

z/VM
7.3

CMS Macros and Functions Reference



Note:

Before you use this information and the product it supports, read the information in [“Notices” on page 565](#).

This edition applies to version 7, release 3 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-06-16

© **Copyright International Business Machines Corporation 1991, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures.....	ix
Tables.....	xi
About This Document.....	xiii
Intended Audience.....	xiii
Syntax, Message, and Response Conventions.....	xiii
Where to Find More Information.....	xvi
Links to Other Documents and Websites.....	xvi
How to provide feedback to IBM.....	xvii
Summary of Changes for z/VM: CMS Macros and Functions Reference.....	xix
SC24-6262-73, z/VM 7.3 (June 2025).....	xix
SC24-6262-73, z/VM 7.3 (December 2023).....	xix
SC24-6262-73, z/VM 7.3 (September 2022).....	xix
Part 1. CMS Preferred Programming Interface.....	1
Chapter 1. CMS Programming Interface.....	3
CMS Interface Groups.....	4
CMS Preferred Interface.....	5
CMS Preferred Macros.....	5
CMS Preferred Routines.....	9
CMS Preferred Functions.....	9
CMS Compatibility Interface.....	10
CMS Compatibility Macros and Suggested Replacements.....	10
CMS Functions and Suggested Replacements.....	10
Simulated OS/MVS Macros.....	11
OS/MVS Macros for Assembly Only.....	14
Chapter 2. Preferred CMS Macro Instructions.....	15
CMS Macro Coding Conventions.....	15
CMS Macro Formats.....	15
Using the Online HELP Facility.....	16
ABNEXIT.....	18
AMODESW.....	23
AMODESW CALL.....	24
AMODESW QRY.....	26
AMODESW RETURN.....	27
AMODESW SET.....	29
ANCHOR.....	31
APPLMSG.....	34
BATLIMIT.....	46
CMSCALL.....	47
CMSCVT.....	54
CMSDEV.....	56
CMSECVT.....	60
CMSIUCV.....	61

CMSIUCV ACCEPT.....	62
CMSIUCV CONNECT.....	67
CMSIUCV QCMSWID.....	74
CMSIUCV RESOLVE.....	77
CMSIUCV SEVER.....	80
CMSLEVEL.....	84
CMSRET.....	86
CMSSTACK.....	87
CMSSTOR.....	90
CMSSTOR OBTAIN.....	91
CMSSTOR RELEASE.....	100
COMPSWT.....	106
CONSOLE.....	107
CONSOLE CLOSE.....	108
CONSOLE EXCP.....	110
CONSOLE MODIFY.....	113
CONSOLE OPEN.....	116
CONSOLE QUERY.....	121
CONSOLE READ.....	125
CONSOLE WAIT.....	129
CONSOLE WRITE.....	132
CQYSECT.....	138
CSFCB.....	140
CSLENTY.....	141
CSLEXIT.....	144
CSLFPI.....	146
CSLGETP.....	150
CSRCMPSC.....	152
CSRYCMPD.....	155
CSRYCMPS.....	160
DIRBUFF.....	163
DMSABEXP.....	168
DMSABN.....	169
DMSFST.....	170
DMSJNEPL.....	171
DMSQEFL.....	172
DMSSDWA.....	173
DMSSTATE.....	176
ENABLE.....	178
EPLIST.....	180
EXITBUFF.....	182
EXSBUFF.....	185
EXTUAREA.....	188
EXTXCTL.....	189
FPERROR.....	190
FSCB.....	191
FSCBD.....	194
FSCLOSE.....	196
FSERASE.....	199
FSOPEN.....	202
FSPOINT.....	213
FSREAD.....	217
FSSTATE.....	224
FSTD.....	230
FSWRITE.....	232
GETSID.....	239
HNDEXT.....	242
HNDINT.....	247

HNDIO.....	250
HNDIUCV.....	257
HNDIUCV CLR (Clear).....	258
HNDIUCV HLD (Hold).....	261
HNDIUCV REP (Replace).....	263
HNDIUCV RES (Resume).....	267
HNDIUCV SET.....	269
HNDSVC.....	273
HSVCSAVE.....	277
IMMBLOK.....	278
IMMCMD.....	279
INTBLOK.....	283
LABSECT.....	285
LANGBLK.....	287
LINERD.....	289
LINEWRT.....	301
LRDD.....	308
LWRD.....	310
NUCEXT.....	317
NUCEXT ANCHOR.....	318
NUCEXT CLR.....	320
NUCEXT QUERY.....	322
NUCEXT RENAME.....	324
NUCEXT SET.....	326
NUCON.....	333
PARSECMD.....	334
PARSERCB.....	340
PARSERUF.....	342
PRINTL.....	343
PUNCHC.....	350
PVCENTRY.....	352
RDCARD.....	354
RDTAPE.....	357
REGEQU.....	362
REXEXIT.....	363
RXITDEF.....	369
RXITPARM.....	370
SCAN.....	372
SCBLOCK.....	376
SEGMENT.....	378
SGMTEXIT.....	386
SHVBLOCK.....	387
SUBCOM.....	389
SUBCOM ANCHOR.....	390
SUBCOM CLR.....	392
SUBCOM QUERY.....	394
SUBCOM SET.....	396
SUBPOOL.....	401
SUBPOOL CREATE.....	402
SUBPOOL DELETE and RELEASE.....	406
TAPECTL.....	410
TAPESL.....	418
TRANTBL.....	423
TVISECT.....	424
USERSAVE.....	427
USERSECT.....	428
VOLSECT.....	429
WAITD.....	430

