) with respect to unlocked task recovery. See "SRB to Task Percolation" in *SPL: Application Development Guide*.

If the task is already in recovery for another error when SERIAL = YES is specified, the percolation request is deferred pending a requested task retry from any recovery routine covering mainline code. If such a retry is not requested, the task is terminated and all deferred percolations are purged. Only the last FRR to receive control when an error occurs can specify SERIAL = YES.

,RETRY = FRR
,RETRY = ERROR

specifies the cross memory environment in which the retry routine gets control.

RETRY = FRR, the default, specifies that the retry routine gets control in the cross memory environment that exists at the time of entry to the FRR.

RETRY = ERROR specifies that the retry routine gets control in the cross memory environment that existed at the time of the error. Do not specify RETRY = ERROR if the cross memory status at the time of the error is not available, that is, if SDWARPIV is set to one. (Be careful not to create a loop by retrying to an erroneous cross memory state with RETRY = ERROR.)

,RETRY15 = YES
,RETRY15 = NO

In an FRR environment only, specifies that register 15 is restored from SDWASRSV if RETRY15 = YES. Otherwise, it contains the entry point address of the retry routine.

This parameter may be specified only when RC = 4 is specified. If RETRY15 = YES is not coded on any SETRP invocation prior to returning to the system, the effect is that of specifying RETRY15 = NO.

,REMREC = YES
,REMREC = NO

In an FRR or ESTAE environment, specifies that the FRR/ESTAE entry for the currently running FRR/ESTAE routine be removed (REMREC = YES) or not removed (REMREC = NO). This parameter may be specified only when RC = 4 is specified, indicating a retry request.

The entry is removed before control returns to the retry point. If REMREC = YES is not coded on any SETRP invocation before the system receives control, the effect is that of specifying REMREC = NO. The REMREC parameter may be used to remove a recovery routine that has been established with a token, although the token cannot be specified when you code the SETRP macro.

,FRLKRTY = YES
,FRLKRTY = NO

In an FRR environment only, specifies that the locks specified on FRELOCK be freed (FRLKRTY = YES) or not be freed (FRLKRTY = NO) on retry.

This parameter may be specified only when RC=4 is specified. If FRLKRTY=YES is not coded on any SETRP invocation prior to returning to the system, the effect is that of specifying FRLKRTY=NO.

Notes:

1. The variable recording area (SDWAVRA) contains the variable information that is supplied by the user. This consists of footprints or other information about the execution environment at the time of the failure. The execute form of the VRADATA macro and the IHAVRA mapping macro can be used to supply this data in a key-length-data format to simplify later decoding. The variable recording area is preceded by the following control information:
   - A two-byte length field (SDWAVRAL), filled in by the system, specifying the total length available to the user. This is 255 bytes, the length of the SDWAVRA field.
   - A one-byte flag field (SDWADPVA), filled in by the user, specifying the format of the data to be dumped. The flags used to specify the format are:

     SDWAHEX for hexadecimal format
     SDWAEBC for EBCDIC format
     SDWAVRAM for key-length-data

     More than one of these flags can be set. If the SDWAEBC flag is set, the EREP program formats the SDWAVRA in EBCDIC and hexadecimal for SYS1.LOGREC output.
   - A one-byte length field (SDWAURAL), filled in by the user, specifying the actual length of the data.
2. The FRESDWA parameter cannot be specified or defaulted for a functional recovery routine (FRR). The SDWA is always released before an FRR's retry routine gets control.
3. The SERIAL parameter is relevant only for FRRs established for SRBs that have a related task.
4. The SERIAL and RETRY parameters are mutually exclusive.
5. SETRP does the following in response to requests to alter the completion code and/or reason code:
   - If the COMPCOD parameter is altered, SETRP places the new completion code in the SDWACMPC field of the SDWA and sets the SDWACCF flag to indicate that a recovery exit altered the completion code.
   - If the REASON parameter is specified, SETRP places the new reason code in SDWACRC and sets the SDWAREAF flag to indicate that a recovery exit altered the reason code.

The following table indicates which parameters are available to functional recovery routines (FRRs) and which parameters are available to ESTAE-type recovery routines.

Parameter	FRR	ESTAE-type recovery routines
WKAREA	x	x
REGS	x	x
DUMP	x	x
REASON	x	x
RC=0	x	x
RC=4	x	x
RC=16		x
RETADDR	x	x
RETREGS	x	x
RUB	x	x
FRESDWA		x
COMPCOD	x	x
FRELOCK	x	
CPU	x	
RECORD	x	x
RECPARM	x	x
SERIAL	x	
RETRY	x	

Example 1

Operation: Cause a restart interruption on the processor identified by the contents of register 7. In this example, the interrupted function is spinning on a lock currently being held by the processor identified in register 7.

```
SETRP CPU=(7)
```

Example 2

Operation: The first FRR established for an SRB routine requests percolation, freeing of the CML lock (the ASCB address is in register 2), and serialization of percolation to the related task.

```
SETRP  RC=0,FRELOCK=(CML(2)),SERIAL=YES
```

Example 3

Operation: An FRR requests retry with the retry routine getting control in the same cross memory mode as the time of FRR entry. The retry address is in register 3.

```
SETRP  RC=4,RETADDR=(3),RETRY=FRR
```

SPIE — Specify Program Interruption Exit

Note: The ESPIE macro is the preferred programming interface.

The SPIE macro specifies the address of an interruption exit routine and the program interruption types that are to cause the exit routine to get control. If the program interruption types specified can be masked, the corresponding program mask bit in the PSW (program status word) is set to 1.

Only callers in 24-bit addressing mode can issue the SPIE macro. If a caller in 31-bit addressing mode issues a SPIE macro, the caller is abended with a system completion code of X'30E'. Callers in 31-bit addressing mode must use the ESPIE macro, which performs the same function as the SPIE macro for callers in both 24-bit and 31-bit addressing mode.

Note: In MVS/370 the SPIE environment existed for the life of the task. In later versions of MVS, the SPIE environment is deleted when the request block that created it is deleted. That is, when a program running under MVS/XA completes, any SPIE environments created by the program are deleted. This might create an incompatibility with MVS/SP Version 1 for programs that depend on the SPIE environment remaining in effect for the life of the task rather than the request block.

The SPIE macro is not supported in cross memory mode.

The following description of the SPIE macro also appears in *Application Development Macro Reference*, with the exception of interruption type 17. This interruption type designates page faults and its use is restricted to authorized programs. For more information about the SPIE macro, see the chapter on interruptions in *SPL: Application Development Guide*.

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

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

exit addr	
,(interrupts)	*exit addr:* A-type address, or register (2) - (12).
	interrupts: decimal numbers 1-15, or 17 expressed as
	single values: (2,3,4,7,8,9,10)
	ranges of values: ((2,4),(7,10))
	combinations: (2,3,4,(7,10))

The parameters are explained as follows:

exit addr
specifies the address of the exit routine to be given control when a specific program interruption occurs. The exit routine receives control in 24-bit addressing mode.

,(*interrupts*)

indicates the type of interruption for which the exit routine is to be given control. The interruption types are as follows:

Number	Interruption Type
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow (maskable)
9	Fixed-point divide
10	Decimal overflow (maskable)
11	Decimal divide
12	Exponent overflow
13	Exponent underflow (maskable)
14	Significance (maskable)
15	Floating-point divide
17	Page fault

Notes:

1. If a specified program interruption type is maskable, the corresponding bit is set to 1. Interruption types not specified above are handled by the system.
2. The system returns the address of the previous PICA or a PICA in which the first word contains binary zeroes in register 1. If no previous SPIE environment existed, the system returns zeros in register 1.
3. If an exit address is zero or no parameters are specified, the current SPIE and any previously active ESPIE environments are canceled.
4. If you are using vector instructions and an interruption of 8, 12, 13, 14, or 15 occurs, your recovery routine can check the exception extension code (the first byte of the two-byte interruption code in the EPIE or PIE) to determine whether the exception was a vector or scalar type of exception.

Example

Operation: Give control to an exit routine for interruption 17. DOITSPIE is the address of the SPIE exit routine.

```
SPIE DOITSPIE,(17)
```

SPIE (List Form)

Use the list form of the SPIE macro to construct a control program parameter list in the form of a program interruption control area.

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

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

exit addr	*exit addr:* A-type address.
,(interrupts)	*interrupts:* decimal numbers 1-15, or 17, expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4),(7,10)) **combinations:** (2,3,4,(7,10))
,MF=L	

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

,MF=L
specifies the list form of the SPIE macro.

SPIE (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the SPIE macro. The PICA (program interruptions control area) can be generated by the list form of SPIE, or you can use the address of the PICA returned in register 1 following a previous SPIE macro. If this macro is being issued to reestablish a previous SPIE environment, code only the MF parameter.

The address of the remote control program parameter list associated with any previous SPIE environment is returned by the SPIE macro.

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

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

exit addr	*exit addr:* RX-type address, or register (2) - (12).
,(interrupts)	*interrupts:* decimal numbers 1-15, or 17, expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4),(7,10)) **combinations:** (2,3,4,(7,10))
,MF=(E,*ctrl addr)*	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl*,*addr*)
specifies the execute form of the SPIE macro using a remote control program parameter list.

Note: If SPIE is coded with a 0 as the control address, the SPIE environment is canceled.

SPLEVEL — SET and TEST Macro Level

Specific macros supplied in the macro library are identified as downward incompatible (to MVS/SP Version 1 or 2). Unless you take specific action, these macros generate downward incompatible statements.

It is possible to generate downward compatible expansions of these macros by using the SPLEVEL macro. The downward incompatible macros interrogate a global symbol (set by SPLEVEL) during assembly to determine the type of expansion to be generated. See " Selecting the Macro Level" on page 1 for additional information about the downward incompatible macros and *Assembler H Version 2 Application Programming: Language Reference* for information about global set symbols.

Existing programs that issue MVS/SP Version 2 macros will execute properly in SP Version 3. MVS/SP Version 3 macros will execute properly without your issuing the SPLEVEL macro.

The SPLEVEL macro is written as follows:

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

SET=*n* SET TEST	*n:* 1, 2 or 3. **Default:** SET=3

The parameters are explained as follows:

SET=*n*
SET
TEST
specifies whether the macro level is being set or tested.

If SET=*n* is specified, the SPLEVEL routine sets a global set symbol equal to *n*, where *n* must be 1, 2 or 3. If a user codes one of the downward incompatible macros, one of the following macro expansions is generated:

- The MVS/SP Version 1 macro expansion if *n*=1
- The MVS/SP Version 2 macro expansion if *n*=2
- The MVS/SP Version 3 macro expansion if *n*=3

If SET is specified without *n*, the SPLEVEL routine uses the default value, 3.

The TEST option is used to determine the macro level that is in effect. The results of the test request are returned to the user in the global set symbol, &SYSSPLV. If TEST is specified and if SPLEVEL SET has not been issued during this assembly, the SPLEVEL routine puts the default value into the global set symbol. If SPLEVEL SET has been issued, the previous value of *n* or the default value is already in the global set symbol.

Example 1

Operation: Select the SP Version 1 version of a specific downward incompatible macro.

```
SPLEVEL SET=1
```

Example 2

Operation: Use SPLEVEL to select the MVS/SP Version 3 version of the MVS macros:

```
* Determine which level of MVS is executing
         TM   CVTDCB,CVTOSEXT
         BNO  SP2
         TM   CVTOSLV0,CVTXAX
         BNO  SP2
* The Version 3 level of MVS is executing
SP3      EQU   *
         SPLEVEL SET=3
         SAC   512
         SYSSTATE ASCENV=AR
         DSPSERV CREATE...
         ALESERV ADD...
         B    CONTINUE
* A Version 1 or 2 level of MVS is executing
SP2      EQU   *
         SPLEVEL SET=2
         GETMAIN ....
         SPLEVEL SET=3
CONTINUE EQU   *
```

A good coding practice is to always reset SPLEVEL to the default value after you have changed it as soon as the special value is not needed. Consider the case where, later in your program, you have another sequence like this with some other macros. The default value should be the one in effect through most of the program, except when specifically required to be otherwise.

SPOST — Synchronize POST

The SPOST macro is used in a cross-memory post environment to ensure that all outstanding cross-memory post requests to the current address space have completed. SPOST resolves a synchronization problem that arises when it becomes necessary to free an ECB that is a potential target for a cross-memory post request. Before issuing SPOST, you must stop any new posts from being initiated.

For explanation of the parameters in a cross-memory post request, see the POST macro.

The SPOST macro is written as follows:

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

Note: SPOST contains no optional or required parameters.

Example

Operation: Execute the SPOST macro with a comment.

```
SPOST                    ,ISSUE SPOST
```

SRBSTAT — Save, Restore, or Modify SRB Status

The SRBSTAT macro allows the caller to save, restore, and modify the status of an SRB in a caller-supplied save area. The caller must be running in SRB mode to use the SAVE or RESTORE option. The caller can be running either in SRB or TCB mode to use the MODIFY option. In either mode, the caller must be in supervisor state, key 0, primary ASC mode, have authority to issue a SSAR instruction to the home address space, and be enabled and unlocked. Register 13 must point to a 72-byte save area addressable in the primary address space. Control returns from the SRBSTAT macro in primary ASC mode.

The SRBSTAT macro is written as follows:

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

SAVE RESTORE MODIFY	
,STSV=*stsv addr*	*stsv addr:* RX-type address or register (1) - (12), register (1) preferred.
,STSV=0	
,NEWFRR=*addr*	*addr:* RX-type address or register (0) or (2) - (12), register (0) preferred.
,PRGAT=*pat addr*	*pat addr:* RX-type address or register (2) - (12), register (2) preferred.

The parameters are explained as follows:

SAVE
RESTORE
MODIFY
specifies whether a save, restore, or modify operation is requested. For SAVE or RESTORE, only the following status is saved or restored:

- General and floating point registers
- Control registers 3 and 4
- CPU affinity mask
- Related ASID/TCB
- Timing information
- FRR stack
- PCLINK stack header

If SAVE is specified, only caller's registers 1 and 15 are destroyed. Register 1 is used to hold an FRR parameter area address if NEWFRR is also specified and register 15 is used for a return code. The PCLINK stack header is saved and zeroed.

If RESTORE is specified, registers 0-13 are restored. The contents of register 14 are the same as when RESTORE was entered. The current PCLINK stack header must be zero; the saved one is restored.

On entry to RESTORE, the PCLINK stack header must be zero. RESTORE cannot be used in an FRR. Note that RESTORE returns to its caller and not to the caller of SAVE. Note that SRBSTAT does not save and restore access registers, extended authorization index (EAX) value, and linkage stack and access list status.

,STSV = *stsv addr*

specifies the address of the save area to be used for the SAVE, RESTORE, or MODIFY operation. The save area can be in private pageable storage, but it must be addressable from the home address space and it must begin on a double word boundary. For RESTORE or MODIFY, the save area must contain valid status.

,STSV = 0

specifies that the current status is to be modified. This parameter is valid only with MODIFY.

For MODIFY, an existing SRB status save area or the current status is modified. Only the purge ASID/TCB information can be modified. All registers are saved and restored except register 15, which contains a return code.

Hexadecimal Code	Meaning
00	The modify function was successfully completed.

,NEWFRR = *addr*

specifies the address of an FRR established with MODE = FULLXM. For SAVE, the address of the FRR parameter area is returned to the caller in register 1. The first word of the parameter area contains the address of the SRB status save area being used.

For RESTORE, the FRR address is used only if the saved status cannot be reinstated on the current processor. An SRB with the FRR option is scheduled specifying this FRR.

For MODIFY, this parameter is invalid.

,PRGAT = *pat addr*

specifies the address of a 6-byte area of storage, currently addressable in the primary address space, that contains the new purge ASID/TCB. Bytes 1 and 2 contain the ASID; bytes 3-6 contain the TCB address. This parameter is required with MODIFY but is invalid with SAVE or RESTORE.

SRBTIMER — Establish Time Limit for System Service

The SRBTIMER macro is used to establish a time limit for a system service running in SRB mode. Time accumulates while the service is running; when the time limit expires, the service abends with a system completion code of X'05B'. The service can retry following the 05B ABEND.

The caller can cancel an established time limit by reissuing the macro and specifying a time limit of zero. The caller can also override the established time limit with a subsequent SRBTIMER macro.

The caller must be in supervisor state, SRB mode, and key 0. Register 13 must point to a 72-byte save area. Programs running in primary ASC mode and in either 24-bit or 31-bit addressing mode can issue the SRBTIMER macro. Programs running in AR ASC mode cannot issue SRBTIMER. The save area must be addressable in the addressing mode in which the macro is issued.

The SRBTIMER macro is written as follows:

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

LIMIT=*stor addr*	*stor addr:* RX-type address or register (0) or (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr:* RX-type address or register (2) - (12).

The parameters are explained as follows:

LIMIT = *store addr*
specifies the virtual storage address of a doubleword field on a doubleword boundary that contains the time limit. The time limit is in the form of a signed 64-bit binary number and must be positive in order for time to elapse. A negative number causes immediate expiration of the time limit. Bit 51 of the binary number is approximately equivalent to one microsecond. If you specify a value greater than 208 days, the control program changes the value to 208 days. The resolution of the timer is model dependent. See *Principles of Operation* for details concerning the timer facility.

,ERRET = *err rtn addr*
specifies the address of the routine to be given control when the SRBTIMER function encounters damaged clocks.

Register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	The time limit was successfully established.
04	The current processor has an operative CPU timer, but not all processors have an operative CPU timer.
08	The current processor has an inoperative CPU timer, but not all processors have an inoperative CPU timer.
0C	All processors in the system have an inoperative CPU timer.
10	The issuer is not in SRB mode. No timing is performed.

STAE — Specify Task Abnormal Exit

Note: The ESTAE macro is the preferred programming interface.

The STAE macro enables the user to intercept a scheduled ABEND and to have control returned to him at a specified exit routine address. The STAE macro operates in both problem program and supervisor modes.

Note: The STAE macro is not supported for users executing in 31-bit addressing mode. Such users will be abended.

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

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

0 *exit addr*	*exit addr:* A-type address, or register (2) - (12).
,CT ,OV	**Default:** CT
,PARAM=*list addr*	*list addr:* A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	**Default:** PURGE=QUIESCE
,ASYNCH=NO ,ASYNCH=YES	**Default:** ASYNCH=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

0
exit addr
specifies the address of a STAE exit routine to be entered if the task issuing this macro terminates abnormally. If 0 is specified, the most recent STAE request is canceled.

,CT
,OV
specifies the creation of a new STAE exit (CT) or indicates that the parameters passed in this STAE macro are to overlay the data contained in the previous STAE exit (OV).

,PARAM = *list addr*
specifies the address of a user-defined parameter list containing data to be used by the STAE exit routine when it is scheduled for execution.

,XCTL=NO
,XCTL=YES
specifies that the STAE macro will be canceled (NO) or will not be canceled (YES) if an XCTL macro is issued by this program.

,PURGE = QUIESCE
,PURGE = HALT
,PURGE = NONE

specifies that all outstanding requests for I/O operations are not saved when the STAE exit is taken (HALT), that I/O processing is allowed to continue normally when the STAE exit is taken (NONE), or that all outstanding requests for I/O operations are saved when the STAE exit is taken (QUIESCE). For QUIESCE, at the end of the STAE exit routine, the user can code a retry routine to handle the outstanding I/O requests.

Note: If any IBM-supplied access method, except EXCP, is being used, the PURGE = NONE option is recommended. If you use PURGE = NONE, all control blocks affected by input/output processing can continue to change during STAE exit routine processing.

If PURGE = NONE is specified and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion develops when an input/output interruption occurs, even if the exit routine is in progress. Thus, it appears that the exit routine failed when, in reality, input/output processing caused the failure.

ISAM Notes: If ISAM is being used and PURGE = HALT is specified or PURGE = QUIESCE is specified but I/O is not restored:

- Only the input/output event on which the purge is done is posted. Subsequent event control blocks (ECBs) are not posted.
- The ISAM check routine treats purged I/O as normal I/O.
- Part of the data set may be destroyed if the data set is being updated or added to when the failure occurred.

,ASYNCH = NO
,ASYNCH = YES

specifies that asynchronous exit processing is allowed (YES) or is not allowed (NO) while the STAE exit is executing.

ASYNCH = YES must be coded if:

- The STAE exit routine requests any supervisor services that require asynchronous interruptions to complete their normal processing.
- PURGE = QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE = NONE is specified and the CHECK macro is issued in the STAE exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH = YES is specified and the ABEND was originally scheduled because of an error in asynchronous exit handling, an ABEND recursion develops when an asynchronous interruption occurs. Thus, it appears that the exit routine failed when, in reality, asynchronous exit handling caused the failure.