WAITECB.....	432
WAITT.....	435
WRTAPE.....	436
WUERROR.....	442
Chapter 3. CMS Preferred Functions.....	443
DISKID.....	444
DMSSEQ.....	446
LANGADD.....	447
LANGFIND.....	449
Part 2. Compatibility Programming Interface.....	451
Chapter 4. CMS Compatibility Macros.....	453
CMSCB.....	454
DISPW.....	459
DMSEXS.....	460
DMSFREE.....	461
DMSFRES.....	463
DMSFRET.....	464
DMSKEY.....	466
IO.....	468
LINEDIT.....	472
RDTERM.....	482
STRINIT.....	485
TEOVEXIT.....	486
WRTERM.....	490
Chapter 5. CMS Compatibility Functions.....	493
ATTN.....	494
NUCEXT.....	495
SUBCOM.....	500
TODACCNT.....	503
WAITRD.....	505
Appendix A. Simplified RACROUTE Macro Functions.....	509
External Interfaces Supported for REXX Callers.....	509
IBM-Provided Binding Files.....	510
IBM-Provided REXX EXECs.....	510
External Interfaces supported for REXX, Assembler, and C Callers.....	513
Testing Whether a Class is Active with DMSWSESM	513
Creating an Audit Log Entry with DMSWSAUD	515
Testing a User's Authority to Access a Resource with DMSWSAUT	517
Calling Using the IBM-Provided REXX EXECs.....	519
Calling Without Using the IBM-Provided REXX EXECs.....	522
Appendix B. VSE Macros.....	527
VSE Assembler Language Macros Supported.....	527
VSE Supervisor and I/O Macros Supported by CMS/DOS.....	529
Appendix C. CRR Participation Macros.....	545
ADAPTRC.....	546
Appendix D. NETDATA Format.....	551
Exception.....	552
Control Record Formats.....	552
Text Units.....	552

Dates and Times.....	553
Numeric Values.....	553
Text Unit Keys.....	553
File Block Size.....	554
File Name.....	555
File Organization.....	555
Receive Results.....	556
Receipt Request.....	556
File Mode.....	556
Node of Originator.....	557
Time of Transmission.....	557
User ID of Originator.....	557
Date of Last Change.....	557
Logical Record Length.....	557
Number of Files.....	558
Record Format.....	558
File Size.....	559
Note File.....	559
Target Node.....	559
Target User ID.....	559
Program Name.....	559
Header Record (INMR01).....	560
File Utility Control Record (INMR02).....	561
Data Control Record (INMR03).....	562
User Control Record (INMR04).....	563
Trailer Control Record (INMR06).....	563
Acknowledgement Control Record (INMR07).....	563
Notices.....	565
Programming Interface Information.....	566
Trademarks.....	566
Terms and Conditions for Product Documentation.....	566
IBM Online Privacy Statement.....	567
Bibliography.....	569
Where to Get z/VM Information.....	569
z/VM Base Library.....	569
z/VM Facilities and Features.....	570
Prerequisite Products.....	572
Related Products.....	572
ANCHOR Identifier Registration Form.....	573
Index.....	575

Figures

1. NETDATA File Format..... 551

2. Data and Control Record Formats..... 551

3. INMR02 Control Record Format..... 552

4. Text Units.....552

Tables

1. Examples of Syntax Diagram Conventions.....	xiii
2. Comparison of CMS Virtual Machine Architectures.....	4
3. CMS Preferred Macros.....	6
4. CMS Preferred Functions.....	9
5. CMS Compatibility Macros and Suggested Replacements.....	10
6. CMS Functions and Suggested Replacements.....	10
7. Simulated OS/MVS Macros.....	12
8. CMSCALL Call Chart.....	51
9. SVC 202 Call Chart.....	51
10. PSW Settings When a Called Routine Starts.....	52
11. Register Contents When a Called Routine Starts.....	52
12. ABEND Codes Specific to CMSCALL.....	53
13. Releasing Storage Allocation.....	104
14. CSRYCMPD Macro.....	157
15. CMPSCDICT_SD DSECT.....	158
16. CMPSCDICT_SDE DSECT.....	158
17. CMPSCDICT_UE DSECT.....	158
18. CMPSCDICT_PE DSECT.....	159
19. Summary of Interrupt Types Affected by ENABLE INTTYPE Options.....	179
20. General Data Buffer Fields.....	183
21. Virtual Printer Maximum Data Bytes.....	347
22. Equate statements generated by REGEQU.....	362
23. TVIFUNCT keyword meanings.....	425

24. TVIRFMT byte meanings.....	426
25. TODACCNT 16-byte timing information.....	503
26. Service Module Results.....	521
27. VSE Macros Supported by CMS.....	527
28. Physical IOCS Macros Supported by CMS/DOS.....	529
29. SVC Support Routines and Their Operation.....	529
30. CMS/DOS Support of DTFCN Macro.....	537
31. CMS/DOS Support of DTFCN macro.....	538
32. CMS/DOS Support of DTFDI Macro.....	539
33. CMS/DOS Support of DTFMT Macro.....	540
34. CMS/DOS Support of DTFPR Macro.....	541
35. CMS/DOS Support of DTFSD Macro.....	542

About This Document

This document provides information to enable you to:

- Use CMS macroinstructions when you write programs to run in the CMS environment.
- Use the CMSIUCV and HNDIUCV assembler language macros to start or end communications with another program in an Inter-User Communications Vehicle (IUCV) or Advanced Program-to-Program Communication/VM (APPC/VM) environment.
- Run CMS functions from programs.

Intended Audience

This information is for application programmers and system programmers who work with IBM® assembler language and want to use z/VM® CMS macros and functions.

You should be knowledgeable about assembler language programming or have at least a two-year programming certificate.





Syntax, Message, and Response Conventions

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The  symbol indicates the beginning of the syntax diagram.
- The  symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.
- The  symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.
- The  symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in [Table 1 on page xiii](#).



Table 1. Examples of Syntax Diagram Conventions	
Syntax Diagram Convention	Example
Keywords and Constants A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown. In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.	 KEYWORD 

Table 1. Examples of Syntax Diagram Conventions (continued)

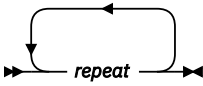
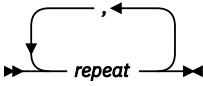
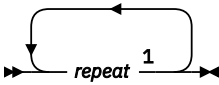
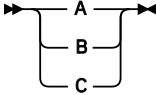
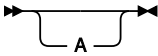
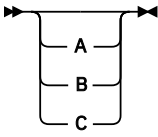
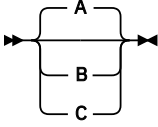
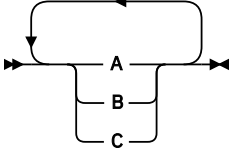
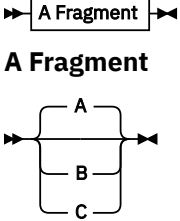
Syntax Diagram Convention	Example
Abbreviations <p>Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated.</p> <p>In this example, you can specify KEYWO, KEYWOR, or KEYWORD.</p>	<p>►► KEYWOrd ◄◄</p>
Symbols <p>You must specify these symbols exactly as they appear in the syntax diagram.</p>	<p>* Asterisk</p> <p>: Colon</p> <p>, Comma</p> <p>= Equal Sign</p> <p>- Hyphen</p> <p>() Parentheses</p> <p>. Period</p>
Variables <p>A variable appears in highlighted lowercase, usually italics.</p> <p>In this example, <i>var_name</i> represents a variable that you must specify following KEYWORD.</p>	<p>►► KEYWOrd — <i>var_name</i> ◄◄</p>
Repetitions <p>An arrow returning to the left means that the item can be repeated.</p> <p>A character within the arrow means that you must separate each repetition of the item with that character.</p> <p>A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated.</p> <p>Syntax notes may also be used to explain other special aspects of the syntax.</p>	<p>  </p> <p>  </p> <p>  </p> <p>Notes: ¹ Specify <i>repeat</i> up to 5 times.</p>
Required Item or Choice <p>When an item is on the line, it is required. In this example, you must specify A.</p> <p>When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.</p>	<p>►► A ◄◄</p> <p>  </p>

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	Example
<p>Optional Item or Choice</p> <p>When an item is below the line, it is optional. In this example, you can choose A or nothing at all.</p> <p>When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.</p>	 
<p>Defaults</p> <p>When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line.</p> <p>In this example, A is the default. You can override A by choosing B or C.</p>	
<p>Repeatable Choice</p> <p>A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item.</p> <p>In this example, you can choose any combination of A, B, or C.</p>	
<p>Syntax Fragment</p> <p>Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name.</p> <p>In this example, the fragment is named "A Fragment."</p>	

Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

- xxx** Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.
- []** Brackets enclose optional text that might be displayed.
- { }** Braces enclose alternative versions of text, one of which will be displayed.
- |** The vertical bar separates items within brackets or braces.
- ...** The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

Where to Find More Information

For information about related publications, see the [“Bibliography” on page 569](#).

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See [How to send feedback to IBM](#) for additional information.

Summary of Changes for z/VM: CMS Macros and Functions Reference

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

SC24-6262-73, z/VM 7.3 (June 2025)

This edition includes changes that are provided or announced after the general availability of z/VM 7.3.

SC24-6262-73, z/VM 7.3 (December 2023)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 7.3.

[VM66724] CMS Tape Block Size Increase

With the PTF for APAR VM66724, z/VM 7.3 increases the block size supported by CMS native tape I/O functions from 65,535 (64K-1) bytes to approximately 1 megabyte (1,114,094 bytes).