,RELATED = *value*

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

Control returns to the instruction following the STAE macro; register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of STAE request.
04	STAE was unable to obtain storage for STAE request.
08	Attempt was made to cancel or overlay a nonexistent STAE request.
0C	Exit routine or parameter list address was invalid, or STAI request was missing a TCB address.
10	Attempt was made to cancel or overlay a STAE request of another user, or an unexpected error was encountered while processing this request.

Example

Operation: Request an overlay of the existing STAE recovery exit with the following options: new exit address is ADDR, parameter list is at PLIST, halt I/O, do not take asynchronous exits, transfer ownership to the new request block resulting from any XCTL macros.

```
STAE ADDR,OV,PARAM=PLIST,XCTL=YES,PURGE=HALT,ASYNCH=NO
```

STAE (List Form)

The list form of the STAE macro is used to construct a remote control program parameter list.

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

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede STAE.
STAE	
ƀ	One or more blanks must follow STAE.
exit addr	*exit addr:* A-type address.
,PARAM=*list addr*	*list addr:* A-type address.
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	**Default:** PURGE=QUIESCE
,ASYNCH=NO ,ASYNCH=YES	**Default:** ASYNCH=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the STAE macro.

STAE (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the STAE macro. The control program parameter list can be generated by the list form of the STAE macro. If you want to dynamically change the contents of the remote STAE parameter list, you can do so by coding a new exit address and/or a new parameter list address. If exit address or PARM= is coded, only the associated field in the remote STAE parameter list is changed. The other field remains as it was before the current STAE request was made.

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

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

exit addr 0	*exit addr:* RX-type address, or register (2) - (12).
,CT ,OV	
,PARAM=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	
,ASYNCH=NO ,ASYNCH=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*ctrl addr)*	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E, *ctrl addr***)**
specifies the execute form of the STAE macro using a remote control program parameter list..

Example

Operation: Provide the pointer to the recovery code in the register called EXITPTR, and the address of the STAE exit parameter list in register 9. Register 8 points to the area where the STAE parameter list (created with the MF=L option) was moved.

```
STAE (EXITPTR),PARAM=(9),MF=(E,(8))
```

STATUS — Change Subtask Status

You can use the STATUS macro to change the dispatchability status of one of your program's subtasks.

The STATUS macro is also described in the *Application Development Macro Reference*, with the exception of the SRB, ASID, and TASK parameters, which are restricted in use and available only to supervisor state, key zero callers. These restricted parameters allow the caller to manipulate the dispatchability of TCBs, SRBs, ASCBs, or a STEP.

The SYNCH operand of STATUS STOP is not supported in MVS/XA or MVS/ESA™ Programs that issue STATUS STOP, SYNCH should be changed to issue STATUS STOP without the SYNCH operand. Users who specify the SYNCH operand with STATUS STOP will receive an MNOTE of severity 12 at assembly time.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The description of the STATUS macro is divided into two parts: the START/STOP option, and the SET/RESET option.

The START/STOP options of the STATUS macro are written as follows:

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

START STOP	
,TCB=*tcb addr*	*tcb addr:* RX-type address, or register (2) - (12), or 0.
,SRB	*ASID addr:* RX-type address, or register (2) - (12).
,SRB, ASID=*ASID addr*	**Note:** ASID may only be specified with START.
,SRB,TASK=YES	**Default:** TASK=YES
,SRB,TASK=NO	
,SRB,TASK=YES,ASID=*ASID addr*	
,SRB,TASK=NO,ASID=*ASID addr*	
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are described as follows:

START
STOP
specifies that the appropriate START/STOP count is to be adjusted and the dispatchability bits are to be set/reset.

,TCB = *tcb addr*
,SRB
,SRB,ASID = *ASID addr*
specifies the status of the stop/start function:

TCB
: specifies the address of a fullword on a fullword boundary containing the address of the TCB that is to have its START/STOP count adjusted.

  Note: The TCB resides in storage below 16 megabytes.

SRB
: specifies that the STOP function affects the dispatchability of system-level SRBs only; all other tasks in the address space are set/reset nondispatchable. For START, the ASID addr specifies the address of a halfword containing the address space identifier. If ASID is passed in a register, it must be in bit positions 16-31, and bits 0-15 must be zero.

TASK=
: specifies whether the STATUS, STOP, and START functions affect the dispatchability of all other tasks in the address space. TASK=YES is the default. If TASK=YES is specified or defaulted, STATUS sets or resets task dispatchability in the address space. TASK=NO requests STATUS to ignore setting or resetting task dispatchability. TASK=NO modifies only system level SRB dispatchability and not TCB dispatchability. TASK=NO has the following restrictions:

  - Issuers of STATUS must ensure that the dispatchability of all other tasks in the address space need not be modified.
  - Issuers must be in key 0.
  - Issuing programs must include the IHAASCB mapping macro.

,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.

SET/RESET Options

The SET/RESET options of the STATUS macro are written as follows:

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

SET RESET	
,MC ,MC,STEP ,SD ,ND	**Note:** If MC or MC,STEP is specified, no other parameters can be specified. *mask:* for SD, any of decimal digits 1-32 (except 18), separated by commas; for ND, any of decimal digits 1-16 (except 14), separated by commas.
,STEP ,STEP,*(mask)* ,*tcb addr*,*(mask)* ,,*(mask)*	*tcb addr:* RX-type address, or register (2) - (12). **Default:** STEP
,E	**Note:** This parameter can only be specified with *tcb addr,(mask)*.
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12). **Note:** For SET, this parameter can only be specified with *tcb addr,(mask)*.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

SET
RESET
specifies that the TCBs or ASCBs are to be set or reset nondispatchable.

,MC
,MC,STEP
,SD
,ND
specifies the nondispatchability status:

ND specifies that the primary nondispatchability bits are affected by this request.

SD specifies that the secondary nondispatchability bits are affected by this request.

MC and MC,STEP specifies that all TCBs in the job step TCBs (except the issuer's TCB) are to be set/reset nondispatchable.

,STEP
,STEP,*(mask)*
,tcb addr,(mask)
,,(mask)
specifies more information on the nondispatchability status:

STEP specifies that all job step TCBs (except the issuer's TCB) are to be set/reset nondispatchable.

tcb addr indicates that the specified TCB (except the issuer's TCB) and all its subtasks are to be set/reset nondispatchable.

(mask) specifies the nondispatchability bits that are to be set/reset.

,E specifies that only the specified TCB is to be set/reset nondispatchable.

,ASID = ASID *addr*
specifies the address of a halfword containing the address space identifier. If ASID is passed in a register, it must be in bit positions 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 may be any valid coding values.

Example

Operation: Set primary nondispatchability bit 3 for the specified TCB and all its subtasks.

```
STATUS SET,ND,TCBADDR,(3)
```

STORAGE — Obtain and Release Storage

The STORAGE macro requests that the system obtain or release an area of virtual storage. The two forms of the macro are:

- STORAGE OBTAIN, which obtains virtual storage.
- STORAGE RELEASE, which releases virtual storage.

The requirements for the caller are:

Authorization:	Supervisor state, key 0 - 7, or problem state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN or PASN not = HASN
Amode:	Any
ASC mode:	Primary or AR
Serialization:	Enabled; caller can hold local lock or CML lock of target address space.

The STORAGE macro uses general purpose registers (GPRs) 0, 1, 14, and 15 and access register (AR) 1 as parameter and linkage registers. It preserves GPRs 2-13 and ARs 2-13.

The STORAGE macro is also described in *Application Development Macro Reference*, with the exception of the KEY and ALET parameters. These parameters are restricted to programs running in supervisor state or key 0 and, therefore, are only described here.

OBTAIN Option of STORAGE

The STORAGE macro with the OBTAIN parameter requests that the system allocate an area of virtual storage to the active task. The virtual storage area begins on a doubleword or page boundary and is not necessarily cleared to zeroes when allocated. The length you specify must not exceed the length available; the length available depends on how much storage has already been allocated, and, for subpools 0 - 127, 240, 250, 251, and 252, the region size. For some subpools, the system releases the storage when the owning task terminates. Other subpools require that you issue STORAGE RELEASE or FREEMAIN to release them. See *SPL: Application Development Guide* for a list of subpools and their attributes.

The STORAGE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede STORAGE.
STORAGE	
b	One or more blanks must follow STORAGE.
OBTAIN	
,LENGTH=*length value* ,LENGTH=*(max length,min length)*	*length value:* symbol, decimal number, or register (2)-(12). *max length:* symbol, decimal number, or register (2)-(12). *min length:* symbol, decimal number, or register (2)-(12).
,ADDR=*stor addr*	*stor addr*: RX-type address or register (1)-(12). **Default:**: ADDR=(1).
,SP=*subpool number*	*subpool number:* symbol, decimal number, or register (2)-(12). **Default:**: SP=0.
,ALET=*alet-value*	*alet-value:* decimal number, RX-type address, or access register. **Default**: ALET=0.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default**: BNDRY=DBLWD

,KEY=*key number*	*key number:* decimal number or register (2)-(12). **Default:** KEY=0 **Note:** KEY is valid only with SP.
,LOC=BELOW ,LOC=(BELOW, ANY) ,LOC=ANY ,LOC=(ANY, ANY) ,LOC=RES ,LOC=(RES, ANY)	**Default:** LOC=RES
,RTCD=*rtcd addr*	*rtcd addr:* RX-type address or register (2)-(12) or (15). **Default:** RTCD=(15).
,COND=YES ,COND=NO	**Default:** COND=NO.
,RELATED=*value*	*value:* Any valid macro parameter specification.

The parameters are explained as follows:

OBTAIN
requests that the system obtain virtual storage.

,LENGTH=*length value*
,LENGTH=(*max length,min length***)**
specifies the amount of storage the system is to obtain. *length value* specifies the length, in bytes, of the requested virtual storage. *max length* and *min length* specify the maximum and minimum amounts of storage. These numbers should be a multiple of 8; if they are not, the system uses the next higher multiple of 8.

,ADDR=*stor addr*
specifies the location where the system is to return the address of the storage it allocates.

,SP=*subpool number*
specifies the subpool number for the storage. (See *SPL: Application Development Guide* for a list of valid subpools.) If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. If you omit this parameter, the system uses subpool 0.

Notes:

1. Callers executing in supervisor state and key zero, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage via the SDUMP macro, they must specify subpool 252 rather than 0.
2. Storage requested from subpool 250 is always assigned from subpool 0 regardless of the caller's state or PSW key.

,ALET=*alet-value*
specifies the ALET of the target address space — that address space in which the storage is to index an entry obtained. The ALET must have the value 1 or 2, or be on the caller's dispatchable unit access list (DU-AL) and, if the ALET indexes a private entry, the caller must be authorized to the target address space through the extended authorization index (EAX). For more information, see *SPL: Application Development — Extended Addressability*. If you omit this parameter, the system assumes storage is in the primary address space.

,BNDRY=DBLWD
,BNDRY=PAGE
specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

If the request specifies one of the LSQA or SQA subpools, the system ignores the BNDRY=PAGE keyword. Requests for storage from these subpools are then fulfilled from a single page, unless the request is greater than a page. See *SPL: Application Development Guide* for a list of the LSQA and SQA subpools.

The default is BNDRY=DBLWD.

,KEY=*key-number*

Indicates the protection key of the storage. If you pass the key in a register, it must be in bits 24-27 in that register. KEY is valid only with SP and only applies to subpools 227-231 and 241. KEY allows you to obtain both global and local storage in the specified storage protection key.

The default for KEY is 0.

,LOC=BELOW
,LOC=(BELOW,ANY)
,LOC=ANY
,LOC=(ANY,ANY)
,LOC=RES
,LOC=(RES,ANY)

specifies the location of virtual and central (also called real) storage. This parameter is especially helpful for callers with 24-bit dependencies. In all cases when LOC is specified, central storage is allocated anywhere until the storage is fixed (by definition or by the PGFIX, PGFIXA, or PGSER macro.) You can specify the location of central storage (after the storage is fixed) and virtual storage (whether or not the storage is fixed) in the following manner.

LOC=BELOW indicates that central and virtual storage are to be located below 16 megabytes.

LOC=(BELOW,ANY) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere.

LOC=ANY and LOC=(ANY,ANY) indicate that virtual and central storage can be located anywhere.

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 located below 16 megabytes; if the caller resides above 16 megabytes, virtual and central storage are to be located anywhere.

LOC=(RES,ANY) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere. In either case, central storage can be located anywhere.

Notes:

1. A caller cannot allocate virtual storage below 16 megabytes from the following subpools: 203-205, 213-215, 223-225, 247, and 248. Thus, a caller cannot specify LOC=BELOW and LOC=(BELOW,ANY) for these subpools. Also, a caller residing below 16 megabytes cannot specify LOC=RES and LOC=(RES,ANY) for these subpools. When you specify LOC=ANY, the actual location of the virtual storage (that is, whether it is above or below 16Mb) depends on the subpool you specify on the SP parameter:

   - Some subpools (for example, subpool 226) are supported **only below** 16Mb. For these subpools, STORAGE OBTAIN locates virtual storage below 16Mb, regardless of how you specify LOC.
   - Some subpools (for example, 203-204) are supported **only above** 16Mb. For these subpools, STORAGE OBTAIN locates virtual storage above 16Mb. If you specify LOC=BELOW for one of these subpools, the system abends your program.

   All other subpools are supported both above and below 16Mb. For these subpools, specifying LOC=ANY causes STORAGE OBTAIN to try to allocate virtual storage above 16Mb. If the attempt fails, it tries to allocate virtual storage below 16Mb. If this attempt also fails, it does not allocate any storage.

,RTCD = *rtcd addr*
specifies the location where the system is to store the return code. This parameter is valid only for conditional requests.

,COND = NO
,COND = YES
specifies whether the request is unconditional or conditional.

COND = YES specifies that the task does not abend if the system cannot allocate the storage. (However, the system cannot prevent some abends.) If you specify COND = YES, also specify RTCD to define the location where the system is to store a return code.

COND = NO specifies that the system abends the task if it cannot allocate the virtual storage. COND = NO is the default.

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

When control returns from the OBTAIN request, GPR 15 contains one of the following return codes:

- 0 — Virtual storage was allocated.
- 4 — No virtual storage was allocated.
- 8 — Central storage was not available for backing the request.
- C — A page table needed to satisfy a request for LSQA is paged out.

The RELEASE Option of STORAGE

The STORAGE macro with the RELEASE parameter requests that the system release an area of virtual storage or an entire virtual storage subpool, previously allocated through the STORAGE or GETMAIN macro. The system abends the active task if the specified virtual storage does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not currently allocated to the active task.

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

RELEASE	
,LENGTH=*length value*,ADDR=*stor addr* ,LENGTH=*length value*,ADDR=*stor addr*,SP=*subpool number* ,SP=*subpool number*	*length value:* symbol, decimal number, or register (2)-(12). *stor addr*: RX-type address or register (1)-(12). *subpool number:* symbol, decimal number, or register (2)-(12).
,ALET=*alet-value*	*alet-value:* decimal number, RX-type address, access register. **Default**: ALET=0.
,KEY=*key number*	*key number:* decimal number or register (2)-(12). **Default:** KEY=0 **Note:** KEY is valid only with SP.
,RTCD=*rtcd addr*	*rtcd addr:* RX-type address or registers (2)-(12), (15) **Default:** RTCD=(15).
,COND=YES ,COND=NO	**Default:** COND=NO
,RELATED=*value*	*value:* Any valid macro parameter specification.

The parameters are explained as follows:

RELEASE
requests that the system release virtual storage.

,LENGTH = *length value*
specifies the number of bytes of storage that the system is to release. If you specify LENGTH, you must also specify ADDR. To free an entire subpool, use SP instead of LENGTH and ADDR.

,ADDR = *stor addr*
specifies the address of the storage to be released. If you specify ADDR, you must also specify LENGTH. To free an entire subpool, use SP instead of LENGTH and ADDR.

,SP = *subpool number*
specifies the subpool number for the storage to be released. The subpool number must be a valid subpool number between 0 and 255. See *SPL: Application Development Guide* for a list of valid subpools. If you specify the subpool in a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. If you omit this parameter, the system uses subpool 0.

If you specify SP to request that the system release all of the storage in a subpool, do not specify LENGTH and ADDR. (This action is called a "subpool release".) Issue subpool releases only for the following subpools: 1-127, 203, 204, 213, 214, 223, 224, 229, 230, 233, 236, 237, 240, and 250-253; and if the caller is in key 0, subpool 0. If you try to issue a subpool release for any other subpool, an abend occurs with a reason code of 478 or 40A. See *SPL: Application Development Guide* for a list of the characteristics of valid subpools.

Note: Callers executing in supervisor state and key zero, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage via the SDUMP macro, they must specify subpool 252 rather than 0.

,ALET = *alet-value*
specifies the ALET of the address space in which the storage is to be released. The ALET must be 1 or 2, or be on the caller's dispatchable unit access list (DU-AL) and, if the ALET is PRIVATE, the caller must be authorized through the extended authorization index (EAX) to the address space. For additional information, see *SPL: Application Programming — Extended Addressability*. If you omit this parameter, the system assumes storage is to be obtained in the primary address space.

,KEY = *key-number*
Indicates the protection key of the storage. If you pass the key in a register, it must be in bits 24-27 in that register. KEY is valid only with SP and only applies to subpools 227-231 and 241. KEY allows you to release both global and local storage in the specified storage protection key.

The default for KEY is 0.

,RTCD = *rtcd addr*
specifies the location where the system is to store the return code. This parameter is only valid for conditional requests.

,COND = NO
,COND = YES
specifies whether the request is unconditional or conditional.

COND = YES specifies that the task does not abend if the system cannot release the storage. However, the system cannot prevent some abends. The RTCD parameter specifies the location where the system is to store a return code. COND = NO specifies that the system abend the active task if it cannot release the storage.

COND = NO is the default.

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

When control returns from the COND=YES release request, general register 15 contains one of the following return codes:

- 0 — Virtual storage was released.
- 4 — Not all virtual storage was released.
- 8 — Part of the area being freed is still fixed.
- C — Page table is paged out.

Examples of the OBTAIN and RELEASE Options

Example 1: Code the instructions to obtain 1000 bytes of virtual storage from subpool 203 — above 16 megabytes, if available. The system returns the address of the storage in register 3. If the request fails, the system abends the caller.

```
LA      2,1000
STORAGE OBTAIN,LENGTH=(2),ADDR=(3),SP=203,LOC=ANY,COND=NO
ST      3,STRGA
```

To release the 1000 bytes obtained above from subpool 203, and abend the caller if the request fails, issue:

```
LA      2,1000
STORAGE RELEASE,LENGTH=(2),ADDR=STRGA,SP=203,COND=NO
 .
 .
STRGA   DS   F
```

Example 2: Code the instructions to obtain 4096 bytes of virtual storage from subpool 227 — above 16 megabytes, if possible. The address is returned at location STRGA. The protection key is 5. The system is to store the return code at location MY_RC.

```
STORAGE OBTAIN,LENGTH=ONE_PAGE,ADDR=STRGA,SP=MY_SUBPOOL,              X
      KEY=5,LOC=ANY,COND=YES,RTCD=MY_RC
```

To release the 4096 bytes obtained above from subpool 227, issue:

```
L       2,KEY_5
STORAGE RELEASE,LENGTH=ONE_PAGE,ADDR=STRGA,SP=MY_SUBPOOL,             X
      KEY=(2),COND=YES,RTCD=MY_RC
 .
 .
MY_RC    DS   F
STRGA    DS   F
KEY_5    DC   X'00000050'
ONE_PAGE   EQU  4096
MY_SUBPOOL EQU  227
```

Note that, when the caller passes the key in a register, the key must be in bits 24-27. Note also, that KEY=KEY_5 is not valid, as KEY_5 is not a register.

Example 3: Code the instructions to obtain 4096 bytes of virtual storage from subpool 227. Indicate that, if the system cannot obtain 4096 bytes, the caller can settle for as little as 1024 bytes. The system returns the address of the storage obtained at location STRGA. The protection key is 5. The system is to store the return code at location MY_RC.

```
STORAGE OBTAIN,LENGTH=(ONE_PAGE,ONE_K), ADDR=STRGA,                    X
              SP=MY_SUBPOOL,KEY=5,LOC=ANY,COND=YES,RTCD=MY_RC
ST      0,STRG_LEN
```

To release the storage obtained above in subpool 227, issue:

```
L       2,KEY_5
L       3,STRG_LEN
STORAGE RELEASE,LENGTH=(3),ADDR=STRGA,SP=MY_SUBPOOL,                   X
               KEY=(2),COND=YES,RTCD=MY_RC
 .
 .
STRG_LEN   DS   F
MY_RC      DS   F
STRGA      DS   F
KEY_5      DC   X'00000050'
ONE_K      EQU  1024
ONE_PAGE   EQU  4096
MY_SUBPOOL EQU  227
```

Example 4: Code the instructions to set up an 18-word save area, such as one that a program in AR address space control (ASC) mode would obtain to call a program in primary mode. The program issuing the STORAGE macro is in 31-bit addressing mode, and the code is reentrant.

```
PGM   CSECT
PGM   AMODE 31
PGM   RMODE ANY
      BAKR   14,0               SAVE CALLER'S ARS, GPRS AND RETURN
*                               ADDRESS ON LINKAGE STACK
      SAC    512                SWITCH TO AR ASC MODE
      LAE    12,0(15,0)         SET UP PROGRAM BASE REGISTER AND AR
      USING  PGM,12
      STORAGE OBTAIN,LENGTH=72  GET REENTRANT SAVEAREA
      LAE  13,0(1,0)            PUT SAVEAREA ADDRESS IN AR/GPR 13
      MVC   4(4,13),=C'F1SA'    PUT ACRONYM INTO SAVEAREA TO
*                               INDICATE STATUS SAVED ON LINKAGE STACK
      .
*   BEGIN PROGRAM CODE HERE
```

To release this save area, issue the following instructions:

```
      .
      LAE  1,0(0,13)           COPY SAVEAREA ADDRESS
      STORAGE  RELEASE,ADDR=(1),LENGTH=72  FREE SAVEAREA
      .
      SLR  15,15       SET RETURN CODE OF ZERO
      PR               RETURN TO CALLER, RESTORE CALLERS STATUS
```

SUSPEND — Suspend Execution of a Request Block

The SUSPEND macro places a request block (RB) in a suspended state until an expected event occurs, causing the task to resume processing.

The SUSPEND macro is written as follows:

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

RB=PREVIOUS RB=CURRENT	**Default:** PREVIOUS

The parameters are explained as follows:

RB=PREVIOUS
RB=CURRENT
specifies which RB on the TCB to suspend. The previous RB is the caller's RB. The current RB is the first RB on the TCB chain.

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	Address of the suspended TCB
1	Address of the suspended RB
2 - 10	Unchanged
11 - 15	Used as work registers by the macro

Example

Operation: Suspend the execution of the most recently chained request block of the current task.

```
SUSPEND  RB=CURRENT
```

SVCUPDTE — SVC Update

The SVCUPDTE macro provides a means to dynamically replace or delete SVC table entries. Callers who use this service are responsible for providing recovery. Improper deletion or replacement of system provided SVC routines causes unpredictable results and might terminate the system.

The resource name, SYSZSVC TABLE, is available as the operand of an ENQ or DEQ macro, to be used when you must serialize the execution of a program that uses the SVCUPDTE macro.

The caller may be in either 24 or 31-bit addressing mode.

Users of this macro must:

- Be in supervisor state and key 0,
- Ensure that register 13 contains the address of a 72-byte save area,
- Ensure that the code for the SVC routine added to the SVC table has the correct attributes for the type of SVC specified, and
- Include the CVT mapping macro.

See *SPL: Application Development Guide* for additional information about the SVCUPDTE macro.

The SVCUPDTE macro is written as follows:

---

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

---

num	*num:* symbol, decimal number, hexadecimal number (for example, X'02'), or register (2) - (12). Do not specify *num* with extract. **Note:** *num* cannot be 109, 116, 122 or 137, which are for extended SVCs.
,REPLACE ,DELETE ,EXTRACT	
,TYPE=1 ,TYPE=2 ,TYPE=3 ,TYPE=4 ,TYPE=5 ,TYPE=6	**Note:** This parameter is not valid with DELETE or EXTRACT.
,EP=*addr*	*addr:* A-type address, decimal number, hexadecimal number, or register (2) - (12). *addr* should be a full 31-bit value with AMODE in bit 0.
,EPNAME=*entry-name*	*entry-name:* symbol **Note:** EP and EPNAME are not valid with TYPE=5 and are not needed with the DELETE option.
,LOCKS=*(lname, lname,...)*	*lname:* CMS or LOCAL. **Note:** LOCKS is invalid with DELETE and EXTRACT, and cannot be specified with TYPE=6.

,APF=YES	**Default:** APF=NO
,APF=NO	**Note:** APF is not valid with DELETE.
,AR=YES	**Note:** AR is valid only with REPLACE.
,AR=NO	**Default:** AR=NO.
,NPRMPT=YES	**Default:** NPRMPT=NO
,NPRMPT=NO	**Note:** NPRMPT is not valid with DELETE.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

num
specifies the number of the SVC that is being inserted or deleted.

,REPLACE
,DELETE
specifies the function to be performed. REPLACE indicates that a SVC table entry is to be inserted in the SVC table. This could be a new SVC or a replacement for an existing SVC. DELETE indicates that the specified SVC number is to be deleted from the SVC table. The SVCUPDTE routine deletes the number by placing the address of the SVC error routine into the table entry. When you execute an SVC instruction with a deleted SVC number, the result is an abnormal termination with an X'Fxx' abend. (xx is the hexadecimal representation of the number specified.) However, if you issue an SVCUPDTE macro with a deleted SVC number, no abend results.

,TYPE=1
,TYPE=2
,TYPE=3
,TYPE=4
,TYPE=5
,TYPE=6
specifies the SVC type for a REPLACE request. See the topic "Programming Conventions for SVC Routines" in *SPL: Application Development Guide* for information concerning the characteristics and restrictions for each type of SVC.

,EXTRACT
indicates that the user has supplied an EP or EPNAME and wishes to have the SVC number of that routine returned in register 0. The **num** parameter is not valid with this option.

,EP=*addr*
specifies the entry point address of the SVC routine. The addressing mode of the entry point is defined by bit 0 of the entry point address of the SVC routine. If bit 0=1, the SVC routine will be entered in 31-bit addressing mode; if bit 0=0, the SVC routine will be entered in 24-bit addressing mode.

,EPNAME=*entry-name*
specifies the entry name of the SVC routine. The entry name must be the load module name or alias of a module in LPA or the entry name of a module link edited into the nucleus. The AMODE of the SVC routine is determined when the SVC routine is link edited.

Note: The service routine must obtain a 72-byte work area to support this option.

,LOCKS=(*lname,lname,...***)**
specifies the lock(s) required when the SVC routine executes. The lock(s) specified can be CMS or LOCAL.

Notes:

1. TYPE=6 cannot specify any locks.
2. TYPE=1 must not specify LOCAL.

,APF = YES
,APF = NO
specifies whether or not the SVC is to be APF-authorized.

,AR = YES
,AR = NO
specifies whether or not the SVC can be issued by a program in access register mode. If you specify NO, a program that issues the SVC while in access register mode abends with a completion code of X'0F8'. This parameter is valid only with REPLACE.

,NPRMPT = YES
,NPRMPT = NO
indicates whether or not the SVC can be preempted for I/O interruptions.

,RELATED = *value*
provides information to document the macro by relating the function performed to another service or function. The format can be any valid coding value that the user chooses.

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

Hexadecimal Code	Meaning
0	The macro completed successfully.
4	The macro was coded incorrectly. For example, the user requested REPLACE without specifying an SVC number.
8	The DELETE parameter was not specified correctly.
0C	A REPLACE request contained incorrect information. For example, the user specified an SVC type that was not 1 through 6.
10	A REPLACE request contained illogical information. For example: • A type 5 SVC specified an entry point. • A type 6 SVC specified a lock. • Neither an entry point nor an EPNAME was provided for a REPLACE request that is not a type 5. • Both an entry point and an EPNAME are provided. • The entry point provided is zero. • The CMS lock was requested without the LOCAL lock.
14	The function specified was not REPLACE, DELETE, or EXTRACT.
18	The user has attempted to update an extended SVC router entry in the SVC table (*num* was specified as 109, 116, 122, or 137).
1C	Unable to locate the entry point address for an EPNAME specification.
20	An EXTRACT request contains illogical information. For example: • Neither an entry point address nor an EPNAME is specified. • Both an entry point address and an EPNAME are specified. • An SVC number is specified. • The entry point address specified is zero.
24	Unable to locate the SVC routine for the EXTRACT request.
28	An error occurred while updating the SVC table.

Example 1
Operation: Delete SVC 200 from the SVC table.

```
SVCUPDTE 200,DELETE
```

Example 2
Operation: Insert SVC 201 in the SVC table. This is a type 2 SVC, with entry point at location SVCADDR. The SVC cannot be preempted for I/O interruptions.

```
SVCUPDTE 201,REPLACE,NPRMPT=NO,TYPE=2,EP=SVCADDR
```

Example 3
Operation: Replace SVC 202 in the SVC table. This is a type 1 SVC with entry point at the location in register 2.

```
SVCUPDTE 202,REPLACE,TYPE=1,EP=(2)
```

Example 4
Operation: Replace SVC 203 in the SVC table. SVC 203 is a type 4 SVC requiring the LOCAL lock. The routine has been loaded into LPA with the name MYSVC.

```
SVCUPDTE 203,REPLACE,TYPE=4,LOCKS=LOCAL,EPNAME=MYSVC
```

Example 5
Operation: Determine the SVC number associated with the name IGC062. The SVC number is to be returned in register 0.

```
SVCUPDTE ,EXTRACT,EPNAME=IGC062
```

Example 6
Operation: Replace SVC 202 in the SVC table. This is a type 3 SVC with entry point at explicit location X'FFEC00'. Note that this example uses a symbol as the SVC number.

```
SVCUPDTE  SVCNUM,REPLACE,TYPE=3,EP=X'FFEC00'
          .
          .
          .
SVCNUM    EQU   202
```

SVCUPDTE (List Form)

The list form of the SVCUPDTE macro builds a non-executable parameter list that can be referred to by the execute form of the SVCUPDTE macro.

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

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

num	*num:* symbol, decimal number, hexadecimal number (for example X'02'). **Note:** This parameter must be specified on the execute and the list form of the macro. Do not specify **num** with EXTRACT.
,REPLACE ,DELETE ,EXTRACT	
,TYPE=1 ,TYPE=2 ,TYPE=3 ,TYPE=4 ,TYPE=5 ,TYPE=6	**Note:** This parameter is not valid with DELETE.
,EP=*addr*	*addr:* A-type address, decimal number, or hexadecimal number.
,EPNAME=*entry-name*	*entry-name:*symbol **Note:** EP and EPNAME are not valid with TYPE=5 and are not needed with the DELETE option. This parameter must be supplied on either the execute or the list form.
,AR=YES ,AR=NO	**Note:** AR is valid only with REPLACE. **Default:** AR=NO.
,LOCKS=*(lname, lname,...)*	*lname:* CMS or LOCAL. **Note:** This option is not valid with DELETE or EXTRACT and must not be specified with TYPE=6.
,NPRMPT=YES ,NPRMPT=NO	**Default:** NPRMPT=NO **Note:** NPRMPT is not valid with DELETE.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

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

,MF=L
specifies the list form of the SVCUPDTE macro.

Example 1

Operation: Use the list form of the macro to replace SVC 202 in the SVC table. It is a type 2 SVC with entry point at location SVCADDR. The SVC routine needs the local lock.

```
SVCUPDTE 202,REPLACE,TYPE=2,LOCKS=LOCAL,MF=L,EP=SVCADDR
```

Example 2

Operation: Use the list form of the macro to replace SVC 201 in the SVC table. The routine is a type 2 SVC.

```
SVCUPDTE 201,REPLACE,TYPE=2,MF=L
```

SVCUPDTE (Execute Form)

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

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

num	register (2) - (12). **Note:** This parameter must be supplied on either the execute or the list form of the macro with REPLACE or DELETE, and it must not be specified with EXTRACT.
,EP=*addr*	*addr:* register (2) - (12). **Note:** This parameter is not valid with TYPE=5 and must be supplied on either the execute or the list form of the macro. This parameter is not needed with the delete option.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E,*addr*)	*addr:* RX-type address or register (2) - (12).

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

,MF=(E,*addr*)
specifies the execute form of the SVCUPDTE macro.

Example

Operation: Use the execute form of the SVCUPDTE macro to perform the function specified by the remote control parameter list whose address is given in register 2.

```
SVCUPDTE MF=(E,(2))
```

SWAREQ — Invoke SWA Manager in Locate Mode

The SWAREQ macro has no standard form. It only has a list, an execute, and a modify form. The MF parameter, which indicates the form of the macro, is required.

When you invoke this macro in execute form, it uses the two parameters, FCODE and EPA, to modify the parameter list, which is at the location you specify by the *addr* value in the **MF=(E,***addr***)** parameter. After ensuring the validity of the parameters, it invokes the SWA manager in locate mode. The SWA manager obtains its input from the parameter list, and performs the function associated with the specified FCODE. If you do not specify any parameters, the macro assumes the parameter list already exists, and it simply invokes the SWA manager. Register 13 must contain the address of a standard 18-word save area.

The modify form of SWAREQ is functionally the same as the execute form, except that the macro only modifies the parameter list without invoking the SWA manager. The list form of SWAREQ generates the parameter list that is modified by the other two forms of the macro, and it does not invoke the SWA manager.

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

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

,FCODE=*fncde*	*fncde:* function code
,EPA=*addr*	*addr:* address of the pointer to the EPA. In the list form, this address may only be specified symbolically.
,MF=L	

The parameters are explained as follows:

,FCODE=*fncde*
specifies the function code for the locate mode request. Valid codes are:

RL Read/Locate
WL Write/Locate

For more information about the meaning of each code, see *SPL: Application Development Guide*.

,EPA=*addr*
specifies the address of the pointer to the extended parameter area (EPA).

,MF=L
specifies the list form of the SWAREQ macro.

SWAREQ (Execute Form)

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

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

,FCODE=*fncde*	*fncde:* function code
,EPA=*addr*	*addr:* external parameter area pointer address. It may be specified symbolically, as a register enclosed in parentheses, or as a symbol equated to a register enclosed in parentheses.
,UNAUTH=YES ,UNAUTH=NO	**Default:** UNAUTH=NO.
,MF=(E,*addr)*	*addr:* RX-type address or register (1) - (12).

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

,UNAUTH=YES
,UNAUTH=NO
specifies that the system is to invoke the unauthorized form of the SWA manager. The unauthorized form of the SWA manager provides the output of the RL function of the authorized SWA manager. If you also specify the FCODE parameter, the SWAREQ macro checks the syntax of the FCODE parameter but does not use the function code.

To use SWAREQ, you must be authorized, holding no locks, in task mode, and not in cross memory mode. However, when you are using SWAREQ to perform a Read Locate, you can override these restrictions by specifying UNAUTH=YES.

You must also issue the macro IEFZB505 LOCEPAX=YES.

,MF=(E,*addr*)
E specifies the execute form of the SWAREQ macro, and **addr** specifies the address of the parameter list.

SWAREQ (Modify Form)

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

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

,FCODE=*fncde*	*fncde:* function code
,EPA=*addr*	*addr:* external parameter area pointer address. It may be specified symbolically, as a register enclosed in parentheses, or as a symbol equated to a register enclosed in parentheses.
,MF=(M,*addr*)	*addr:* RX-type address or register (1) - (12) .

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

,MF=(M,*addr*)
M specifies the modify form of the SWAREQ macro, and *addr* specifies the address of the parameter list.

SYMREC — Process Symptom Record

The SYMREC macro updates the symptom record with system environment information and then logs the symptom record in the SYS1.LOGREC data set. The symptom record is a data area in the user's application that has been mapped by the ADSR macro and that is referenced by a parameter of the SYMREC macro. The data in the symptom record is a description of a programming failure and a description of the environment in which the failure occurred. As the application detects errors during execution, it stores diagnostic information into the symptom record and issues SYMREC to log the information.

The caller can be enabled or disabled for interrupts. If disabled, the input data to SYMREC must be in fixed storage or in disabled reference (DREF) storage. The caller must be in primary ASC mode and can hold any locks.

While the SYMREC macro can be issued in 24-bit or 31-bit addressing mode, the addresses passed to the SYMREC service must be 31-bit addresses. Register 13 must contain the address of a standard 18-word save area.

When SYMREC is invoked, it checks that all the required input fields of the ADSR symptom record are set by the caller. If the required input fields are not set, SYMREC issues appropriate return and reason codes (described in *SPL: Application Development Guide*).

The SYMREC macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SYMREC.
SYMREC	
ƀ	One or more blanks must follow SYMREC.
SR=*addr*	*addr:* A-type address or register (2) - (12).

The parameters are explained as follows:

SR = *addr*
specifies the address of the symptom record. The SR keyword is required.

When SYMREC returns control, registers 15 and 0 contain the following hexadecimal return codes and reason codes, respectively:

Return Code	Reason Code	Explanation
0000		Symptom record component completed successfully and the symptom record was recorded.
	0000	Successful completion of the SYMREC macro service routine.
0004		One or more errors detected on the SYMREC macro statement. The entire input record was recorded. Following are specific reasons why the symptom record component processed unsuccessfully:
	0164	The input symptom record was successfully copied. However, an attempt to write section 1 information from the completed symptom record failed. The area was found non-accessible to a write request.

Return Code	Reason Code	Explanation
0008		One or more errors detected on the SYMREC macro statement. A partial symptom record was recorded. Following are specific reasons why the symptom record component processed unsuccessfully:
	0158	Total length of the input symptom record exceeds the maximum.
	015C	Optional segments of the input symptom record were found non-accessible. The record includes the accessible entries of the input symptom record.
000C		Serious error on the SYMREC macro statement. No symptom record was recorded. Following are specific reasons why the symptom record component processed unsuccessfully:
	0104	The first 2 bytes of the input symptom record do not contain the SR operand.
	0108	The input symptom record does not contain the required entries for section 2.
	010C	The input symptom record does not contain the required entries for section 2.1.
	0114	The input symptom record does not contain the required entries for section 3.
	0128	Portions of the input symptom record were found non-accessible to a write request.
	012C	Required portions of the input symptom record were found non-accessible to a write request.
	0134	Input symptom record address is in non-accessible storage.
	0144	Program attributes of the job issuing the SYMREC macro are not written in accordance with the symptom record component standards.
0010		Serious error in the symptom record component. Error is not related to SYMREC macro statement. No symptom record was recorded. Following are specific reasons why the symptom record component processed unsuccessfully:
	0F04	Insufficient space in the LOGREC buffer to accommodate the symptom record.
	0F08	SYMREC macro service routine could not acquire storage for its workarea and a copy of the symptom record.
	0F0C	Failure occurred while moving the symptom record to the LOGREC buffer.
	0F10	SYMREC macro service routine has a logic error.
	0F14	SYMREC macro service routine has shut itself down. It has exceeded the maximum allowable logic errors for the service routine.
	0F18	SYMREC macro service routine has shut itself down. It has exceeded the maximum allowable incomplete SYMREC requests for processing.
0014		Symptom record component is not operable.

SYMREC (List Form)

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

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SYMREC.
SYMREC	
ƀ	One or more blanks must follow SYMREC.
SR=*addr*	*addr:* A-type address (31 bit).
,MF=(L)	

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

,MF=L
specifies the list form of the SYMREC macro.

SYMREC (Execute Form)

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

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

SR=*addr*	*addr:* A-type address (31 bit) or register (2) - (12).
,MF=(E,*list addr*)	*list addr:* RX-type address or register (2) - (12).

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

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

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

The SYNCH macro takes a synchronous exit to a processing program. After the processing program has been executed, the program that issued the SYNCH macro regains control. The SYNCH macro is also described in *Application Development Macro Reference* with the exception of the KEYADDR, STATE, KEYMASK, and XMENV parameters. These parameters are restricted to programs in supervisor state, key 0-7, or APF-authorized.

If you are executing in 31-bit addressing mode, you must use the MVS/SP Version 2 of this macro, or a later version.

The SYNCH macro is intended for use by primary mode programs only. If your program runs in access register (AR) mode, use SYNCHX, which provides the same function as SYNCH. Descriptions of SYNCH and SYNCHX in this book are:

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

If the caller is in AR mode, the system passes the following values, unchanged, to the processing program:

- ARs 2 - 13
- Bits 16 and 17 of the current PSW indicating the ASC mode (primary or AR mode, where primary = secondary = home)
- Extended authorization index (EAX)

Parameters for SYNCH and SYNCHX must be in the caller's primary address space. Callers in AR mode must qualify the parameter list address with an ALET of 0.

On entry to the processing program, the high-order bit, bit 0, of register 14 is set to indicate the addressing mode of the issuer of the SYNCH macro. If bit 0 is 0, the issuer is executing in 24-bit addressing mode; if bit 0 is 1, the issuer is executing in 31-bit addressing mode.

The SYNCH macro is written as follows:

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

entry point addr	*entry point addr:* RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,KEYADDR=*addr* ,KEYADDR=NOKEYADDR	*addr:* RX-type address, or register (2) - (12) **Default:** KEYADDR=NOKEYADDR (The key in the TCB is used.)
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB

,KEYMASK=*addr*	*addr:* RX-type address, or register (0) - (12).
,XMENV=*addr*	*addr:* RX-type address or register (0) - (12)
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER **Note:** AMODE=DEFINED can only be specified if the entry point is provided in a register.

The parameters are explained as follows:

entry point addr
specifies the address of the entry point of the processing program to receive control.