The following CMS macros are updated:

- “[RDTAPE](#)” on [page 357](#)
- “[WRTAPE](#)” on [page 436](#)

SC24-6262-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

Part 1. CMS Preferred Programming Interface

This section of the book introduces the CMS programming interfaces and defines the macros and functions that make up the CMS preferred interface. It contains the following chapters:

- Chapter 1, “[CMS Programming Interface](#),” on [page 3](#) introduces and defines the CMS programming interface.
- Chapter 2, “[Preferred CMS Macro Instructions](#),” on [page 15](#) describes the CMS macros in the preferred interface group.
- Chapter 3, “[CMS Preferred Functions](#),” on [page 443](#) describes the CMS functions that are considered a part of the preferred interface group.

Chapter 1. CMS Programming Interface

This chapter introduces and defines the CMS programming interface. It describes the different interface groups, the intent of each group, and the facilities that make up the group.

The CMS programming interface is a way for you to get CMS to do work for you. It is made up of:

- CMS preferred interface group
- CMS compatibility group
- OS/MVS and DOS/VSE group.

To understand the concept of the CMS programming interface groups, you should first understand the virtual machine environments in which CMS runs.

z/VM provides two versions of CMS:

ESA/390 CMS (generally referred to simply as CMS)

CMS runs in the following virtual machine architectures:

ESA/390 (ESA or XA virtual machine)

An ESA virtual machine simulates IBM Enterprise Systems Architecture/390 (ESA/390), which is a superset of IBM Enterprise Systems Architecture/370 (ESA/370), which is a superset of IBM System/370 Extended Architecture (370-XA). The XA virtual machine designation is supported for compatibility; an XA virtual machine is functionally equivalent to an ESA virtual machine.

ESA/XC (XC virtual machine)

An XC virtual machine processes according to IBM Enterprise Systems Architecture/Extended Configuration (ESA/XC), which is an architecture unique to virtual machines. Although XC virtual machines run with dynamic address translation off, they can take advantage of a subset of dynamic address translation architectural features, and in particular, data spaces.

z/Architecture® CMS (z/CMS)

z/CMS runs in the following virtual machine architectures:

z/Architecture (ESA or XA virtual machine)

z/Architecture uses 31-bit addressing mode in an ESA, XA, or Z virtual machine. CMS programs can use z/Architecture instructions, including those that operate on 64-bit registers, while permitting existing ESA/390 architecture CMS programs to continue to function without change.

When z/CMS is IPLed in an ESA/390 (ESA or XA) virtual machine, z/CMS switches the virtual machine to z/Architecture mode and thereafter executes in z/Architecture mode.

z/XC (XC virtual machine)

A z/XC guest uses VM Data Spaces with z/Architecture in the same way that an ESA/XC guest uses VM Data Spaces with Enterprise Systems Architecture. CMS applications that run in z/Architecture can use multiple address spaces. z/CMS can use VM Data Spaces for accessing Shared File System (SFS) Directory Control (DIRCONTROL) directories. z/XC supports programs that employ z/Architecture instructions and registers (within the limits of z/CMS support) and programs that exploit data spaces in the same CMS session.

When z/CMS is IPLed in an XC virtual machine, z/CMS switches the virtual machine to z/XC mode and thereafter executes in z/XC mode.

Unless otherwise indicated, "CMS" means either version, and descriptions of CMS functions apply to both CMS and z/CMS. For information on the z/CMS specifications and how to use z/CMS, see [*z/VM: CMS Planning and Administration*](#).

The virtual machine mode is defined by using the MACHINE or GLOBALOPTS directory statement, the CP SET MACHINE command, or the Systems Management application programming interfaces.

Note: CP does not support System/370 architecture (370 mode) virtual machines. However, the 370 Accommodation Facility allows many CMS applications written for System/370 virtual machines to run in

ESA/390 and ESA/XC virtual machines. The CP level of the 370 Accommodation Facility is activated with the CP SET 370ACCOM command. The CMS level of the 370 Accommodation Facility is activated with the CMS SET CMS370AC command. In addition, although the 370 option of the GENMOD command is not supported, modules previously generated with the 370 option can be run in an ESA/390 or ESA/XC virtual machine by issuing the CMS SET GEN370 OFF command. For more information about the 370 Accommodation Facility, see *z/VM: CP Programming Services*. See *z/VM: CP Commands and Utilities Reference* for information on the SET 370ACCOM command and see *z/VM: CMS Commands and Utilities Reference* for information on the SET CMS370AC and SET GEN370 commands.

The relationships between the virtual machine modes are summarized in [Table 2 on page 4](#).

<i>Table 2. Comparison of CMS Virtual Machine Architectures</i>					
CMS Version	Virtual Machine Architecture Mode¹	Virtual Machine Architecture	Addressing Scheme	Addressable Primary Storage	Addressable Data Space²
CMS	ESA, XA ³	ESA/390	31-bit	2047 MB	2 GB per data space ⁴
CMS	XC	ESA/XC	31-bit and access registers	2047 MB	2 GB per data space
z/CMS	ESA, XA ³ , Z	z/Architecture ⁵	31-bit ⁶	2047 MB ⁷	2 GB per data space ⁴
z/CMS	XC	z/XC ⁸	31-bit ⁶ and access registers	2047 MB ⁷	2 GB per data space
Notes: <ol style="list-style-type: none"> 1. Architecture mode is set by using the SET MACHINE command and the MACHINE statement of the directory entry. 2. Multiple data spaces are possible. 3. The XA designation is supported for compatibility. An XA virtual machine is functionally equivalent to an ESA virtual machine. 4. Data spaces can be read but cannot be modified. Data spaces can be modified only in virtual machines that run in XC architecture mode. 5. When z/CMS is IPLed in an ESA/390 virtual machine, z/CMS switches the virtual machine to z/Architecture and thereafter executes in z/Architecture mode. 6. Although z/CMS does not exploit or explicitly support 64-bit addressing mode, programs running on z/CMS can enter 64-bit addressing mode. 7. Although z/CMS does not directly exploit storage above 2047 MB, z/CMS can be IPLed in a virtual machine with more than 2 GB of storage and allows programs to use storage above 2 GB. 8. When z/CMS is IPLed in an XC virtual machine, z/CMS switches the virtual machine to z/XC and thereafter executes in z/XC mode. 					

With the CMS preferred interface, you can use macros, callable services (routines), and functions to code an application that is architecture-independent. The application can run in ESA, XA, and XC virtual machines. Therefore, you can write and assemble applications that are portable across the architectures.

CMS Interface Groups

Understanding the content and purpose of each group within the programming interface can help you select the CMS facility that is most appropriate for your program.

1. **CMS preferred interface group**—These macros, routines, and functions make up the heart of the CMS programming interface. They provide you with a means of making program calls, managing storage, performing I/O, providing communications with other systems, handling interrupts, and processing

abends. They run in ESA, XA, and XC virtual machines and support 24-bit and 31-bit addressing. They help you avoid architecture-constrained facilities like I/O instructions, they reduce your need to reference CMS internal data areas and control blocks, and they make it easier for you to develop programs that are portable across architectures.

IBM encourages you to use the preferred interface when writing your CMS application programs.

“CMS Preferred Interface” on page 5 lists the macros, routines, and functions that make up the preferred interface and summarizes the services they perform. The macros and functions are described later in this book. For more information on the routines, see “CMS Preferred Routines” on page 9.

2. **CMS compatibility group**—These are macros, functions, and services that CMS maintains for compatibility with previous releases. Existing programs that use interfaces in the compatibility group can run in 24-bit addressing mode in an ESA, XA, or XC virtual machine. Compatibility group interfaces may cause unpredictable results in 31-bit addressing mode.

For new programs, IBM recommends that you use interfaces in the preferred group rather than interfaces in the compatibility group. “CMS Compatibility Interface” on page 10 lists the compatibility group interfaces and their suggested replacements.

3. **OS/MVS and DOS/VSE group**—These are macros also provided by the OS/MVS and DOS/VSE operating systems. CMS supports these macros to make it easier to run programs developed for OS/MVS or DOS/VSE on CMS. The OS/MVS and DOS/VSE group consists of the following sub-groups:
 - a. **Simulated OS/MVS macros**—CMS simulates the function of these OS/MVS macros so that you can use them in your programs. While these macros provide some portability between VM and OS/MVS systems, the CMS simulation of these macros is **not** necessarily the same as the current MVS™ support. CMS simulates only a selected subset of OS/MVS macros and, because of operational differences between VM and MVS, macros that are supported may work differently between the two systems. For more information on how to use the OS/MVS macros, see the OS/MVS publications.

For CMS application programs, IBM recommends that you use macros in the preferred group rather than OS/MVS macros. “Simulated OS/MVS Macros” on page 11 lists the simulated OS/MVS macros.

- b. **Nonsimulated OS/MVS macros**—You can use these macros to develop and compile programs for execution on MVS systems; however, because CMS does not simulate these macros, programs that use them will not run on CMS. “OS/MVS Macros for Assembly Only” on page 14 lists these macros.
 - c. **DOS/VSE macros**—These are DOS/VSE macros that CMS simulates. The CMS simulation of these macros is not necessarily the same as the current DOS/VSE support. For information on these macros see Appendix B, “VSE Macros,” on page 527.

For CMS application programs, IBM recommends that you use macros in the preferred group rather than DOS/VSE macros.

Note: CMS macros, control blocks, and functions that are not part of the defined programming interface are considered CMS internal interfaces. They should not be used by programs other than CMS.

CMS Preferred Interface

Table 3 on page 6 lists the assembler macros in the CMS preferred interface and describes what function each macro provides. All of the macros in the preferred interface support 24-bit and 31-bit addressing.

Table 4 on page 9 defines the CMS functions DISKID, DMSSEQ, LANGADD, and LANGFIND, which are considered part of the preferred interface group.

CMS Preferred Macros

Note: OpenExtensions™ macros are also considered part of the preferred interface. These macros, which provide mapping for OpenExtensions callable services, are not listed in Table 3 on page 6 or described in this book. For more information, see *z/VM: OpenExtensions Callable Services Reference*.