,RESTORE=NO
,RESTORE=YES
specifies whether registers 2-13 are to be restored when control is returned to the issuer of SYNCH.

,KEYADDR=*addr*
,KEYADDR=NOKEYADDR
addr specifies the address of a one-byte area that contains the key in which the exit is to receive control. The key must be in bits 0-3; bits 4-7 must be zero. If KEYADDR=*addr* is not specified, the key in the TCB is used as the default.

,STATE=PROB
,STATE=SUPV
specifies the state in which the requested program receives control. PROB specifies problem state and SUPV specifies supervisor state.

,KEYMASK=*addr*
specifies the address of a halfword, which along with the protect key of the currently active TCB, will be an operand in an OR instruction. The results of that instruction produce the PKM of the routine to which your program will take a synchronous exit.

If you specify KEYMASK, do not specify XMENV.

,XMENV=*addr*
specifies the address of a parameter list that the caller passes to the SYNCH macro service. The parameter list contains values that set up a cross memory environment for the new PRB. The parameter list consists of a 10-byte list of values that determine some of the characteristics the PRB will have when it receives control. The parameter list must reside in the primary address space and the AR that qualifies the address must be set be set to 0. The format of the parameter list is as follows:

Bytes	Content of field
0 - 1	The value X'0A'
2 - 3	PKM value, which along with the protect key of the currently active TCB, will be an operand in an OR instruction. The results of that instruction produce the PKM of the routine to which the synchronous is to be taken.
4 - 5	SASN - defining the secondary address space for the exit routine
6 - 7	Extended authorization index (EAX) for the exit routine
8 - 9	PASN - defining the primary address space for the exit routine

If you specify XMENV, do not specify KEYMASK.

,AMODE=24
,AMODE=31
,AMODE=DEFINED
,AMODE=CALLER

specifies the addressing mode in which the requested program is to receive control.

If AMODE=24 is specified, the requested program will receive control in 24-bit addressing mode.

If AMODE=31 is specified, the requested program will receive control in 31-bit addressing mode.

If AMODE=DEFINED is specified, the user must provide the entry point using a register, not an RX-type address. The requested program will receive control in the addressing mode indicated by the high-order bit of the entry point address. If the bit is off, the requested program will receive control in 24-bit addressing mode; if the bit is set, the requested program will receive control in 31-bit addressing mode.

If AMODE=CALLER is specified, the requested program will receive control in the addressing mode of the caller.

Example 1

Operation: Take a synchronous exit to a processing program whose entry point address is specified in register 8.

```
SYNCH (8)
```

Example 2

Operation: Take a synchronous exit to a processing program labeled SUBRTN and restore registers 2-13 when control is returned.

```
SYNCH SUBRTN,RESTORE=YES
```

Example 3

Operation: Take a synchronous exit to a processing program whose entry point address is specified in register 5, modify the program's protect key by the KEYADDR and KEYMASK values, and restore registers 2-13 when control returns.

```
SYNCH (5),RESTORE=YES,KEYADDR=KEYBYTE,KEYMASK=MSKADDR
   .
   .
   .
KEYBYTE DC X'80'
MSKADDR DC X'0080'
```

Example 4

Operation: Take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that this program is to execute in 24-bit addressing mode.

```
SYNCH (8),RESTORE=YES,AMODE=24
```

SYNCHX — Take a Synchronous Exit to a Processing Program

The SYNCHX macro allows a program running in primary or AR mode to take a synchronous exit to a processing program. This macro is the same as the SYNCH macro, except that, for callers in AR mode, it generates code and addresses that are appropriate in AR mode. All parameters on the SYNCH macro are valid for the SYNCHX macro.

Before you issue the SYNCHX macro, issue the SYSSTATE ASCENV=AR macro to tell the SYNCHX macro to generate code appropriate for AR mode.

The SYNCHX macro is written as follows:

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

entry point addr	*entry point addr:* RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,KEYADDR=*addr* ,KEYADDR=NOKEYADDR	*addr:* RX-type address, or register (2) - (12) **Default:** KEYADDR=NOKEYADDR (The key in the TCB is used.)
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB
,KEYMASK=*addr* ,XMENV=addr	*addr:* RX-type address, or register (0) - (12). **addr:** RX-type address or register (0) - (12)
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER **Note:** AMODE=DEFINED can only be specified if the entry point is provided in a register.

The parameters are described under the syntax of the standard form of the SYNCH macro.

SYNCH and SYNCHX (List Form)

The list form of the SYNCH or SYNCHX macro is used to construct a control program parameter list.

The list form of the SYNCH or SYNCHX macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SYNCH or SYNCHX.
SYNCH or SYNCHX	
b	One or more blanks must follow SYNCH or SYNCHX.
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB
,KEYMASK=*addr* ,XMENV=*addr*	*addr:* A-type address. *addr:* RX-type address or register (0) - (12)
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER
,MF=L	

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

,MF=L
specifies the list form of the SYNCH macros.

Example

Operation: Use the list form of the SYNCH macro to specify that registers 2-13 are to be restored when control returns from executing the SYNCH macro and that the addressing mode of the program is to be defined by the high-order bit of the entry point address. Assume that the execute form of the macro specifies the program address.

```
SYNCH ,RESTORE=YES,AMODE=DEFINED,MF=L
```

SYNCH and SYNCHX (Execute Form)

The execute form of the SYNCH or SYNCHX macro uses a remote program parameter list that can be generated by the list form of SYNCH or SYNCHX.

The execute form of the macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SYNCH or SYNCHX.
SYNCH	
b	One or more blanks must follow SYNCH or SYNCHX.
entry point addr	*entry point addr:* RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,KEYADDR=*addr* ,KEYADDR=NOKEYADDR	*addr:* RX-type address, or register (2) - (12).
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB
,KEYMASK=*addr* ,XMENV=*addr*	*addr:* RX-type address, or register (0) - (12). *addr:* RX-type address or register (0) - (12)
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER **Note:** AMODE=DEFINED can only be specified if the entry point is provided in a register.
,MF=(E,*ctrl addr)*	*ctrl addr:* RX-type address or register (1), (2) - (12)

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

,KEYADDR=NOKEYADDR
indicates that the default(the key in the TCB) should be used instead of the key in the parameter list defined by a list form of the macro.

,MF=(E,*ctrl addr)*
specifies the execute form of the SYNCH macro using a list generated by the list form of SYNCH.

Example

Operation: Use the execute form of the SYNCH macro to take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that the program is to receive control in the same addressing mode as the caller and that the parameter list is located at SYNCHL2.

```
SYNCH (8),RESTORE=YES,AMODE=CALLER,MF=(E,SYNCHL2)
```

SYSEVENT — System Event

The SYSEVENT macro provides the interface to the system resource manager (SRM). By using SYSEVENT mnemonics, you can notify SRM of an event or ask SRM to perform a specific function.

Include the CVT mapping macro as a DSECT in the calling program. If a specific SYSEVENT requires additional parameters, load register 1 with the address of a parameter list before issuing the macro.

Callers who use ENTRY=BRANCH must:

- Be in supervisor state, PSW key 0
- Hold the LOCAL lock for TRAXERPT, TRAXFRPT, and TRAXRPT. (There are no locking requirements for the other SYSEVENTs.)
- Provide the address of a serialized 72-byte save area in register 13

Callers who use ENTRY=SVC with STGTEST can be problem state with any PSW key. All other uses of SYSEVENT with ENTRY=SVC require that the caller be APF authorized, supervisor state, or PSW key 0.

Additional restrictions concerning the use of each SYSEVENT, including input and output requirements, follow the description of the parameters.

The SYSEVENT macro is written as follows:

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

sysevent mnemonic	*sysevent mnemonic:* symbol. **Note:** See the description of the parameters for the valid options.
,ENTRY=SVC ,ENTRY=BRANCH	**Defaults:** ENTRY=BRANCH for the following SYSEVENTs: TRAXERPT TRAXFRPT TRAXRPT ENTRY=SVC for the following SYSEVENTs: DONTSWAP OKSWAP TRANSWAP STGTEST
,TYPE=BLOCK ,TYPE=BYTE	**Note:** TYPE=BLOCK and TYPE=BYTE are valid only for SYSEVENT STGTEST. **Default:** TYPE=BLOCK

The parameters are explained as follows:

sysevent mnemonic
identifies the SYSEVENT being requested.

,ENTRY=SVC
,ENTRY=BRANCH
specifies the type of interface to SRM (SVC or branch).

Only users who do not hold a lock can specify ENTRY=SVC. This is the default for DONTSWAP, OKSWAP, TRANSWAP, and STGTEST.

ENTRY=BRANCH is required if the caller holds a lock and for all "fast path" SYSEVENTs. The "fast path" SYSEVENTs include TRAXERPT, TRAXFRPT, and TRAXRPT. ENTRY=BRANCH is the default for these "fast path" SYSEVENTs. For branch entry, callers must provide a 72-byte save area and place the address of the save area in register 13.

TYPE=BYTE
TYPE=BLOCK
indicates whether SYSEVENT STGTEST is to return values that reflect either available central and expanded storage, **or** available expanded storage. TYPE=BYTE requests central and expanded storage; TYPE=BLOCK requests expanded storage.

SYSEVENT mnemonics

A description of the SYSEVENTs available for use follows. These mnemonics are grouped according to the basic function that they perform.

Notify SRM of Transaction Completion

The SYSEVENTs TRAXRPT, TRAXFRPT, and TRAXERPT notify SRM that a subsystem transaction has completed and provide the transaction's starting time or elapsed time and, optionally, its resource utilization. This performance data can be reported using the resource management facility (RMF).

To obtain reports, an IEAICSxx member of parmlib must be in effect and RMF workload activity reporting must be active. See *SPL: Initialization and Tuning* for additional information concerning IEAICSxx.

In addition to the general requirements for SYSEVENTs, TRAXRPT, TRAXFRPT, and TRAXERPT require the user to:

- Provide a parameter list
- If the issuing program is disabled, ensure that the parameter list and save area are fixed
- Provide error recovery

A description of the individual mnemonics follows:

TRAXRPT
notifies SRM that a transaction has completed and provides its start time. Register 1 must point to a serialized parameter list with the following format:

Offset in Hex	Length	Field Description
00	8	Transaction start time in store clock instruction (STCK) format
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification or blanks
20	8	Transaction class or blanks

Note: You can obtain this parameter list by using the IHATRBPL mapping macro in your program.

The names must be in ECBDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length even though the

parameter list provides an eight-character field. Use the first four characters of the field for the sybsystem name.

TRAXFRPT

notifies SRM that a transaction has completed and provides the elapsed time. Because the issuer calculates the elapsed time before issuing the macro, this path is shorter than the path for TRAXRPT. Register 1 must point to a serialized parameter list with the following format:

Offset in Hex	Length	Field Description
00	4	Transaction elapsed time (1.024 milliseconds units)
04	4	Reserved - must be zero
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification or blanks
20	8	Transaction class or blanks

Note: You can obtain this parameter list by including the mapping macro IHATRBPL in your program. The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

TRAXERPT

notifies SRM that a transaction has completed, provides its start time, and includes resource utilization data for determining service consumption. Register 1 must point to a serialized parameter list in the following format:

Offset in Hex	Length	Field Description
00	8	Transaction start time in STCK format
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification or blanks
20	8	Transaction class or blanks
28	8	Task (TCB) time in STCK format or zeros
30	8	SRB time in STCK format or zeros
38	8	Main storage occupancy in page seconds (pages times msec, where msec is task (TCB) time in 1.024 millisecond units)
40	4	Logical I/O count or zeros
44	1	X'00' if the previous field contains the logical I/O count X'80' if the previous field contains the device connect time interval (DCTI)
45	3	Reserved must be zero

Note: You can obtain this parameter list by including the mapping macro IHATREPL in your program.

The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

When SYSEVENT processing is completed, the subsystem regains control at the instruction following the SYSEVENT macro. Register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Data for the transaction has been reported successfully to the SRM.
08	Processing could not be completed at this time. No queue elements are available for recording the data. No data is reported to the SRM, but an immediate reissue could be successful.
0C	Reporting is temporarily suspended for one of the following reasons: • RMF workload activity reporting is not active. • There is no installation control specification (IEAICSxx parmlib member with RPGN specified for some subsystem other than TSO) in effect. The TOD clock is stopped. No data is reported, but a later reissue could be successful.
10	Reporting is inoperative. The TOD clock is in error or the reporting interface is not installed. No data is reported.

Example 1

Operation: Use the SYSEVENT TRAXRPT to report transaction data providing transaction identifiers and the transaction start time.

```
 .
 .
 .
Transaction begins      Initialize transaction identifiers
      (TRAXDES)
 STCK INITTIME          Save start time
 .
Process transaction
Transaction completes
 LA R13,SVAREA          Provide 72-byte save area
 LA R1,PARMS            Point to parameter area
 MVC 0(8,R1),INITTIME   Move in start time
 MVC 8(32,R1),TRAXDESC  Get subsystem name, transaction
                        name, userid, and class
 SYSEVENT TRAXRPT
 .
 .
INITTIME DS D
PARMS    DS 5D
SVAREA   DS 18F
TRAXDESC DS CL32
```

Example 2

Operation: Use the SYSEVENT TRAXERPT to report transaction data, providing transaction identifiers, start time and resource utilization data.

```
 .
 .
Transaction begins      Initialize transaction identifiers
      (TRAXDESC)
 STCK INITTIME          Save start time
 .
Process transaction     Accumulate resource utilization data
      (TRAXDESC)
 .
Transaction completes
 LA  R13,SVAREA         Provide 72-byte save area
 LA  R1,PARMS           Point to parameter area
 MVC 0(8,R1),INITTIME   Move in start time
 MVC 8(64,R1),TRAXDESC  Get subsystem name, transaction
                        name, user id, class, and
                        resource utilization data
 SYSEVENT TRAXERPT
 .
 .
 .
INITTIME DS D
PARMS    DS 9D
SVAREA   DS 18F
TRAXDESC DS CL64
```

Example 3

Operation: Use the SYSEVENT TRAXFRPT to report transaction data, providing transaction identifiers and calculating the elapsed time.

```
 .
 .
Transaction begins      Initialize transaction identifiers (TRAXDESC)
 .
 .
Process transaction     Calculate elapsed time (TOTLTIME)
 .
Transaction completes   Calculate elapsed time (TOTLTIME)
 LA  R13,SVAREA         Provide 72-byte save area
 LA  R1,PARMS           Point to parameter area
 MVC 0(4,R1),TOTLTIME   Move in elapsed time
 XC  4(4,R1),4(R1)      Clear reserved field
 MVC 8(32,R1),TRAXDESC  Get subsystem name, transaction name,
                        user id, and class
 SYSEVENT TRAXFRPT
 .
 .
 .
TOTLTIME DS F
PARMS    DS 5D
SVAREA   DS 18F
TRAXDESC DS CL32
```

Control Swapping

The SYSEVENTs DONTSWAP, OKSWAP, and TRANSWAP control swapping. The choice of mnemonic depends on the period of time for which the address space is to be non-swappable.

For a short period of time (less than one minute), use DONTSWAP to make it non-swappable and OKSWAP to make it swappable.

For an extended period of time (more than one minute), use TRANSWAP to make the address space non-swappable and OKSWAP to make it swappable.

A description of the individual mnemonics follows:

DONTSWAP

notifies SRM that the address space from which this SYSEVENT is issued cannot be swapped out until the system receives a matching OKSWAP for each DONTSWAP issued or until the jobstep ends.

No input parameters are required. One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The request was not honored because it was not for the current address space.
08	The request was not honored because the issuer was not authorized or the outstanding count of DONTSWAP requests had reached its maximum.

OKSWAP

notifies SRM that the address space from which the SYSEVENT was issued can be considered for swapping.

No input parameters are required. One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The request was not honored because it was not for the current address space.
08	The request was not honored because the issuer was not authorized.

TRANSWAP

forces a swap out. After the subsequent swap-in, frames are allocated from preferred storage and the address space is non-swappable. TRANSWAP prevents programs from allocating frames in reconfigurable storage. If the program issuing SYSEVENT depends on the transition to complete, you should ensure that register 1 contains the address of an ECB. SYSEVENT will then post this ECB when it swaps out the address space. If no dependency exists, set register 1 to 0 (zero).

One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored. If an ECB was specified, your program should issue a WAIT macro specifying the same ECB.
04	The transition was previously done or the address space is permanently non-swappable.

If an ECB was specified, the following POST codes may occur in the last three bytes of the ECB:

Hexadecimal Code	Meaning
000000	The transition is complete.
000004	The address space became non-swappable before it could be swapped out.

Example 1

Operation: Make the current address space non-swappable for a time period of less than one minute.

```
SYSEVENT DONTSWAP
         .
         .
         .
SYSEVENT OKSWAP
```

Example 2

Operation: Make the current address space non-swappable for an indefinite period of time.

```
         LA       R1,ECBWORD      Supplies an ECB
         SYSEVENT TRANSWAP
         LTR      R15,R15
         BNZ      FAILED
         WAIT     ECB=ECBWORD
                  .
                  .
FAILED                            Processing from FAILED label
                  .
                  .
```

Obtain System Measurement Information

STGTEST provides information about the current physical use of resources. This is not an indication of how much virtual storage your installation will allow you to obtain. For more information on obtaining virtual storage for hiperspaces or data spaces, see DSPSERV.

The user must supply the address of a storage area large enough to store the requested data.

A description of the individual mnemonics follows:

STGTEST
returns information about the amount of storage available in the system. The purpose of SYSEVENT STGTEST is to help an application decide whether to use an additional virtual storage area, such as a hiperspace. Information is about either central and expanded **or** only expanded storage.

When you use this information, be aware of the dynamic nature of storage. **Output of the SYSEVENT STGTEST represents the current state of storage and does not reserve this storage for the caller or guarantee that it will be available for use.**

TYPE = BYTE
TYPE = BLOCK
specifies whether the system is to provide information about central storage and expanded storage (through TYPE = BYTE), or expanded storage (through TYPE = BLOCK). The default is TYPE = BLOCK.

Register 1 must contain the address of a three-word output area where SRM is to return the information. After SRM returns, each word contains a storage amount that represents a specific number of frames. Before you choose a number to use as the basis for decision, be aware of how your decision affects the performance of the system. General rules are:

- Use of the first number will affect system performance very little, if at all.
- Use of the second number might affect system performance to some degree.
- Use of the third number might substantially affect system performance.

If you base decisions on the value in the second or third word, SRM may have to take processor storage away from other programs and replace it with auxiliary storage.

If the requesting address space does not have storage isolation: SRM calculates the values that it returns to the three words in different ways depending on whether the request is TYPE = BYTE or TYPE = BLOCK, and whether the application's address space has storage isolation. Figure 26 describes the values in the three words when storage isolation is not in effect. Values are returned in units of 4K bytes.

Figure 26. Calculations for SYSEVENT STGTEST with No Storage Isolation

Condition	Values for TYPE = BYTE	Values for TYPE = BLOCK
Word One	Amount of central and expanded storage not in use	Amount of expanded storage not in use
Word Two	The value in word one, plus all expanded and central storage frames that belong to address spaces that have been inactive for more than *n* seconds, where *n* is the value set on ESCTBDS in IEAOPTxx member of parmlib.	The value in word one, plus all expanded storage frames that belong to address spaces that have been inactive for more than *n* seconds, where *n* is the value set on ESCTBDS in IEAOPTxx member of parmlib.
Word Three	The value in word two, plus some auxiliary storage slots not in use[1]	The value in word two, plus some auxiliary storage slots not in use[1]

1 SRM will not recommend an amount that will cause an auxiliary storage shortage. See the section on prevention of storage shortages in *SPL: Initialization and Tuning*.

If the requesting address space has storage isolation: To calculate the values for applications that have storage isolation, SRM first calculates the values for each word as if storage isolation were not in effect. It then modifies the values depending on:

- The number of frames the address space already has
- The minimum and (optionally) the maximum values specified through the PWSS keyword in the IEAIPSxx member of parmlib

Example

An application needs a standard hiperspace. Before it makes the request, the application uses SYSEVENT STGTEST to find out how much expanded storage is available. The values that SRM returns determine how large a hiperspace the application will create.

Operation: Obtain a report on the available expanded storage in the system.

```
        LA  1,ESPARM
        SYSEVENT STGTEST,TYPE=BLOCK
           .
           .
ESPARM DS  3F
```

The application will base its decision on the numbers in the first and second words of the output area.

SYSSTATE — Set and Test Address Space Control (ASC) Mode

Certain macros that support callers in both access register (AR) and primary address space control (ASC) mode need to know which ASC mode your program is running in. The macros that support callers in AR mode generate the code and addresses that are appropriate for AR mode; macros that support callers in primary mode generate the code and addresses that are appropriate for callers in primary mode. These macros use the SYSSTATE TEST macro to test a global symbol that is set through the SYSSTATE ASCENV macro. The name of the global symbol is &SYSASCE.

A good programming practice is to issue the SYSSTATE ASCENV=AR macro at the time your program changes ASC mode to AR mode. Then, when your program returns to primary mode, issue SYSSTATE ASCENV=P.

See Figure 4 on page 6 for a list of the macros that check the setting of the global symbol.

The requirements for the caller are:

Authorization:	Supervisor state or problem state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Not applicable
Amode:	Any
ASC mode:	Any
Serialization:	None

The SYSSTATE macro is written as follows:

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

ASCENV=P ASCENV=AR	**Default**: ASCENV=P **Note**: If you specify TEST, do not specify ASCENV.
TEST	**Note:**If you specify ASCENV, do not specify TEST.

The parameters are explained as follows:

ASCENV=P
ASCENV=AR
specifies the ASC mode in which your program is running.

If your program is in AR mode, use ASCENV=AR. If your program is in primary mode, use ASCENV=P.

TEST
determines the current ASC mode. SYSSTATE TEST checks the global symbol that was set by the the most recent invocation of SYSSTATE ASCENV. Depending on the setting of the global symbol, the caller of SYSSTATE TEST generates code and addresses appropriate for primary mode or AR mode.

Example: To change the ASC mode to AR mode and set the global symbol, issue:

```
SAC       512
SYSSTATE ASCENV=AR
```

TCBTOKEN — Request or Translate the TTOKEN

The TTOKEN is the 16-byte identifier of a task. Unlike a TCB address, each TTOKEN is unique within the IPL; the system does not reassign this same identifier to any other TCB.

The TCBTOKEN macro provides five mutually exclusive services depending on how you specify the TYPE parameter:

- TYPE=TOTTOKEN gives you the TTOKEN for the task associated with a specified TCB address.
- TYPE=TOTCB gives you the TCB address for a specified TTOKEN.
- TYPE=CURRENT gives you the TTOKEN for the current task.
- TYPE=PARENT gives you the TTOKEN for the task that attached the current task.
- TYPE=JOBSTEP gives you the TTOKEN for the job step task.

Typical situations when you would use TYPE=TOTTOKEN are:

- When you create a data space and want to assign ownership of the data space to a second task.

  In this case, you know the TCB address for the second task, but you don't know its TTOKEN (for input to the DSPSERV CREATE macro). Use TYPE=TOTTOKEN to obtain the TTOKEN.

- When you want to delete a data space you do not own.

  In this case, you know the TCB address for the other task, but you don't know its TTOKEN (for input to the DSPSERV DELETE macro). Use TYPE=TOTTOKEN to obtain the TTOKEN.

- When you want to know whether the owner of a data space still exists.

  In this case, you know the TTOKEN for the owning task. If the system returns the TCB address in response to the TYPE=TOTCB parameter, the task still exists.

SPL: Application Development Extended Addressability describes STOKENs and TTOKENs.

Requirements

The requirements for the caller are:

Authorization:	Problem or supervisor state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
Amode:	31-bit
ASC mode:	Primary or AR
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	For TOTTOKEN and TOTCB requests, the caller must hold the local lock or CML lock of the specified address space. For CURRENT, PARENT, and JOBSTEP requests, there is no requirement.
Control parameters:	Control parameters can reside in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Restrictions and Limitations

None

Register Information

When the system returns control to the caller, the contents of some registers are unpredictable. 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

When control returns to the caller, the 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

Programming Requirements
None

Performance Implications
None

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

---

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

TYPE=TOTTOKEN TYPE=TOTCB TYPE=CURRENT TYPE=PARENT TYPE=JOBSTEP	**Note:** See the table following this diagram for information on parameter usage with TYPE.
,TCB=*tcb addr*	*tcb addr:* RX-type address or register (2) - (12).
,TTOKEN=*ttoken addr*	*ttoken addr:* RX-type address.
,ASCB=*ascb addr* ,STOKEN=*stoken addr*	*ascb addr:* RX-type address or register (2) - (12). *stoken addr:* RX-type address. **Default:** Home address space.
,RELATED=*value*	*value:* Any valid macro parameter specification.

The following table shows how the parameters may be specified with the TYPE keywords.

Parameters	TYPE= TOTTOKEN	TYPE= TOTCB	TYPE= CURRENT	TYPE= PARENT	TYPE= JOBSTEP
TCB	required	required	not valid	not valid	not valid
TTOKEN	required	required	required	required	required
ASCB	optional	optional	not valid	not valid	not valid
STOKEN	optional	not valid	not valid	not valid	not valid
RELATED	optional	optional	optional	optional	optional

Parameter Descriptions

The parameters are explained as follows:

TYPE = TOTTOKEN
TYPE = TOTCB
TYPE = CURRENT
TYPE = PARENT
TYPE = JOBSTEP

specifies the type of TCB information requested, as follows:

TOTTOKEN	The system returns the TTOKEN of the task whose TCB address is specified in the TCB parameter. The TTOKEN is returned at the address specified by the TTOKEN parameter.
TOTCB	The system returns the TCB address for the task whose TTOKEN is specified in the TTOKEN parameter. The TCB address is returned at the address specified by the TCB parameter.
CURRENT	The system returns the TTOKEN of the currently active task. The TTOKEN is returned at the address specified by the TTOKEN parameter.
PARENT	The system returns the TTOKEN of the task that attached the currently active task. The TTOKEN is returned at the address specified by the TTOKEN parameter.
JOBSTEP	The system returns the TTOKEN of the job step task for the address space in which the currently active task is running. The TTOKEN is returned at the address specified by the TTOKEN parameter.

,TCB = *tcb addr*

specifies the TCB address. For TYPE = TOTTOKEN, *tcb addr* contains the TCB address that is to be translated to a TTOKEN. For TYPE = TOTCB, *tcb addr* points to a fullword where the system returns the TCB address for the task whose TTOKEN is specified by the TTOKEN parameter.

,TTOKEN = *ttoken addr*

specifies the address of the 16-byte TTOKEN. For TYPE = TOTTOKEN, TYPE = CURRENT, TYPE = PARENT, and TYPE = JOBSTEP, *ttoken addr* is the address at which the TTOKEN associated with the specified TCB is returned. For TYPE = TOTCB, *ttoken addr* is the address of the TTOKEN for the task whose TCB address is to be obtained.

,ASCB = *ascb addr*
,STOKEN = *stoken addr*

identifies the address space of the TCB. ASCB specifies the address of the fullword containing the ASCB address. STOKEN specifies the address of the 8-byte STOKEN that identifies the address space in which the TCB resides. If you do not specify either ASCB or STOKEN, TCBTOKEN uses the home address space by default.

,RELATED = *value*

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

Return Codes

When TCBTOKEN returns control, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	TCBTOKEN services completed successfully.
04	The input STOKEN or TTOKEN does not represent a valid address space.
08	No local lock was held.
0C	A local lock was held, but not the local lock of the associated address space.
10	The TCB could not be referenced.
14	The TCB did not pass the acronym check.
18	The TCB has terminated.
1C	The TCB associated with the TTOKEN represents a different task than when the TTOKEN was obtained.
20	An unexpected error occurred.
24	The contents of access register 1, used to address the parameter list, were not valid.
28	The parameter list is not valid.
2C	The ASCB address is the address of the wait ASCB. The system cannot obtain the TTOKEN.
30	The task is scheduled for termination, but has not yet terminated.
34	The caller is not running in task mode. This return code is valid only for TYPE=CURRENT, TYPE=PARENT, or TYPE=JOBSTEP.

Example 1

Operation: Obtain the TTOKEN for the task whose TCB address is specified in THEIR_TCB. The task resides in the address space whose ASCB address is specified in register 4. Store the returned TTOKEN in THEIR_TOKEN.

```
TCBTOKEN  TYPE=TOTTOKEN,TCB=THEIR_TCB,TTOKEN=THEIR_TTOKEN,ASCB=(4)
```

Example 2

Operation: Obtain the TTOKEN for the currently active task and store it in CURRENT_TTOKEN.

```
TCBTOKEN  TYPE=CURRENT,TTOKEN=CURRENT_TTOKEN
```

Example 3

Operation: Obtain the TCB address of the job step TCB and store it in JOBSTEP_TCB_ADDR.

```
TCBTOKEN  TYPE=JOBSTEP,TTOKEN=JOBSTEP_TTOKEN
TCBTOKEN  TYPE=TOTCB,TTOKEN=JOBSTEP_TTOKEN,TCB=JOBSTEP_TCB_ADDR
```

TCBTOKEN (List Form)

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

Syntax Diagram

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

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

,RELATED=*value*	*value:* Any valid macro parameter specification.
,MF=L	

Parameter Descriptions

The parameters are explained below:

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

,MF=L
specifies the list form of the TCBTOKEN macro.

TCBTOKEN (Execute Form)

Use the execute form of the TCBTOKEN 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 Diagram

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

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

TYPE=TOTTOKEN TYPE=TOTCB TYPE=CURRENT TYPE=PARENT TYPE=JOBSTEP	**Note:** See the table following this diagram for information on parameter usage with TYPE.
,TCB=*tcb addr*	*tcb addr:* RX-type address or register (2) - (12).
,TTOKEN=*ttoken addr*	*ttoken addr:* RX-type address.
,ASCB=*ascb addr* ,STOKEN=*stoken addr*	*ascb addr:* RX-type address or register (2) - (12). *stoken addr:* RX-type address. **Default:** Home address space.
,RELATED=*value*	*value:* Any valid macro parameter specification.
,MF=(E,*cntl-addr*)	*cntl-addr:* RX-type address or register (1) - (12).

The following table shows how the parameters may be specified with the TYPE keywords.

Parameters	TYPE= TOTTOKEN	TYPE= TOTCB	TYPE= CURRENT	TYPE= PARENT	TYPE= JOBSTEP
TCB	required	required	not valid	not valid	not valid
TTOKEN	required	required	required	required	required
ASCB	optional	optional	not valid	not valid	not valid
STOKEN	optional	not valid	not valid	not valid	not valid
RELATED	optional	optional	optional	optional	optional

Parameter Descriptions

The parameters are the same as those for the standard form of the TCBTOKEN macro with the following addition:

,MF=(E,*cntl-addr*)
specifies the execute form of the TCBTOKEN macro. This form uses a remote parameter list. The *cntl-addr* specifies the address of the remote parameter list that the list form of the macro generates.

TCTL — Transfer Control from an SRB Process

The TCTL (transfer control) macro allows an SRB process to exit from its processing and to pass control directly to a task. The caller must be in primary ASC mode.

The TCTL macro is coded as follows:

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

TCB=(4) TCB=*tcbaddr*	**Default:** TCB address contents of register (4) *tcbaddr:* A-type address or registers (2) - (12).

The parameters are explained as follows:

TCB=(4)
TCB=*tcbaddr*
specifies the task designated for dispatching. Register (4) is the default; it is assumed to contain the appropriate TCB address.

Note: The TCB resides in storage below 16 megabytes.

The TCTL macro uses registers as follows:

Register	Use
0-3	Work registers
4	TCB address
5-14	Work registers
15	EPA of the TCTL routine

Example

Operation: From SRB mode processing, terminate the SRB and give control to the task specified in register 4.

```
TCTL TCB=(4)
```

TESTAUTH — Test Authorization of Caller

The TESTAUTH macro is used on behalf of a privileged or sensitive function to verify that its caller is appropriately authorized.

TESTAUTH supports the authorized program facility (APF) - a facility that permits the identification of programs that are authorized to use restricted functions. In addition, TESTAUTH provides the capability for testing for system key 0-7 and supervisor state.

The TESTAUTH macro is written as follows:

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

FCTN=*fctn* FCTN=*fctn*,AUTH=*auth*	*fctn:* decimal digit 0 or 1 or register (2) - (12). *auth:* decimal digit 0 or 1, or register (2) - (12). **Default:** FCTN=0
,STATE=NO ,STATE=YES	**Default:** STATE=NO
,KEY=NO ,KEY=YES	**Default:** KEY=NO
,RBLEVEL=2 ,RBLEVEL=1	**Default:** RBLEVEL=2
,BRANCH=NO ,BRANCH=YES	**Default:** BRANCH=NO

The parameters are explained as follows:

FCTN=*fctn*
FCTN=*fctn* **,AUTH=***auth*
specifies the authorization (via APF) of a program.

FCTN=0 specifies that APF-authorization is not checked.

FCTN=1 specifies that APF-authorization is checked.

AUTH=0 specifies that the job step is not authorized to perform any restricted function.

AUTH=1 specifies that the job step is authorized to perform restricted functions.

Note: If FCTN=1 is specified by itself (that is, without the AUTH parameter), the JSCB is used to check for authorization. AUTH should only be coded when it is not possible for TESTAUTH to acquire the code from the JSCB.

,STATE=NO
,STATE=YES
specifies whether or not (YES or NO) a check is to be made for supervisor/problem program state. (Supervisor state is authorized, problem program state is not authorized.)

,KEY=NO
,KEY=YES
specifies whether or not (YES or NO) a check is to be made of the protection keys. (Protection keys 0-7 are authorized, protection keys 8-15 are not authorized.)

Note: TESTAUTH is used to test one or more of three conditions FCTN,STATE, or KEY. If any of the requested conditions are tested favorably, a return code of 0 is returned in register 15. If all of the requested conditions are tested unfavorably, the return code is set to 4.

,RBLEVEL=2
,RBLEVEL=1
specifies whether the TESTAUTH caller is a type 2, 3, or 4 SVC (RBLEVEL=2), or a type 1 SVC (RBLEVEL=1). If the TESTAUTH caller is not an SVC, specify RBLEVEL=1.

,BRANCH=NO
,BRANCH=YES
specifies a branch entry (YES) or an SVC entry (NO). If BRANCH=YES is specified, registers 2 and 3 are modified by the TESTAUTH routine. Only SVC routines can use BRANCH=YES.

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

Hexadecimal Code	Meaning
00	Task is authorized.
04	Task is not authorized.

Example 1
Operation: Test jobstep for APF authorization.

```
TESTAUTH FCTN=1
```

Example 2
Operation: Test for APF authorization and supervisor state, and do not check protection keys.

```
TESTAUTH STATE=YES,KEY=NO,FCTN=1
```

TIMEUSED — Obtain Accumulated CPU or Vector Time

The TIMEUSED macro returns an eight-byte hexadecimal number in a doubleword storage area that you specify. The number is the total CPU or vector time used by the current TCB or SRB up until you issue the macro. The format of the number is time-of-day (TOD) clock or microseconds time format.

If you use the SRBSTAT save and restore services, the number includes the interval between dispatch and save, and between restore and TIMEUSED. It does not include the interval between save and restore. If you have not yet issued restore, the number includes only the interval between save and TIMEUSED.

The requirements for the caller are:

Authorization:	Supervisor state, PSW key 0 when you specify LINKAGE=BRANCH. Supervisor or problem state, any key when you specify LINKAGE=SYSTEM.
Dispatchable unit mode:	Task or SRB when LINKAGE=BRANCH. Task when LINKAGE=SYSTEM.
Cross memory mode:	Any
Amode:	31-bit addressing mode
ASC mode:	Primary or secondary when LINKAGE=BRANCH. Primary or AR (access register) when LINKAGE=SYSTEM.
Serialization:	Enabled or disabled, unlocked
Control parameters:	N/A

Register 13 must point to a 72-byte save area when you specify LINKAGE=BRANCH.

TIMEUSED is also documented in the *Application Development Macro Reference*, but without the LINKAGE=BRANCH parameter. That parameter is available only to authorized callers.

The TIMEUSED macro is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede TIMEUSED.
TIMEUSED	
b	One or more blanks must follow TIMEUSED.
STORADR=*addr*	*addr*: RX-type address or register (2)-(12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=BRANCH
,RELATED=*value*	*value*: Any valid macro parameter specification
,CPU=TOD ,CPU=MIC	**Default:** CPU=TOD if you do not specify either the CPU or VECTOR keywords.
,VECTOR=TOD ,VECTOR=MIC	

The parameters are explained as follows:

STORADR = *addr*
specifies the 31-bit address of a doubleword area where the accumulated CPU or vector time is returned.

,LINKAGE = SYSTEM
,LINKAGE = BRANCH
specifies the type of linkage used in TIMEUSED processing. LINKAGE = BRANCH indicates branch entry. You may specify or default to LINKAGE = BRANCH if you are a key zero supervisor state program running under a TCB or SRB. LINKAGE = SYSTEM indicates the linkage is by a PC instruction.

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

,CPU = TOD
,CPU = MIC
specifies that TIMEUSED should return the total CPU time in either TOD clock format (CPU = TOD) or in microseconds (CPU = MIC). You may specify CPU = MIC only if LINKAGE = SYSTEM. If you specify either CPU = option, you may not specify the VECTOR parameter.

,VECTOR = TOD
,VECTOR = MIC
specifies that TIMEUSED should return the total vector time in either TOD clock format (VECTOR = TOD) or in microseconds (VECTOR = MIC). You may specify VECTOR = only if LINKAGE = SYSTEM. If you specify either VECTOR = option, you may not specify the CPU parameter.

Register 15 contains the following hexadecimal return codes from TIMEUSED:

Hexadecimal Code	Meaning
00	The service completed successfully.
04	The CPU timer is not useable.
08	Unexpected error

Example 1
Operation: Using the unauthorized TIMEUSED service, request the total CPU time in TOD clock format to be stored at the address in register 2.

```
TIMEUSED   STORADR=(2),LINKAGE=SYSTEM,CPU=TOD
```

Example 2
Operation: Using the unauthorized TIMEUSED service in task mode, request the total vector time in microseconds to be stored at the address in register 2.

```
TIMEUSED   STORADR=(2),LINKAGE=SYSTEM,VECTOR=MIC
```

Example 3
Operation: Using the authorized TIMEUSED service, request the total CPU time in TOD clock format to be stored at the address in register 2.

```
TIMEUSED   STORADR=(2),LINKAGE=BRANCH
```

T6EXIT — Type 6 Exit

The T6EXIT macro returns control from a Type 6 SVC routine to the SVC first level interrupt handler (FLIH). This exit macro can only be used in a Type 6 SVC.

The T6EXIT macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede T6EXIT.
T6EXIT	
b	One or more blanks must follow T6EXIT.

RETURN=CALLER	**Default:** RETURN=CALLER
RETURN=DISPATCH	
RETURN=SRB	

The explanation of the RETURN parameter is as follows:

RETURN=
specifies how the Type 6 SVC has chosen to exit.

CALLER specifies that the return is directly to the caller or issuer of the SVC. CALLER is the default return option.

DISPATCH specifies that the return should be through the dispatcher. This function is for the use of routines that have suspended the current task.

No registers are returned to the caller.

SRB specifies that the system should immediately dispatch an SRB. This SRB must:

- Be initialized by the Type 6 SVC
- Be pointed to by register 1
- Execute in the same address space as the SVC. The SRB has the same format as the SCHEDULE SRB.

Note: No registers are returned to the caller.

Example

Operation: Terminate Type 6 SVC processing and return control from the Type 6 SVC to the caller of the SVC.

```
T6EXIT  RETURN=CALLER
```

VSMLIST — List Virtual Storage Map

The VSMLIST macro provides information about the allocation of virtual storage. All addresses returned by the macro are 31-bit addresses. The information is returned in a work area that you specify. You must set bytes 0-3 of the work area to zero before the first invocation of this macro for a specific request. The format of the work area is described under "Virtual Storage Management" in *SPL: Application Development Guide*.

This macro can be used in cross memory mode. All addresses are associated with the current address space.

The following information can be requested:

- The ranges of virtual storage allocated to the SQA, by subpool, and the free space within those ranges
- The ranges of virtual storage allocated to the CSA, by subpool, and the free space within those ranges
- The ranges of CSA space that are unallocated
- The ranges of virtual storage allocated to the LSQA in the current address space, by subpool, and the free space within those ranges
- The ranges of virtual storage allocated to private area subpools, by TCB, and the free space within those ranges
- The ranges of private area that are unallocated
- On entry to this macro, register 13 must contain the address of a 72-byte save area. VSMLIST preserves registers 2-13.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The VSMLIST macro is written as follows:

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

SP=SQA SP=CSA SP=LSQA SP=PVT SP=*sp list addr*	*sp list addr:* RX-type address or register (0), (2)
,WKAREA=*(addr,length)*	*addr:* RX-type address or register (0), (2) *length:* symbol, decimal digit, or register (0), (2) - (12).
,TCB=*(tcb addr)* ,TCB=*(tcb addr*,ALL) ,TCB=(, ALL)	**Default:** TCB address in PSATOLD. *tcb addr:* RX-type address or register (0), (2) - (12). **Note:** The TCB parameter is required only for SRB routines, if SP=PVT or SP=*sp list addr* and the list contains private area subpools.
,SPACE=ALLOC ,SPACE=FREE ,SPACE=UNALLOC	**Default:** SPACE=ALLOC **Note:** SPACE=UNALLOC can be specified only for SP=CSA or SP=PVT.
,LOC=ANY ,LOC=BELOW	**Default:** LOC=ANY
,REAL	
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM

The parameters are explained as follows:

SP=SQA
SP=CSA
SP=LSQA
SP=PVT
SP=*sp list addr*

specify the storage areas for which information is requested. The following subpools are listed for the specified storage areas:

- SQA: 226, 239, 245, 247, 248
- CSA: 227, 228, 231, 241
- LSQA: 205, 215, 225, 255
- PVT: 0, 1-127, 229, 230, 236, 237, 251, 252

GETMAIN/FREEMAIN/STORAGE processing translates the original subpool numbers that were specified on the GETMAIN, FREEMAIN, or STORAGE macros to internal subpool numbers as shown below:

Original Subpool Number	Internal Subpool Number
203, 204	205
213, 214	215
223, 224	225
233-235, 253, 254	255
240, 250	0

VSMLIST reports the translated internal subpool numbers, not the original subpool numbers. In addition, VSMLIST does not report invalid subpool numbers (subpool numbers greater than 255) or undefined subpool numbers.