Table 3. CMS Preferred Macros	
Macro	Function
ABNEXIT	Sets or clearsabend exit routines.
AMODESW	Switches or sets a program's addressing mode (AMODE) and provides an architecture-independent replacement to assembler language linkage instructions (BAL, BALR, BSM, BASSM, BAS, BASR).
ANCHOR	Allows setting, querying, and clearing a fullword that can be used to save the address of a program's data between calls.
APPLMSG	Accesses and displays messages from a message repository file.
BATLIMIT	Is a table of processor, punch, and printer limits for CMS batch jobs.
CMSCALL	Calls other programs. Use it as a replacement for SVC 202.
CMSCVT	Communications vector table.
CMSDEV	Places information about device characteristics in a user-provided buffer.
CMSECVT	Extended communications vector table.
CMSIUCV	For IUCV: establish or end IUCV communications with another program or with CP. For APPC/VM: establish or end an APPC/VM conversation, resolve a symbolic destination name, or query a workunit associated with a conversation.
CMSLEVEL	Maps the value of the feature or licensed program returned by the QUERY CMSLEVEL command. Note: You can also use the DMSQEFL CSL routine to return information about the level of CMS to a program.
CMSRET	Program return mechanism. Use it with CMSCALL.
CMSSTACK	Places data on the program stack. Use it as a replacement for the ATTN function.
CMSSTOR	Obtains and releases free storage. Use it as a replacement for the DMSFREE and DMSFRET macros.
COMPSWT	Sets the compiler switch on or off.
CONSOLE	Performs full-screen I/O services.
CQYSECT	Maps console path and device information to the buffer a user specifies on the CONSOLE OPEN or CONSOLE QUERY macro.
CSFCB	Maps the data referenced by the fourth word in the Extended Plist for the CMS subcommand interface when inhibiting implicit recursion of execs.
CSLENTY	Provides the entry logic for a callable services library (CSL) routine.
CSLEXIT	Provides the exit logic for a CSL routine.
CSLFPI	Allows an application to invoke a CSL routine using a fast path.
CSLGETP	Allows a CSL routine to get information about passed parameters.
CSRCMPSC	Calls Data Compression Services to compress and expand data.
CSRYCMPD	Maps compression and expansion dictionary entries.
CSRYCMPSP	Maps the CBLOCK parameter list for calls to Data Compression Services.
DIRBUFF	Maps the records returned by a Get Directory request.
DMSABEXP	Used with the DCBabend exit to map the parameter list.

Table 3. CMS Preferred Macros (continued)	
Macro	Function
DMSABN	Abends a virtual machine.
DMSFST	Maps the file status table for a given file.
DMSJNEPL	Maps the parameter list used by the DMSJNE exit routine.
DMSSDWA	Maps the area pointed to by register 1 upon entry to an ABNEXIT routine.
DMSSTATE	Conditions preferred-group macros so that access-register mode toleration code is expanded at assembly time.
ENABLE	Enables and disables the PSW interrupt mask.
EPLIST	Maps the extended parameter list passed in register 0.
EXITBUFF	Generates a DSECT for the general data buffer that SFS provides for the File Space Usage and User Storage Group Full exits.
EXSBUFF	Maps the records returned by an Exist request for a file or a directory.
EXTUAREA	Contains external interrupt status information.
EXTXCTL	Resumes execution of code that was suspended by a X'2603' external interrupt after this interrupt occurred.
FPERROR	Maps the file pool extended error information returned in the <i>wuerror</i> parameter of CSL routines.
FSCB	Sets up a file system control block.
FSCBD	Maps the file system control block.
FSCLOSE	Closes a file.
FSERASE	Erases a file.
FSOPEN	Opens a file.
FSPOINT	Resets the write and/or read pointers for a file.
FSREAD	Reads a record from a file.
FSSTATE	Checks for an existing file.
FSTD	Maps the FST area.
FSWRITE	Writes a record into a disk file.
GETSID	Stores a device's subchannel identification number (SID) in register 1.
HNDEXT	Defines handler routines for external interrupts.
HNDINT	Defines handler routines for I/O interrupts.
HNDIO	Defines handler routines for I/O interrupts and returns device-related information.
HNDIUCV	Initializes or ends a program's IUCV or APPC/VM environment.
HND SVC	Defines handler routines for SVCs.
HSVCSAVE	Maps the save area passed to interrupt handlers defined by HND SVC.
IMMBLOK	Maps the immediate command name block.
IMMCMD	Declares, clears, and queries immediate commands.
INTBLOK	Maps the I/O information that the HNDIO macro returns.