If SP=*sp list addr* is specified, the user must supply the address of a subpool list. The first halfword of the list contains the number of entries in the list. Each of the following halfwords in the list contains a subpool number. If a valid subpool number appears more than once in the subpool list, it is reported only once.

,WKAREA=(*addr,length***)**
indicates the address and length of a user-supplied work area. The system uses this work area to hold the parameter list, control information, and data that is to be returned to the caller. The work area should begin on a word boundary and be a minimum of 4K bytes in length.

You must set bytes 0-3 of this work area to zero before the first invocation of VSMLIST for a specific request. See "Virtual Storage Management" in *SPL: Application Development Guide* for a description of the work area.

,TCB=(*tcb addr***)**
,TCB=(*tcb addr***,ALL)**
,TCB=(,ALL)
specify the TCB associated with the virtual storage allocated to the private area subpools. The TCB must be located in the currently addressable address space. If ALL is specified, the storage associated with the TCB and all of its subtasks is reported.

Notes:

1. If ALL is specified and the TCB is high in the task structure (for example, the TCB for RCT), more than one region could be listed. The regions in the private area are the RCT region, the V=V region, and the V=R region (for V=R jobs).
2. The TCB resides in storage below 16 megabytes.

,SPACE=ALLOC
,SPACE=FREE
,SPACE=UNALLOC
specify whether allocated, allocated and free, or unallocated storage is to be reported.

ALLOC indicates that the virtual addresses and lengths of blocks of storage allocated to the specific area are to be listed.

FREE indicates that in addition to the information supplied by ALLOC, the virtual addresses and lengths of free space within the allocated blocks are to be listed.

UNALLOC indicates that the virtual addresses and lengths of unallocated blocks of storage are to be listed. Both TCB and REAL are ignored when UNALLOC is specified.

Note: An allocated block of storage is a block that is a multiple of 4K in size and contains some storage that has been allocated via a GETMAIN or STORAGE macro. The free storage is the storage within an allocated block that has not been allocated via a GETMAIN or STORAGE macro. An unallocated block of storage is a block that is a multiple of 4K in size and contains no allocated storage.

,LOC=ANY
,LOC=BELOW
indicate which virtual storage control blocks are to be searched. If LOC=ANY is specified, all of the virtual storage control blocks are searched. If LOC=BELOW is specified, only those queues with virtual storage below 16 megabytes are searched.

,REAL
indicates that the high order bit of the address field of the allocated block descriptor is to be used to inform the caller which LOC parameter was used. If the storage block was allocated using any LOC specification of GETMAIN or STORAGE except LOC=(,BELOW), the indicator is turned on; if the storage block was allocated using the LOC=(,BELOW) parameter of the GETMAIN or STORAGE macros, the indicator is turned off. If REAL is not specified the indicator remains off (zero).

,LINKAGE = SYSTEM
,LINKAGE = BRANCH

indicate whether the VSMLIST routine uses a PC instruction (LINKAGE = SYSTEM) or branch entry (LINKAGE = BRANCH) for linkage and whether the VSMLIST routine provides serialization and recovery.

If LINKAGE = SYSTEM is specified, the VSMLIST routine provides linkage using a PC instruction and also provides recovery and serialization. The user cannot hold a lock higher than the local lock.

The caller's secondary ASID is preserved when a PC is issued; however, the caller cannot be in secondary addressing mode when issuing the macro.

Note: Serialization is not provided across calls to VSMLIST.

If LINKAGE = BRANCH is specified, the VSMLIST routine uses branch entry for linkage and does not provide recovery or serialization. Before issuing VMLIST, provide serialization as follows:

- For SQA or CSA requests, issue the SETLOCK macro as follows:

  ```
  SETLOCK OBTAIN,TYPE=VSMFIX
  ```

  When the system returns control to your program, issue the SETLOCK macro as follows:

  ```
  SETLOCK RELEASE,TYPE=VSMFIX
  ```

- For LSQA or PVT requests, obtain the LOCAL lock.

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

Hexadecimal Code	Meaning
0	The macro executed successfully and all the information has been placed in the data portion of the work area.
4	The macro executed successfully, but additional information has not been reported because there was not enough room in the data portion of the work area. To obtain the missing information, the user can continue to issue the macro, with the same options, until the return code in register 15 is 0.
8	An error occurred in scanning virtual storage control blocks. The information in the data area is valid, but incomplete. This return code is obtained only by users who specify LINKAGE = SYSTEM.
C	The work area was too small or either invalid parameters or invalid control information was detected. This return code is obtained only by users who specify LINKAGE = BRANCH. Users who specify LINKAGE = SYSTEM will receive a X'C78' abend.

Note: Bytes 0-3 of the work area also contain the return code. Prior to the first invocation of the VSMLIST macro, the user must set these bytes to zero. If the return code is 4 and the user wants to re-invoke the VSMLIST macro, these bytes must not be changed.

Example 1

Operation: List the ranges of the allocated and free storage in the SQA. Specify the address of the VSM work area in register 2 and the length of the work area in register 3.

```
VSMLIST SP=SQA,SPACE=FREE,WKAREA=((2),(3))
```

Example 2

Operation: List the ranges of the allocated storage in the CSA. Specify the address of the work area in register 2 and the length of the work area in register 3. Provide branch entry linkage.

```
VSMLIST SP=CSA,SPACE=ALLOC,WKAREA=((2),(3)),LINKAGE=BRANCH
```

Example 3

Operation: List the ranges of unallocated storage in the private area. The variable X contains the address of the work area, which has a length of 4096 bytes.

```
VSMLIST SP=PVT,SPACE=UNALLOC,WKAREA=(X,4096)
```

Example 4

Operation: List the ranges of allocated storage, below 16 megabytes, in each of the subpools specified in the subpool list at location Y. The variable X contains the address of the work area, which has a length of 4096 bytes.

```
VSMLIST SP=Y,SPACE=ALLOC,WKAREA=(X,4096),LOC=BELOW
```

VSMLOC — Verify Virtual Storage Allocation

The VSMLOC macro verifies that a given storage area has been allocated using the GETMAIN or STORAGE macros. All addresses communicated between the caller and the VSMLOC routine must be 31-bit addresses. You can use VSMLOC in cross memory mode. All addresses are associated with the current address space.

The VSMLOC macro is written as follows:

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

SQA CSA LSQA PVT CPOOLFIX CPOOLPAG CPOOLLCL	
,AREA=*(addr,length)*	*addr:* RX-type address or register (0) - (12). *length:* symbol, decimal digit or register (0), (2) - (12). Use only with SQA, CSA, LSQA, and PVT.
,AREA=*(addr)*	*addr:* RX-type address or register (0) - (12). Use only with CPOOLFIX, CPOOLPAG, and CPOOLLCL.
,TCB=*addr*	*addr:* RX-type address or register (0) - (12). Can only be specified with PVT.
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM

The parameters are explained as follows:

SQA
CSA
LSQA
PVT
used to verify that storage for SQA, CSA, LSQA, or PVT (private area storage) has been allocated in the current address space.

CPOOLFIX
used to verify that storage for a global fixed cell pool has been allocated. Users who obtain their storage from subpools 226, 227, 228, 239, or 245 should specify this keyword.

CPOOLPAG
used to verify that storage for a global pageable cell pool has been allocated. Users who obtain storage from subpools 231, 241, 247, or 248 should specify this keyword.

CPOOLLCL
used to verify that storage for a local cell pool has been allocated. Users who obtain storage from subpools 0-127, 203-205, 213-215, 223-225, 229, 230, 233-237, 240, 250-255 should specify this keyword.

,AREA=(*addr,length*)
indicates the start of the virtual storage area (*addr*) and the length of the virtual storage area (*length*) to be verified.

,AREA=(*addr*)
indicates the start of the virtual storage area (*addr*) to be verified.

,TCB=*addr*
indicates that VSMLOC is to place the address of the TCB associated with the verified storage in the register or storage area specified by the TCB parameter. If the return code from VSMLOC is not zero, the register or storage area specified by the TCB parameter is set to zero. The TCB parameter can only be specified with PVT.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
indicates the type of linkage that VSMLOC is to use and also indicates whether the VSMLOC routine is to provide recovery and serialization.

If LINKAGE=SYSTEM is specified, the VSMLOC routine uses a basic PC instruction for linkage and provides recovery and serialization. The following restriction applies: If LSQA, PVT, or CPOOLLCL is specified, the user can hold only the local lock.

The caller's secondary ASID is preserved when a basic PC is issued; however, the caller cannot be in secondary addressing mode when issuing the macro.

If LINKAGE=BRANCH is specified, the VSMLOC routine uses branch entry linkage and does not provide recovery or serialization. Before issuing VSMLOC, provide serialization as follows:

- For CSA, SQA, and CPOOLFIX requests, issue the SETLOCK macro as follows:

  ```
  SETLOCK OBTAIN,TYPE=VSMFIX
  ```

  When the system returns control to your program, issue the SETLOCK macro as follows:

  ```
  SETLOCK RELEASE,TYPE=VSMFIX
  ```

- For CPOOLPAG requests, issue the SETLOCK macro as follows:

  ```
  SETLOCK OBTAIN,TYPE=VSMPAG
  ```

  When the system returns control to your program, issue the SETLOCK macro as follows:

  ```
  SETLOCK RELEASE,TYPE=VSMPAG
  ```

- For LSQA, CPOOLLCL, and private area storage requests, obtain the LOCAL lock.

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

Hexadecimal Code	Meaning
0	The virtual storage specified has been allocated in the given storage area.
4	The virtual storage is not in the specified area or overlaps free space or different subpools in the given area.
8	An error occurred in processing virtual storage control blocks. This return code is obtained only by users who specify LINKAGE=SYSTEM.
12	The input is invalid. This return code is obtained only by users who specify LINKAGE=BRANCH. Users who specify LINKAGE=SYSTEM will receive a X'C78' abend.

If the return code in register 15 is 0, byte 3 in register 0 contains the subpool ID. If the return code in register 15 is not 0, byte 3 in register 0 contains 0.

Note: Users should mask off bytes 0-2 before attempting to use the subpool ID returned in register 0.

Example 1

Operation: Verify that the virtual storage, starting at the address given in register 2 and having a length specified in register 3, has been allocated in the SQA.

```
VSMLOC SQA,AREA=((2),(3))
```

Example 2

Operation: Verify that the 8-bytes of virtual storage starting at X have been allocated in the CSA. Use a PC instruction for linkage and let VSMLOC provide recovery and serialization.

```
VSMLOC CSA,AREA=(X,8),LINKAGE=SYSTEM
```

Example 3

Operation: Verify that the 8-bytes of virtual storage starting at the address specified in register 2 have been allocated in the LSQA. Use branch entry for linkage.

```
VSMLOC LSQA,AREA=((2),8),LINKAGE=BRANCH
```

Example 4

Operation: Verify that the virtual storage, starting at X and having a length specified in register 3, has been allocated in private area storage. Use branch entry for linkage.

```
VSMLOC PVT,AREA=(X,(3)),LINKAGE=BRANCH
```

Example 5

Operation: Verify that the 100 bytes of virtual storage starting at the address specified in register 1 have been allocated in private area storage. The address of the TCB associated with the storage verified is returned in register 4.

```
VSMLOC PVT,AREA=((1),100),TCB=(4),LINKAGE=BRANCH
```

VSMREGN — Obtain Private Area Region Size

The VSMREGN macro provides the virtual starting address and sizes of the private area regions associated with a given TCB in the current address space.

VSMREGN runs in the state and key of the caller. You can use VSMREGN in cross memory mode. All addresses communicated between VSMREGN and the caller are 31-bit addresses, associated with the current address space. If the TCB default is not used, the caller must hold the local lock.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The VSMREGN macro is written as follows:

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

WKAREA=*addr*	*addr:* RX-type address or register (0) - (12).
,TCB=*tcb addr*	**Default:** (except for SRB routines) TCB *tcb addr:* RX-type address or register (0), (2) - (12).

The parameters are written as follows:

WKAREA=*addr*
indicates the virtual address of a 16-byte work area, which is used by VSMREGN to return the requested information. The format of the work area is:

Bytes	Meaning
0-3	Virtual address of the region below 16 megabytes
4-7	Length of the region below 16 megabytes
8-11	Virtual address of the region above 16 megabytes
12-15	Length of the region above 16 megabytes

,TCB=*tcb addr*
indicates the virtual address of the TCB to be used to identify the region (the region control task (RCT) region, the V=V region, or the V=R region). SRB routines and routines whose currently addressable address space is not the home address space must specify the TCB operand. They cannot use the default value.

Note: The TCB resides in storage below 16 megabytes.

When control returns from the VSMREGN routine, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	Successful completion

Example 1

Operation: Find the virtual address and length of the private area of the TCB whose address is in PSATOLD. Return the information in the work area whose address is given in register 2.

```
VSMREGN WKAREA=(2)
```

Example 2

Operation: Find the virtual address and length of the private area of the TCB specified in register 3. Return this information in the work area whose address is given in register 2.

```
VSMREGN WKAREA=(2),TCB=(3)
```

Example 3

Operation: Find the virtual address and length of the private area of the TCB whose address is X. Return this information in the work area whose address is given in register 2.

```
VSMREGN WKAREA=(2),TCB=X
```

Example 4

Operation: Find the virtual address and length of the private area of the TCB whose address is given in register 3. Return this information in the work area whose address is X.

```
VSMREGN WKAREA=X,TCB=(3)
```

WAIT — Wait for One or More Events

The WAIT macro is used to inform the system that performance of the active task cannot continue until one or more specific events, each represented by a different ECB (event control block), have occurred. Bit 0 and bit 1 of each ECB must be set to 0 before it is used. The caller must be enabled, unlocked, and in primary address space control (ASC) mode.

The system takes the following action:

- For each event that has already occurred (each ECB is already posted), the count of the number of events is decreased by 1.
- If the number of events is 0 by the time the last event control block is checked, control is returned to the instruction following the WAIT macro.
- If the number of events is not 0 by the time the last ECB is checked, control is not returned to the issuing program until sufficient ECBs are posted to bring the number to 0. Control is then returned to the instruction following the WAIT macro.

The WAIT macro is written as follows:

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

event nmbr,	*event nmbr:* symbol, decimal digit, or register (0) or (2) - (12). **Default:** 1 **Value range:** 0-255
ECB=*ecb addr* ECBLIST=*ecb list addr*	*ecb addr:* RX-type address, or register (1) or (2) - (12). *ecb list addr:* RX-type address, or register (1) or (2) - (12).
,LONG=NO ,LONG=YES	**Default:** LONG=NO
,LINKAGE=SVC ,LINKAGE=SYSTEM	**Default**: LINKAGE=SVC
,EUT=NOSAVE ,EUT=SAVE	**Default**: EUT=NOSAVE
,RELATED=*value*	*value:* Any valid macro keyword specification.

The parameters are explained as follows:

event nmbr,
specifies the number of events waiting to occur.

ECB=*ecb addr*
ECBLIST=*ecb list addr*
specifies the address of an ECB on a fullword boundary or the address of a virtual storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an ECB; the high order bit in the last fullword must be set to 1 to indicate the end of the list.

The ECB parameter is valid only if the number of events is specified as one or is omitted. The number of ECBs in the list specified by the ECBLIST form must be equal to or greater than the specified number of events.

If you specify ECBLIST, *ecb list addr* and all ECBs on the list must be in the home address space.

,LONG = NO
,LONG = YES
specifies whether the task is entering a long wait (YES) or a regular wait (NO).

,LINKAGE = SVC
,LINKAGE = SYSTEM
specifies whether the caller is in cross memory mode (LINKAGE = SYSTEM) or not (LINKAGE = SVC).

When the caller is not in cross memory mode (the primary, secondary, and home address spaces are the same), use LINKAGE = SVC. With this parameter, linkage is through an SVC instruction.

When the caller is in cross memory mode (the primary, secondary, and home address spaces are not the same), use LINKAGE = SYSTEM. With this parameter, linkage is through a PC instruction. Note that the ECB must be in the home address space.

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

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE), and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

The RELATED parameter may be used, for example, as follows:

```
WAIT1   WAIT     1,ECB=ECB,RELATED=(RESUME1,
                 'WAIT FOR EVENT')
  .
  .
  .
RESUME1 POST     ECB,0,RELATED=(WAIT1,
                 'RESUME WAITER')
```

Note: Each of these macros will fit on one line when coded, so there is no need for a continuation indicator.

,EUT = NOSAVE
,EUT = SAVE
specifies whether EUT FRRs, if present, should be preserved around the WAIT processing. Specify this keyword only if you specify LINKAGE = SYSTEM.

CAUTION:
A job step with all of its tasks in a WAIT condition is terminated upon expiration of the time limits that apply to it.

Example: You have previously initiated one or more activities to be completed asynchronously to your processing. As each activity was initiated, you set up an ECB in which bits 0 and 1 were set to 0. You now wish to suspend your task via the WAIT macro until a specified number of these activities have been completed.

Completion of each activity must be made known to the system via the POST macro. POST causes an addressed ECB to be marked complete. If completion of the event satisfies the requirements of an outstanding WAIT, the waiting task is marked ready and will be executed when its priority allows.

Example 1

Operation: Wait for one event to occur (with a default count).

```
        WAIT  ECB=WAITECB
        .
        .
WAITECB DC    F'0'
```

Example 2

Operation: Wait for 2 events to occur.

```
          WAIT   2,ECBLIST=LISTECBS
          .
          .
LISTECBS  DC  A(ECB1)
          DC  A(ECB2)
          DC  X'80'
          DC  AL3(ECB3)
```

Example 3

Operation: Enter a long wait for a task.

```
          WAIT   1,ECBLIST=LISTECBS,LONG=YES
          .
          .
          .
LISTECBS  DC  A(ECB1)
          DC  A(ECB2)
          DC  X'80'
          DC  AL3(ECB3)
```

WTL — Write To Log

Note: IBM recommends you use the WTO macro with the MCSFLAG=HRDCPY parameter instead of WTL, because WTO supplies more data than WTL.

The WTL macro causes a message to be written to the system log. The message can include any character that can be used in a C-type (character) DC statement, and is assembled as a variable-length record.

The description of the WTL macro follows. The WTL macro is also described in *Application Development Macro Reference* with the exception of the OPTION parameter. The use of the OPTION parameter is restricted to users who are authorized (APF-authorized, in system key 0-7, or in supervisor state). If the OPTION keyword is used by a non-authorized user, it is ignored.

Note: The exact format of the output of the WTL macro varies depending on the job entry subsystem (JES2 or JES3) that is being used, the output class that is assigned to the log at system initialization, and whether DLOG is in effect for JES3. In JES3, system log entries are preceded by a 23-character prefix that includes a time stamp and routing information. If the combined prefix and message exceeds 126 characters, the log entry is truncated at the first blank or comma encountered when scanning backward from the 126th character of the combined prefix and message. See *Operations: JES3 Commands* for information about the format of the log entry when using JES3.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTL.
WTL	
b	One or more blanks must follow WTL.
'msg'	*msg:* Up to 126 characters if OPTION=NOPREFIX is specified. Up to 128 characters if OPTION=PREFIX is specified.
,OPTION=PREFIX ,OPTION=NOPREFIX	**Default:** OPTION=NOPREFIX

The parameters are explained as follows:

'msg'
specifies the message to be written to the system log. The message must be enclosed in apostrophes, which will not appear in the system log. See Figure 27 for a list of the printable EBCDIC characters passed to display devices or printers.

Note: If the msg text exceeds 126 characters, truncation occurs at the last embedded blank before the 126th character; when there are no embedded blanks, truncation occurs after the 126th character.

,OPTION = PREFIX
,OPTION = NOPREFIX
specifies whether the WTL text contains a prefix identifying the system log record. If PREFIX is specified, the text already contains a prefix. If NOPREFIX is specified or if this parameter is omitted, a 2-character prefix will be added by the control program. The OPTION keyword is ignored by any program running in the JES3 primary address space.