Table 3. CMS Preferred Macros (continued)	
Macro	Function
LABSECT	Maps control block for tape label processing.
LANGBLK	Generates a language control block for an application.
LINERD	Reads a line of input from the terminal.
LINEWRT	Writes a line of output to the terminal.
LRDD	Used with the LINERD macro to map the LINERD descriptors for multiple inputs.
LWRD	Used with the LINEWRT macro to map the LINEWRT descriptors for multiple outputs.
NUCEXT	Declares, clears, and queries nucleus extensions.
NUCON	Generates a mapping of the ACMSCVT, ADEVTAB, AEXEC, NUCXFRES, and USERLVL fields of the NUCON macro. Note: These are the only fields in NUCON that are supported as programming interfaces.
PARSECMD	Parses command arguments.
PARSERCB	Generates a parser control block DSECT.
PARSERUF	Generates a mapping for the user token validation parameter function control block.
PRINTL	Prints one or more lines on the printer.
PUNCHC	Punches a card.
PVCENTRY	Maps the parser validation code table.
RDCARD	Reads a card from the reader.
RDTAPE	Reads a record from tape.
REGEQU	Generates symbolic register equates.
REXEXIT	Invokes and maintains a list of user specified global exits for REXX programs.
RXITDEF	Assigns correct values to the symbols used for the exit routine function and subfunction codes.
RXITPARM	Maps the parameter list used to pass information between the language processor and an exit routine.
SCAN	Creates tokenized and extended parameter lists from input data.
SCBLOCK	Maps the subcommand block.
SEGMENT	Manages saved segments and segment spaces.
SGMTEXIT	Maps the SGMTEXIT control block.
SHVBLOCK	Maps the shared variable block.
SUBCOM	Defines, clears, and queries subcommand environments.
SUBPOOL	Manages free storage subpools.
TAPECTL	Positions a tape.
TAPESL	Processes standard HDR1 and EOF1 tape labels.
TRANTBL	Generates a DSECT mapping of system translation tables.
TVISECT	Generates a DSECT mapping for a nucleus extension module named DMSTVI.

Table 3. CMS Preferred Macros (continued)	
Macro	Function
USERSAVE	Maps control block for call status information.
USERSECT	An 18-fullword scratch area for user-defined purposes.
VOLSECT	Maps control block for tape label processing - used when more than 16 volume IDs are specified by the user.
WAITD	Suspends program execution until the next interrupt occurs for the specified device.
WAITECB	Suspends program execution until the specified event or events occur.
WAITT	Suspends program execution until all pending terminal I/O has completed.
WRTAPE	Writes a record to tape.
WUERROR	Maps the work unit extended error information returned in the <i>wuerror</i> parameter of CSL routines.

CMS Preferred Routines

All of the routines in the CMS preferred interface group support 24-bit and 31-bit addressing. The routines are documented in the following books:

- *z/VM: CMS Callable Services Reference* describes routines that perform various general programming tasks, such as:
 - File pool and minidisk file I/O
 - File pool administration
 - Accessing REXX variables
 - Extract/Replace
 - Manipulating the CMS program stack
 - Resource recovery
 - Using VM data spaces
 - Error checking and debugging
- *z/VM: CMS Application Multitasking* describes routines that perform multitasking and related programming tasks.
- *z/VM: OpenExtensions Callable Services Reference* describes routines that manipulate OpenExtensions Byte File System (BFS) data.
- *z/VM: Systems Management Application Programming* describes routines that perform systems management functions for virtual systems (guests) in a z/VM environment.

CMS Preferred Functions

Table 4. CMS Preferred Functions	
Function	Description
DISKID	Obtains information on the physical organization of a reserved minidisk.
DMSSEQ	Counts the number of logical lines in the terminal input buffer.
LANGADD	Adds a LANGBLK to the language block chain.
LANGFIND	Gets the address of an application's language control block.

CMS Compatibility Interface

Table 5 on page 10 below and Table 6 on page 10 list the macros and functions that CMS supports for compatibility only. Replacements are suggested when applicable. Existing programs can continue to use compatibility interfaces in programs that do not support 31-bit addressing.

Note:

1. The interfaces in the compatibility group are documented only in this reference.
2. The SVC 202 instruction is also considered part of the compatibility group. The CMSCALL macro is its suggested replacement.
3. The DMSEXS and DMSKEY macros allow users to modify CMS internal data areas; their use is not encouraged.

CMS Compatibility Macros and Suggested Replacements

<i>Table 5. CMS Compatibility Macros and Suggested Replacements</i>	
Macro	Suggested Replacement
DISPW	Use the CONSOLE macro to perform full-screen I/O and the LINEWRT and LINERD macros to perform line mode I/O.
DMSEXS	None—its use is not encouraged.
DMSFREE	CMSSTOR macro
DMSFRES	No replacement required; CMS performs the function internally.
DMSFRET	CMSSTOR macro
DMSKEY	None—its use is not encouraged.
LINEDIT	APPLMSG macro
RDTERM	LINERD macro
SCAN system service	SCAN macro
STRINIT	By default, CMS treats the STRINIT macro as a no-op. If necessary, you can use the SET STORECLR command to retain GETMAIN storage until end-of-command and enable STRINIT. If possible, programs should use the CMSSTOR macro rather than GETMAIN to obtain free storage.
TEOVEXIT	None.
WRTERM	LINEWRT macro. Unlike WRTERM, the LINEWRT macro does not allow you to specify text on the macro call itself (you have to specify the text in a buffer). You can use the APPLMSG macro to specify text on the macro call and display it at a terminal.

TEOVEXIT is documented in [“TEOVEXIT” on page 486](#).

CMS Functions and Suggested Replacements

The following CMS functions are considered part of the compatibility group.