Hex Code	EBCDIC Character	Hex Code	EBCDIC Character	Hex Code	EBCDIC Character	Hex Code	EBCDIC Character
40	(space)	7B	#	99	r	D5	N
4A	¢	7C	@	A2	s	D6	O
4B	.	7D	'	A3	t	D7	P
4C	<	7E	=	A4	u	D8	Q
4D	(	7F	"	A5	v	D9	R
4E	+	81	a	A6	w	E2	S
4F	|	82	b	A7	x	E3	T
50	&	83	c	A8	y	E4	U
5A	!	84	d	A9	z	E5	V
5B	$	85	e	C1	A	E6	W
5C	*	86	f	C2	B	E7	X
5D	)	87	g	C3	C	E8	Y
5E	;	88	h	C4	D	E9	Z
5F	¬	89	i	C5	E	F0	0
60	-	91	j	C6	F	F1	1
61	/	92	k	C7	G	F2	2
6B	,	93	l	C8	H	F3	3
6C	%	94	m	C9	I	F4	4
6D	—	95	n	D1	J	F5	5
6E	>	96	o	D2	K	F6	6
6F	?	97	p	D3	L	F7	7
7A	:	98	q	D4	M	F8	8
						F9	9

Figure 27. Characters Printed or Displayed on an MCS Console

Notes:

1. If the display service or printer is defined to JES3, the following characters are translated to blanks:

   | ! ; ¬ : "

2. The system recognizes the following hexadecimal representations of the U.S. national characters: @ as X'7C'; $ as X'5B'; and # as X'7B'. In countries other than the U.S., the U.S. national characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the $ character generates a X'4A'.

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

- 00 — Successful completion.
- 04 — WTL processing was not successful. Register 0 contains one of the following reason codes:
  04 — Recovery could not be established.
  08 — System log is not active.
  12 — WTL limit reached.
  16 — Record size too small.

Example 1

Operation: Write a message to the system log.

```
WTL   'THIS IS THE STANDARD FORMAT FOR THE WTL MACRO'
```

Example 2

Operation: Write a message to the system log specifying a prefix to identify the system log record.

```
WTL   'QL THIS FORMAT OF THE WTL USES THE OPTION KEYWORD',OPTION=PREFIX
```

WTL (List Form)

The list form of the WTL macro is used to construct a control program parameter list. The message parameter must be provided in the list form of the macro.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTL.
WTL	
b	One or more blanks must follow WTL.
'msg'	*msg:* Up to 126 characters.
,MF=L	

The *'msg'* parameter is explained under the standard form of the WTL macro. The OPTION keyword is not permitted on the list form of the WTL macro. A description of the MF parameter follows:

,MF=L
specifies the list form of the WTL macro.

Note: If msg text exceeds 126 characters, truncation occurs at the last embedded blank before the 126th character; when there are no embedded blanks, truncation occurs after the 126th character.

Example

Operation: Build a parameter list for a message to be written to the system log.

```
LOGMSG   WTL   'FUNCTION XXX COMPLETE',MF=L
```

WTL (Execute Form)

The execute form of the WTL macro uses a remote control program parameter list. The parameter list can be generated by the list form of WTL. You cannot modify the message in the execute form.

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

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

MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).
,OPTION=PREFIX ,OPTION=NOPREFIX	**Default:** OPTION=NOPREFIX

The OPTION parameter is explained under the standard form of the WTL macro; this parameter is explained as follows:

MF=(E,*ctrl addr*)
specifies the execute form of the WTL macro. This form uses a remote control program parameter list.

Example

Operation: Write a message constructed in the list form of WTL.

```
WTL   MF=(E,LOGMSG)
```

WTO — Write to Operator

The WTO macro causes a message to be written to one or more operator consoles.

An authorized user (supervisor state with protection key 0-7) can issue a multiple line WTO message of up to 255 lines with one WTO macro. If you are coding more than one multiple line message, and you want to connect the messages, you must ensure that the left-most three bytes of register 0 are set correctly. For the first request (of up to 255 lines), these three bytes must be zero. For subsequent requests, the first three bytes of register 0 may contain the message identifier that the WTO service routine returns in register 1 after the first request. The CONNECT parameter provides another way to connect multiline WTO messages. Therefore, an authorized user can actually issue connect messages that total more than 255 lines.

Do not use the MSGTYP parameter unless you are familiar with multiple console service (MCS), because using this parameter improperly might interfere with the message routing scheme.

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

---

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

'msg' *('text')* *('text',line type)* *('text',line type,...,* *'text',line type)*	*msg:* Up to 126 characters. *text:* Up to 126 characters. Note:*'msg'* or *('text...')* must be the first parameter you code. The permissible *line types*, text lengths, and maximum numbers are shown below: line type — text — maximum number C — 35 char — 1 C type L — 71 char — 2 L type D — 71 char — 255 D type DE — 71 char or E — None — 1 DE type or 1 E type The maximum total number of lines that can be coded in one instruction is 255.
,ROUTCDE=*(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,DESC=*(descriptor code)*	*descriptor code:* decimal digit from 1 to 11. The *descriptor code* is one or more codes, separated by commas.
,AREAID=*id char*	*id char:* alphabetic character A - J, Z.
,MSGTYP=*(msg type)*	*msg type:* any of the following N — SESS,JOBNAMES Y — SESS,STATUS SESS — JOBNAMES,STATUS JOBNAMES — SESS,JOBNAMES,STATUS STATUS

,MCSFLAG=*(flag name)*	*flag name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY CMD BUSYEXIT
,CONNECT=*connect field*	*connect field:* RX-type address or register (2) - (12)
,CONSID=*console id*	*console id:* RX-type address or register (2) - (12)
,KEY=*key*	*key:* RX-type address or register (2) - (12)
,TOKEN=*token*	*token:* RX-type address or register (2) - (12)

The parameters are explained as follows:

'msg'
('text')
('text',line type)
('text',line type,...,'text',line type)
specifies the message or multiple-line message to be written to one or more operator consoles.

The first format is used to write a single-line message to the operator. In the format, the message must be enclosed in apostrophes, which do not appear on the console. It can include any character that can be used in a character (C-type) DC instruction. When a program issues a WTO macro, the system translates the text; only standard printable EBCDIC characters, shown in Figure 27 on page 788 are passed to the display devices. All other characters are replaced by blanks. If the terminal does not have dual-case capability, it prints lowercase characters as uppercase. The message is assembled as a variable-length record.

The second and third formats are used to write a multiple-line message to the operator. For a problem program the message can be up to ten lines long; the system truncates the message at the end of the tenth line. The ten-line limit does not include the control line (message IEE932I), as explained under line type C below. The message can be up to 255 lines long for an authorized program.

Note: If the second format is coded without repetition, for example, *('text')*, the message appears as a single-line message.

The text is one line of the multiple-line message. A line consists of a character string enclosed in apostrophes (which do not appear on the operator console). Any character valid in a C-type DC instruction can be coded. The maximum number of characters depends on which line type is specified.

The line type defines the type of information contained in the 'text' field of each line of the message:

C
indicates that the 'text' parameter is the text to be contained in the control line of the message. The control line normally contains a message title. C may only be coded for the first line of a multiple-line message. If this parameter is omitted and descriptor code 9 is coded, the system generates a control line (message IEE932I) containing only a message identification number. The control line remains static during framing operations on a display console (provided that the message is displayed in an out-of-line display area). Control lines are optional.

L
indicates that the 'text' parameter is a label line. Label lines contain message heading information; they remain static during framing operations on a display console (provided that the message is displayed in an out-of-line display area). Label lines are optional. If coded, lines must either immediately follow the control line or another label

line or be the first line of the multiple-line message if there is no control line. Only two label lines may be coded per message. See "Embedding Label Lines in a Multiline Message" in *SPL: Application Development Guide* for additional information about how to include multiple label lines within a message.

D

indicates that the 'text' parameter contains the information to be conveyed to the operator by the multiple-line message. During framing operations on a display console, the data lines are paged.

DE

indicates that the 'text' parameter contains the last line of information to be passed to the operator. Specify DE on the last line of text of the WTO. If there is no text on the last line, specify E.

E

indicates that the previous line of text was the last line of text to be passed to the operator. The 'text' parameter, if any, coded with a line type of E is ignored. Specify E on the last line of the WTO if that line has no text. If the last line has text, specify DE.

,ROUTCDE=(*routing code***)**

specifies the routing code(s) to be assigned to the message.

The routing codes are:

Message Routing Code	Definition
1	Master console action
2	Master console information
3	Tape pool
4	Direct access pool
5	Tape library
6	Disk library
7	Unit record pool
8	Teleprocessing control
9	System security
10	System error/maintenance/system programmer information
11	Programmer information
12	Emulators
13-20	Reserved for customer use
21-28	Reserved for IBM or customer-defined subsystem use
29-41	Reserved for IBM
42	General information about JES2 or JES3
43-64	Reserved for JES2 or JES3
65-96	Messages associated with particular processors
97-128	Messages associated with particular devices

If you omit the ROUTCDE, DESC, and CONSID keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.

Note: Routing codes 1, 2, 3, 4, 7, 8, 10, and 42 cause hard copy of the message when display consoles are used, or more than one console is active. All other routing codes may go to hard copy as a PARMLIB option or as a result of a VARY HARDCPY command.

,DESC = *(descriptor code)*

specifies the message descriptor code(s) to be assigned to the message. Descriptor codes 1 through 6 and descriptor code 11 are mutually exclusive. Codes 7 through 10 can be assigned in combination with any other code.

The descriptor codes are:

1	System failure	8	Out-of-line message
2	Immediate action required	9	Operator request
3	Eventual action required	10	Dynamic status displays
4	System Status	11	Critical eventual action requested
5	Immediate command response		
6	Job status		
7	Retain action message for life-of-task		

All WTO messages with descriptor codes 1, 2, 3, or 11 are action messages that have an @ or * sign displayed before the first character of the message. This indicates a need for operator action. On operator consoles that support color, descriptor codes determine the color in which a message should be displayed. The colors used for different descriptor codes are described in *Operations: System Commands*.

The system holds messages with descriptor codes 1, 2, 3, or 11 until you delete them. When you no longer need messages with descriptor codes 1, 2, 3, or 11, you should delete those messages using the DOM macro. If messages with descriptor codes 1, 2, 3, or 11 also have descriptor code 7, the system deletes them automatically at task termination.

,AREAID = *id char*

specifies a display area of the console screen on which a multiple-line message is to be written. This parameter is meaningful only for out-of-line MLWTO messages that are to be sent to display consoles.

The character Z designates the message area (the screen's general message area, rather than a defined display area); it is assumed nothing is specified.

Notes:

1. When you specify AREAID, you must specify descriptor codes 8 and 9.
2. If this parameter specifies an area, the area could be overlaid by a currently running dynamic display. Support for queuing messages with descriptor code 8 is by console id only.

,MSGTYP = *(msg type)*

specifies how the message is to be routed.

For SESS, JOBNAMES, or STATUS, the message is to be routed to the console and TSO terminal in operator mode that issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed only to those consoles that requested the information.

For Y or N, the message type specifies whether flags are to be set in the WTO macro expansion to describe what functions (MONITOR SESS, MONITOR JOBNAMES, and MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter.

,MCSFLAG = *(flag name)*

specifies one or more flag names whose meanings are shown below:

Figure 28. MCSFLAG Flag Names

Flag Name	Meaning
REG0	Queue the message to the console whose source ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
RESP	The WTO is an immediate command response.
REPLY	This WTO is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
QREG0	Queue the message unconditionally to the console whose source ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
NOTIME	Do not append time to the message.
NOCPY	Do not queue the message for hard copy.
CMD	The WTO is a recording of a system command issued for hardcopy log purposes.
BUSYEXIT	If there are no message or console buffers for either MCS or JES3, or there is a JES3 WTO staging area excess, the WTO is terminated with a X'20' return code and a reason code, in register 0, equal to the number of active WTO buffers for the issuer's address space. If BUSYEXIT is not specified, the WTO will go into a wait state if WTO buffers are not available.

,CONNECT = *connect field*

specifies a field containing the 4-byte message number of the previous WTO that this WTO is to be connected to. This message number is obtained as an output parameter (returned in register 1) from the previous WTO. This parameter is mutually exclusive with the CONSID parameter, and it is valid only for continuation of multiple-line messages. When this parameter is specified in the list form, it must be coded as CONNECT= with nothing after the =.

Note: You can still use register 0, mentioned at the beginning of the WTO macro description, to connect WTO messages. If you specify both, however, the system uses the CONNECT parameter. It is recommended that new users use the CONNECT parameter.

,CONSID = *console id*

specifies a 4-byte field containing the ID of the console to receive a message. Use this ID in place of a console ID in register 0. If you specify a 4-byte console ID, you must use CONSID instead of register 0. If you specify a 1-byte console ID, you must right justify and pad to the left with zeros.

Notes:

1. If you code the CONSID parameter using a register, the register must contain the console ID itself, rather than the address of the console ID.
2. When you code CONSID on the list form of WTO, code it as CONSID= with nothing after the =.
3. Do not use both CONSID and register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSID parameter to an existing invocation of WTO.
4. CONSID is mutually exclusive with the CONNECT parameter.

,KEY=*key*

specifies a field containing an 8-byte key to be associated with this message. The key must be EBCDIC if used with the MVS DISPLAY R command for retrieval purposes, but it must not be '*'. If a register is used, it contains the address of the key. When this parameter is specified in the list form, it must be coded as KEY= with nothing after the =.

,TOKEN=*token*

specifies a field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token must be unique within an address space. When this parameter is specified in the list form, it must be coded as TOKEN= with nothing after the =.

Note: If you code the TOKEN parameter using a register, the register must contain the token itself, rather than the address of the token.

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 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 the WTO macro returns control, the output registers contain the following values:

Register	Contents
0	Used as a work register by the macro.
1	Message identification number if the WTO macro completed normally. If you are using the CONNECT parameter to connect WTO messages, store this value in the 4-byte CONNECT field and set register 1 to zero. Otherwise, register 1 is used as a work register by the macro.
2-13	Unchanged.
14	Used as a work register by the macro.
15	Return code.

Upon return from the WTO macro, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	No errors encountered.
04	Number of lines passed was 0; request is ignored. Number of lines passed was greater than 255 and you did not specify the CONNECT parameter; only 255 lines are processed. Message text length for a line was less than 1; all lines up to error line are processed.
08	Connecting message ID passed in register 0 does not match any on queue. Request is ignored.
0C	Line type not valid. An end has been forced at the point of the error except if the first line is an E line, in which case the request is ignored.
20	WTO processing has been terminated since it would have caused a wait state, and BUSYEXIT was specified. Register 0 contains the reason code, the number of active WTO buffers for the issuer's address space.
30	Required resource for routing code 11 was not available. Request is ignored for routing code 11; if any other routing code is specified, the request is processed.

Notes:

1. You must clear register 0 except under the following circumstances:
   - You are using register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) with MCSFLAG=REG0. However, IBM recommends using the CONSID parameter rather than register 0.
   - You are using register 0 to pass a message identifier to connect multiple-line messages. However, IBM recommends using the CONNECT parameter rather than register 0.
2. If the list and execute forms of the WTO macro are in separate modules, both modules must be assembled or compiled with the same level of WTO.
3. If the execute form of the macro specifies CONNECT, CONSID, KEY,or TOKEN then the list form, to ensure that the parameter list is generated correctly, must specify the same parameter(s) without data. If data is specified, the system issues an MNOTE and ignores the data.
4. For any WTO parameters that allow a register specification, the value must be right-justified in the register.

Example 1

Operation: Issue a WTO that describes a situation that must be resolved immediately. The message must appear on the master console and the system maintenance console as an immediate action message.

```
WTO  'THIS IS AN IMMEDIATE ACTION MESSAGE',ROUTCDE=(1,10),DESC=(2)
```

Example 2

Operation: Issue a message that is the response to a command. The message must appear only on the console that entered the command. Prior to issuing the WTO, the issuing console's identifier must be placed in register 0.

```
WTO  'THIS IS A COMMAND RESPONSE',DESC=(5),MCSFLAG=(REG0,RESP)
```

WTO (List Form)

The list form of the WTO macro is used to construct a control parameter list.

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

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

'msg' *('text')* *('text',line type)* *('text',line type,...,* *'text',line type)*	*msg:* Up to 126 characters. *text:* Up to 126 characters. **Note:***'msg'* or *('text...')* must be the first parameter you code. The permissible *line types*, text lengths, and maximum numbers are shown below: line type text maximum number C 35 char 1 C type L 71 char 2 L type D 71 char 255 D type DE 71 char 1 DE type or E None 1 E type The maximum total number of lines that can be coded in one instruction is 255.
,ROUTCDE=*(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,DESC=*(descriptor code)*	*descriptor code:* decimal digit from 1 to 11. The *descriptor code* is one or more codes, separated by commas.
,AREAID=*id char*	*id char:* alphabetic character A - Z.
,MSGTYP=*(msg type)*	*msg type:* any of the following N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=*(flag name)*	*flag name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY CMD BUSYEXIT
,CONNECT=	Parameter value not required for list form. Code only ,CONNECT=. If you code CONNECT on the list form of WTO, you must code CONNECT on the execute form.
,CONSID=	Parameter value not required for list form. Code only ,CONSID=. If you code CONSID on the list form of WTO, you must code CONSID on the execute form.

,KEY=	Parameter value not required for list form. Code only ,KEY=. If you code KEY on the list form of WTO, you must code KEY on the execute form.
,TOKEN=	Parameter value not required for list form. Code only ,TOKEN=. If you code TOKEN on the list form of WTO, you must code TOKEN on the execute form.

,MF=L

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

,MF=L
specifies the list form of the WTO macro.

WTO (Execute Form)

The execute form of the WTO macro uses a remote control parameter list. The parameter list can be generated by the list form of WTO. The message cannot be modified on the execute form of the macro.

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

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

CONNECT=*connect field*	*connect field:* RX-type address or register (2) - (12) If you code CONNECT on the execute form of WTO, you must code CONNECT on the list form.
,CONSID=*console id*	*console id:* RX-type address or register (2) - (12) If you code CONSID on the execute form of WTO, you must code CONSID on the list form.
,KEY=*key*	*key:* RX-type address or register (2) - (12) If you code KEY on the execute form of WTO, you must code KEY on the list form.
,TOKEN=*token*	*token:* RX-type address or register (2) - (12) If you code TOKEN on the execute form of WTO, you must code TOKEN on the list form.
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) - (12).

This parameter is explained under the standard form of the WTO macro, with the following exceptions:

,MF=(E, *ctrl addr)*
specifies the execute form of the WTO macro. *ctrl addr* defines the area into which the system stores the parameter list.

Example

Operation: Write a message with a pre-built parameter list pointed to by register 1.

```
WTO     MF=(E,(1))
```

WTOR — Write to Operator with Reply

The WTOR macro causes a message requiring a reply to be written to one or more operator consoles and the hardcopy log. The macro also provides the information required by the system to return the reply to the issuing program.

For information about how to select a macro for an MVS/SP version other than the current version, see " Selecting the Macro Level" on page 1.

Do not use the MSGTYP parameter unless you are familiar with multiple console service (MCS), because using this parameter improperly could impede the entire message routing scheme.

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

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

'msg',reply addr,reply length,ecb addr	*msg:* Up to 122 characters. *reply addr:* A-type address, or register (2) - (12). *reply length:* symbol, decimal number, or register (2) - (12). The minimum length is 1; the maximum length is 119. *ecb addr:* A-type address, or register (2) - (12). **Note:** *'msg',reply addr,reply length,ecb addr* must be the first parameter you code.
,ROUTCDE=*(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,MSGTYP=*(msg type)*	*msg type:* any of the following: N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=*(flag name)*	*flag name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY CMD
,CONSID=*console id*	*console id:* RX-type address or register (2) - (12)
,KEY=*key*	*key:* RX-type address or register (2) - (12)
,TOKEN=*token*	*token:* RX-type address or register (2) - (12)

The parameters are explained as follows:

'msg', reply addr,reply length, ecb addr

'msg' specifies the message to be written to the operator's console. The message must be enclosed in apostrophes, which do not appear on the console. It can include any character that can be used in a character (C-type) DC instruction. When a program issues a WTOR macro, the system translates the text; only standard printable EBCDIC characters are passed to the display devices. See Figure 27 on page 788 for a list of the printable EBCDIC characters. All other characters are replaced by blanks. If the terminal does not have dual-case capability, it prints lowercase characters as uppercase. The message is assembled as a variable-length record.

Note: All WTOR messages are action messages. An indicator appears before the first character of an action message to indicate a need for operator action. The system assigns descriptor code 7 to every WTOR message. Descriptor code 7 causes the message to be retained until task termination unless you receive a reply or you delete the message with the DOM macro. You should delete any unanswered WTOR messages that are no longer current.

reply addr specifies the address in virtual storage of the area into which the system is to place the reply. The reply is left-justified at this address.

reply length specifies the length, in bytes, of the reply message.

ecb addr specifies the address of the event control block (ECB) to be used by the system to indicate the completion of the reply and the id of the replying console. After the system receives the reply, the ECB appears as follows:

Offset	Length(bytes)	Contents
0	1	Completion code
1	2	Reserved
3	1	Console ID in hexadecimal (If you code a 4-byte console ID on the CONSID parameter, you will receive only the low-order byte.)

,ROUTCDE = (*routing code*)

specifies the routing code(s) to be assigned to the message.

The routing codes are:

Message Routing Code	Definition
1	Master console action
2	Master console information
3	Tape pool
4	Direct access pool
5	Tape library
6	Disk library
7	Unit record pool
8	Teleprocessing control
9	System security
10	System error/maintenance/system programmer information
11	Programmer information
12	Emulators
13-20	Reserved for customer use
21-28	Reserved for IBM or customer-defined subsystem use
29-41	Reserved for IBM
42	General information about JES2 or JES3
43-64	Reserved for JES2 or JES3
65-96	Messages associated with particular processors
97-128	Messages associated with particular devices

If you omit the ROUTCDE and CONSID keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.

,MSGTYP = *(msg type)*
specifies how the message is to be routed.

For SESS, JOBNAMES, or STATUS, the message is to be routed to the console and TSO terminal in operator mode that issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed only to those consoles that requested the information.

For Y or N, the message type specifies whether flags are to be set in the WTOR macro expansion to describe what functions (MONITOR SESS, MONITOR JOBNAMES, and MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter.

,MCSFLAG = *(flag name)*
specifies one or more flag names whose meanings are shown below:

Figure 29. MCSFLAG Flag Names

Flag Name	Meaning
REG0	Queue the message to the console whose source ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded on the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
RESP	The WTOR is an immediate command response.
REPLY	This is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
QREG0	Queue the message unconditionally to the console whose source ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded on the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
NOTIME	Do not append time to the message.
NOCPY	Do not queue the message for hard copy.
CMD	The WTOR is a recording of a system command issued for hardcopy log purposes.

,CONSID = *console id*
specifies a 4-byte field containing the ID of the console to receive a message. Use this ID in place of a console ID in register 0. If you specify a 4-byte console ID, you must use CONSID instead of register 0. If you specify a 1-byte console ID, you must right justify and pad to the left with zeros.

Notes:

1. If you code the CONSID parameter using a register, the register must contain the console ID itself, rather than the address of the console ID.
2. When you code CONSID on the list form of WTOR, code it as CONSID= with nothing after the =.
3. Do not use both CONSID and register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSID parameter to an existing invocation of WTOR.

,KEY = *key*
specifies a field containing an 8-byte key to be associated with this message. The key must be EBCDIC if used with the MVS DISPLAY R command for retrieval purposes, but it must not be '*'. If a register is used, it contains the address of the key. When this keyword is specified in the list form, it must be coded as KEY= with nothing after the =.

,TOKEN = *token*
specifies a field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token must be unique within an address space. When this

keyword is specified in the list form, it must be coded as TOKEN= with nothing after the =.

Note: If you code the TOKEN parameter using a register, the register must contain the token itself, rather than the address of the token.

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 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 the WTOR macro returns control, the output registers contain the following values:

Register	Contents
0	Used as a work register by the macro.
1	Message identification number if the WTOR macro completed normally; otherwise, used as a work register by the macro.
2-13	Unchanged.
14	Used as a work register by the macro.
15	Return code.

Upon return from the WTOR macro, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	No errors encountered.
04	Number of lines passed was 0; request is ignored. Message text length for a line was less than 1; all lines up to the error line are processed.

Notes:

1. The caller must clear register 0 unless using register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) with MCSFLAG=REG0. However, IBM recommends using the CONSID parameter rather than register 0.
2. If the list and execute forms of the WTOR macro are in separate modules, both modules must be assembled or compiled with the same level of WTOR.
3. If the execute form of the macro specifies CONSID, KEY, or TOKEN then the list form, to ensure that the parameter list is generated correctly, must specify the same parameter(s) without data. If data is specified, the system issues an MNOTE and ignores the data.
4. For any WTOR keywords that allow a register specification, the value must be right-justified in the register.

Example

Operation: Send a WTOR to a console whose ID is in register 4. This example assumes that the values in fields REPLY and REPECB have previously been set.

```
       USING   *,R12
*
*-------------------------------------------------------------------*
*      ISSUE A WTOR TO A CONSOLE WHOSE ID IS IN REGISTER 4.         *
*-------------------------------------------------------------------*
         WTOR  'USR902A REPLY YES OR NO TO CONTINUE.',REPLY,L10,REPECB,X
               CONSID=(R4)
         .
         .
         .
L10      EQU   10
REPLY    DS    CL10
REPECB   DS    F
         END
```

WTOR (List Form)

The list form of the WTOR macro is used to construct a control parameter list.

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

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

'msg',reply addr,reply length,ecb addr	*msg:* Up to 122 characters. *reply addr:* A-type address, or register (2) - (12). *reply length:* symbol, decimal number, or register (2) - (12). The minimum length is 1; the maximum length is 119. *ecb addr:* A-type address, or register (2) - (12). **Notes:** 1. *'msg',reply addr,reply length,ecb addr* must be the first parameter you code. 2. If you do not code *reply addr* on the list form of WTOR, mark its position with a comma, and code *reply addr* on the execute form. The same is true for *reply length* and *ecb addr*.
,ROUTCDE=*(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,MSGTYP=*(msg type)*	*msg type:* any of the following: N, Y, SESS, JOBNAMES, STATUS, SESS,JOBNAMES, SESS,STATUS, JOBNAMES,STATUS, SESS,JOBNAMES,STATUS
,MCSFLAG=*(flag name)*	*flag name:* any combination of the following, separated by commas: REG0, RESP, REPLY, BRDCST, CMD, HRDCPY, QREG0, NOTIME, NOCPY
,CONSID=	Parameter value not required for list form. Code only ,CONSID=. If you code CONSID on the list form of WTOR, you must code CONSID on the execute form.
,KEY=	Parameter value not required for list form. Code only ,KEY=. If you code KEY on the list form of WTOR, you must code KEY on the execute form.
,TOKEN=	Parameter value not required for list form. Code only ,TOKEN=. If you code TOKEN on the list form of WTOR, you must code TOKEN on the execute form.
,MF=L	

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

,MF=L
specifies the list form of the WTOR macro.

WTOR (Execute Form)

The execute form of the WTOR macro uses a remote control parameter list. The parameter list can be generated by the list form of WTOR. The message cannot be modified on the execute form of the macro.

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

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

reply addr,reply length,ecb addr	*reply addr:* A-type address, or register (2) - (12). *reply length:* symbol, decimal number, or register (2) - (12). The minimum length is 1; the maximum length is 119. *ecb addr:* A-type address, or register (2) - (12). **Notes:** 1. *reply addr,reply length,ecb addr* must be the first parameter you code. 2. If you do not code *reply addr* on the execute form of WTOR, mark its position with a comma, and code *reply addr* on the list form. The same is true for *reply length* and *ecb addr*.
,CONSID=*console id*	*console id:* RX-type address or register (2) - (12) If you code CONSID on the execute form of WTOR, you must code CONSID on the list form.
,KEY=*key*	*key:* RX-type address or register (2) - (12) If you code KEY on the execute form of WTOR, you must code KEY on the list form.
,TOKEN=*token*	*token:* RX-type address or register (2) - (12) If you code TOKEN on the execute form of WTOR, you must code TOKEN on the list form.
,MF=(E,*ctrl addr*) ,MF=(E,*ctrl addr*,EXTENDED)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

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

,MF=(E,*ctrl addr*)
,MF=(E,*ctrl addr*,EXTENDED)
specifies the execute form of the WTOR macro. *ctrl addr* defines the area into which the system stores the parameter list.

If you specify *reply addr*, *reply length*, or *ecb addr* on the execute form of WTOR, together with any of the following parameters, you must specify EXTENDED for the system to generate the parameter list correctly:

KEY
TOKEN
CONSID

Example

Operation: Using the list and execute forms of the WTOR macro, send a message to the console whose ID was previously defined in field CONSVAL. This example assumes that fields REPECB and REPAREA have also been previously set.

```
L10      EQU   10
         USING  *,R12
         WTOR  ,,,,MF=(E,MSGLIST),CONSID=CONSVAL
         .
         .
         .
MSGLIST  WTOR  'USR456A CONFLICTING VALUES SPECIFIED, SECOND VALUE IGNOX
               RED, PLEASE CONFIRM (YES OR NO)',REPAREA,L10,REPECB,    X
               CONSID=,MF=L
CONSVAL  DS    F
REPECB   DS    F
REPAREA  DS    CL10
         END
```

Appendix A. List of the Names of Macros Intended for Customers Use

The macros identified in this appendix are provided to allow a customer installation to write programs that use the services of MVS. Only those macros identified in this appendix should be used to request or receive the services of MVS.

Some macros are listed as both general-use and product-sensitive programming interfaces. These macros have general-use and product-sensitive keywords, fields, or parameters.

General-Use Programming Interfaces

The macros listed in this topic are general-use programming interfaces intended for customer use. Some macros have keywords, fields, or parameters that are designed for IBM internal use only. Such keywords, fields, or parameters are not part of the programming interfaces for use by customers in writing programs that request or receive the services of MVS. Please refer to the appropriate MVS product documentation for the correct classification and use of keywords, fields, and parameters for macros.

Executable Macros

This section lists the executable macros that are general-use programming interfaces for the control program. The standard, execute, list, and modify forms of the macros are programming interfaces.

Figure 30 (Page 1 of 3). General-Use Executable Macros

Name
ABEND
ALESERV
ASCRE
ASDES
ASEXT
ATSET
ATTACH
ATTACHX
AXEXT
AXFRE
AXRES
AXSET
BLSABDPL
BLSACBSP
BLSADSY
BLSAPCQE
BLSQFXL
BLSQMDEF
BLSQMFLD
BLSQSHDR
BLSRDRPX
BLSRESSY
BLSRNAMP
BLSRPRD
BLSRPWHS
BLSRSASY
BLSRXMSP
BLSRXSSP
BLSUPPR2
CALL
CALLDISP
CALLRTM
CALLTSSR
CHANGKEY
CHAP
CIRB
CMDAUTH
COFCREAT
COFDEFIN
COFIDENT
COFNOTIF
COFPURGE
COFREMOV
COFRETRI

Figure 30 (Page 2 of 3). General-Use Executable Macros

Name
COFSDONO
CPOOL
CPUTIMER
CSRCESRV
CSVQUERY
CTRACE
DATOFF
DELETE
DEQ
DETACH
DIV
DOM
DSGNL
DSPSERV
DYNALLOC
ENQ
ESPIE
ESTAE
ESTAEX
ETCON
ETCRE
ETDEF
ETDES
ETDIS
EVENTS
EXCP
EXCPVR
EXTRACT
FESTAE
FRACHECK
FREEMAIN
GETLINE
GETMAIN
GQSCAN
GTRACE
HSPSERV
IDENTIFY
IEFQMREQ
IEFSSREQ
IKJRLSA
IOSINFO
IOSLOOK
ITTFMTB
LINK
LINKX
LLACOPY
LOAD
LOCASCB
LSEXPAND
LXFRE
LXRES
MGCR
MODESET
NIL
NUCLKUP
OIL
OUTADD
OUTDEL
PCLINK
PGANY
PGFIX
PGFIXA
PGFREE
PGFREEA
PGLOAD
PGOUT
PGRLSE
PGSER
POST
PTRACE
PURGE
PURGEDQ
PUTGET
PUTLINE
QEDIT
RACDEF
RACHECK

Figure 30 (Page 3 of 3). General-Use Executable Macros

Name
RACINIT
RACLIST
RACROUTE
RACSTAT
RACXTRT
RESERVE
RESMGR
RESTORE
RESUME
RETURN
RISGNL
SAVE
SCHEDULE
SCHEDXIT
SDUMP
SDUMPX
SETFRR
SETLOCK
SETRP
SMFCHSUB
SMFDETAL
SMFEWTM
SMFEXIT
SMFINTVL
SMFRTEST
SMFSUBP
SMFWTM
SNAP
SNAPX
SPIE
SPLEVEL
SPOST
SRBSTAT
SRBTIMER
SSAFF
STACK
STAE
STATUS
STAX
STIMER
STIMERM
STORAGE
SUSPEND
SVCUPDTE
SWAREQ
SYMREC
SYNCH
SYNCHX
SYSEVENT
SYSSTATE
TCBTOKEN
TCTL
TESTART
TESTAUTH
TIME
TIMEUSED
TTIMER
T6EXIT
UCBDEVN
VRADATA
VSMLIST
VSMLOC
VSMREGN
WAIT
WTL
WTO
WTOR
XCTL
XCTLX

Mapping Macros

This section lists the mapping macros that are general-use programming interfaces for the control program. The data areas are programming interfaces or contain fields that are programming interfaces.

Figure 31 (Page 1 of 3). General-Use Mapping Macros

Macro ID	Name
ADSR	ADSR
AHLFFAP	FFAP
AHLWKAL	WKAL
BLSABDPL	BLSABDPL
BLSACBSP	BLSACBSP
BLSADSY	BLSADSY
BLSAPCQE	BLSAPCQE
BLSQFXL	BLSQFXL
BLSRDATC	BLSRDATC
BLSRDATS	BLSRDATS
BLSRDATT	BLSRDATT
BLSRDRPX	BLSRDRPX
BLSRESSY	BLSRESSY
BLSRNAMP	BLSRNAMP
BLSRPRD	BLSRPRD
BLSRPWHS	BLSRPWHS
BLSRSASY	BLSRSASY
BLSRXMSP	BLSRXMSP
BLSRXSSP	BLSRXSSP
BLSUPPR2	BLSUPPR2
CVT	CVT
ICHPCGRP	CGRP
ICHPISP	ISP
ICHPRCVT	RCVT
ICHRRPF	RRPF
ICHRUTKN	RUTKN
ICHSAFP	SAFP
IEAVVTPC	TPC
IECDIOCM	IOCOM
IEESMCA	SMCA
IEFAJCTB	JCT
IEFASCTB	SCT
IEFDOKEY	IEFDOKEY
IEFDORC	IEFDORC
IEFDOTUM	IEFDOTUM
IEFJESCT	JESCT
IEFJFCBN	JFCB
IEFJFCBX	JFCBX
IEFJSBVT	--
IEFJSCVT	SSCVT
IEFJSIPL	JSIPL
IEFJSSIB	SSIB
IEFJSSVT	SSVT
IEFQMIDS	QMIDS
IEFQMNGR	QMPA
IEFSSOBH	SSOB
IEFSSSM	SSSM
IEFSSVS	SSVS
IEFTIOT1	TIOT
IEFUCBOB	UCB
IEFVTSPL	VTSPL
IEFZB476	EMPARMS
IEFZB4D0	S99PARMS
IEFZB4D2	IEFZB4D2
IEFZB505	EPAL
IEFZB506	EPAM
IEZCIB	CIB
IEZCOM	COM
IEZEAECB	EAECB
IEZJSCB	JSCB
IEZMGCR	MGCRPL
IEZWPL	WPL
IFASMFR	--
IGVVSMD	VSMD
IHAABDPL	ABDPL
IHAACEE	ACEE
IHAASCB	ASCB
IHAASEO	ASEO
IHAASSB	ASSB
IHAASXB	ASXB

Figure 31 (Page 2 of 3). General-Use Mapping Macros

Macro ID	Name
IHADSAB	DSAB
IHAECB	ECB
IHAEPIE	EPIE
IHAETD	ETD0
IHAFRRS	FRRS
IHAIQE	IQE
IHAPCCA	PCCA
IHAPCCAT	PCCAVT
IHAPICA	PICA
IHAPIE	PIE
IHAPSA	PSA
IHAPSL	PSL
IHAPVT	PVT
IHARB	RB
IHARMPL	RMPL
IHASCB	SCB
IHASDST	SDST
IHASDWA	SDWA
IHASMDLR	SMDLR
IHASRB	SRB
IHASSL	SSL
IHASVT	SVT
IHATQE	TQE
IHATRBPL	TRBP
IHATREPL	TREP
IHATROB	TROB
IHATRVT	TRVT
IHATTE	TTE
IHAVFPM	VFPM
IHAVRA	VRAMAP
IHAVSL	VSL
IKJCPPL	CPPL
IKJCSOA	CSOA
IKJCSPL	CSPL
IKJDACB	DAIRACB
IKJDAPL	DAPL
IKJDAP00	--
IKJDAP04	--
IKJDAP08	--
IKJDAP0C	--
IKJDAP10	--
IKJDAP14	--
IKJDAP18	--
IKJDAP24	--
IKJDAP28	--
IKJDAP2C	--
IKJDAP30	--
IKJDAP34	--
IKJECT	ECT
IKJEFFDF	--
IKJEFFGF	--
IKJEFFMT	--
IKJENDP	--
IKJGTPB	GTPB
IKJIDENT	--
IKJIOPL	IOPL
IKJKEYWD	--
IKJLSD	LSD
IKJNAME	--
IKJOPER	--
IKJPARM	--
IKJPGPB	PGPB
IKJPOSIT	--
IKJPPL	PPL
IKJPSCB	PSCB
IKJPTPB	PTPB
IKJRSVWD	--
IKJSTPB	STPB
IKJSUBF	--
IKJTAIE	TAIE
IKJTCB	TCB
IKJTERM	--
IKJTSMSG	--
IKJUPT	UPT
ISGRIB	RIB - RIBE
ITTCTE	ITTCTE
ITTCTSS	CTSS

Figure 31 (Page 3 of 3). General-Use Mapping Macros

Macro ID	Name
ITTCTXI	CTXI
MCHEAD	MCHEAD

Product-Sensitive Programming Interfaces

The macros listed in this topic are product-sensitive programming interfaces intended for customer use. Some macros have keywords, fields, or parameters that are designed for IBM internal use only. Such keywords, fields, or parameters are not part of the programming interfaces for use by customers in writing programs that request or receive the services of MVS. Please refer to the appropriate MVS product documentation for the correct classification and use of keywords, fields, and parameters for macros.

Executable Macros

This section lists the executable macros that are product-sensitive programming interfaces for the control program. The standard, execute, list, and modify forms of the macros are programming interfaces.

Figure 32. Product-Sensitive Executable Macros

Name
CMDAUTH
DOM
RACDEF
RACHECK
RACINIT
RACROUTE
RACXTRT

Mapping Macros

This section lists the mapping macros that are product-sensitive programming interfaces for the control program. The data areas are programming interfaces or contain fields that are programming interfaces.

Figure 33 (Page 1 of 2). Product-Sensitive Mapping Macros

Macro ID	Name
AMDDATA	AMDDATA
COFZCXIT	CXT
CVT	CVT
IARRAX	RAX
IARRCE	RCE
IEECUCM	UCM
IEFALLCT	LCT
IEFDOCNP	DOCNP
IEFJICA	JICA
IEFJMR	JMR
IEFNEL	NEL
IEFSSCF	SSCF
IEFSSCI	SSCI
IEFSSCM	SSCM
IEFSSDM	SSDM
IEFSSEN	SSEN
IEFSSET	SSET
IEFSSWT	SSWT
IEFTCT	TCT
IEFUCBOB	UCB
IEZVX100	CTXT
IHAABEPL	ABEP
IHAASCB	ASCB
IHAASVT	ASVT
IHAASXB	ASXB
IHACDE	CDE
IHACSD	CSD
IHADOMC	DOMC
IHAGDA	GDA
IHALLE	LLE
IHALLP1	LLP1
IHALLP2	LLP2

Figure 33 (Page 2 of 2). Product-Sensitive Mapping Macros

Macro ID	Name
IHAORE	ORE
IHARB	RB
IHASDEPL	SDEPL
IHAWQE	WQE
IHAXTLST	XTLST
IKJTCB	TCB
IRAOUCB	OUCB
ISGGVT	GVT
ISGPEL	PEL
ISGRNLE	RNLE

Index

F

G

H

I

L

M

S

T

U

V

W

X

Readers' Comments

MVS/ESA
System Programming Library:
Application Development Macro Reference

MVS/System Product:
JES2 Version 3
JES3 Version 3

Publication No. GC28-1857-5

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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

Be sure to print your name and address below if you would like a reply, or provide your FAX telephone number if you would prefer a FAX response.

FAX (United States & Canada): 914+296-6496
FAX (Other countries): 001+914+296-6496

Name

Address

Company or Organization

Phone No.

IBM®

Cut or F
Along L

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO BOX 950
POUGHKEEPSIE NY 12602-9935

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alor

File Number: S370-36
Program Number: 5685-001
5685-002

Printed in U.S.A.

GC28-1857-5