<i>Table 6. CMS Functions and Suggested Replacements</i>	
Function	Suggested Replacement
ATTN	CMSSTACK macro, or StackWrite routine
NUCEXT	NUCEXT macro

<i>Table 6. CMS Functions and Suggested Replacements (continued)</i>	
Function	Suggested Replacement
SUBCOM	SUBCOM macro
TODACCNT	None
WAITRD	LINERD macro, or StackRead routine

Simulated OS/MVS Macros

CMS simulates some programming interfaces defined by the MVS operating system. These MVS interfaces are documented in the appropriate MVS book.

Table 7 on page 12 lists the MVS/XA macros that CMS simulates. For more information on the supported parameters for each macro, usage notes, and considerations for using OS/MVS macros in CMS programs, see [z/VM: CMS Application Development Guide for Assembler](#).

¹ The DEVTYPE interface will not return valid track or cylinder details that can be used for DASD space calculations. It is intended only to give access to default device characteristics. If detailed real DASD device characteristics are needed, see CP DIAGNOSE code X'210' in [z/VM: CP Programming Services](#) or the CMS DEVTYPE command in [z/VM: CMS Commands and Utilities Reference](#).

Table 7. Simulated OS/MVS Macros	
Macro	Function
ABEND	Terminates processing with user-specified completion and reason codes.
ATTACH	Passes control to another program at a new task level.
BLDL	Builds a directory list for a partitioned data set.
BSP	Backs up a record on a tape or disk.
BUILDRCD	Causes a buffer pool and a record area to be constructed.
CALL	Transfers control to a control section at a specified entry.
CHAP	No-op.
CHECK	Verifies READ/WRITE completion.
CHKPT	No-op.
CLOSE	Completes and secures I/O processing on a DCB.
CLOSE TYPE=T	Temporarily closes and deactivates the file.
CNTRL	No-op.
DCB	Constructs a data control block.
DCBD	Generates a DSECT for a data control block.
DELETE	Deletes a loaded program.
DEQ	No-op.
DETACH	No-op.
DEVTYPE ¹	Obtains device-type physical characteristics.
ENQ	No-op.
ESPIE	Sets up handlers for program interrupts. The caller can be in either 24-bit or 31-bit addressing mode.
ESTAE	Sets up abend exit routines.
EXCP	Executes a channel program for graphic access method (GAM).
EXTRACT	No-op.
FEOV	Forces an EOV condition on a tape or DASD file.
FIND	Locates a member of a partitioned data set.
FREEBUF	Returns a buffer to the DCB buffer pool.
FREEDBUF	Releases a simulated BDAM buffer.
FREEMAIN	Releases user-acquired storage.
FREEPOOL	Releases the DCB buffer pool.
GET	Reads system-blocked data (QSAM).
GETBUF	Acquires DCB buffer storage.
GETMAIN	Acquires user storage.
GETPOOL	Constructs a buffer pool for a DCB.
IDENTIFY	Adds an entry name to a loaded program.
IHAEPIC	EPIE work area mapping macro.
IHASDWA	Mapping macro for the system diagnostic work area used in ESTAE.
IHAVRA	Mapping macro for the system diagnostic work area variable recording area.
LINK	Passes control to another program at the same task level and returns to the calling program.
LOAD	Reads a program into storage.

Table 7. Simulated OS/MVS Macros (continued)	
Macro	Function
NOTE	Manages data set positioning.
OPEN	Prepares a DCB for I/O processing.
OPEN TYPE=J	Prepares a DCB for I/O processing after an RDJFCB has been issued.
PGLOAD	No-op.
PGOUT	No-op.
PGRLSE	No-op.
PGSER	No-op.
POINT	Manages data set positioning.
POST	Signals event completion.
PUT	Writes system-blocked data (QSAM).
PUTX	Returns the updated record to the data set from which it was read.
RDJFCB	Obtains information from FILEDEF command about an OS/MVS data set.
READ	Reads a physical input record (BSAM, BDAM, BPAM).
RELSE	No-op.
RETURN	Returns from a called program.
SAVE	Saves program registers.
SETRP	Makes requests for recovery from an ESTAE/ESTAI exit.
SNAP	Dumps specified areas of storage.
SPIE	Sets up an exit to be given control under user selected program interrupts. The caller must be in 24-bit addressing mode.
SPLEVEL	Sets macro expansion.
STAE	Sets up an abend exit routine.
STAX	Sets or cancels user exit for terminal attention interrupts.
STIMER	Sets the timer interval and the timer exit routine.
STIMERM	Sets, tests, or cancels multiple timer intervals and the timer exit routines.
STOW	Updates partitioned dataset directories.
SYNADAF	Provides SYNAD analysis function.
SYNADRLS	Releases SYNADAF message and save areas.
SYSSTATE	Conditions preferred-group macros so that access-register mode toleration code is expanded at assembly time.
TCLEARQ	Clears terminal input queue.
TGET/TPUT	Reads or writes a terminal line.
TIME	Gets the time of day.
TTIMER	Tests or cancels the timer.
WAIT	Waits for one or more events.
WRITE	Writes a physical record (BSAM, BDAM, BPAM).
WTO/WTOR	Writes a message to the operator's terminal.
XCTL	Passes control to another program at the same task level and does not return to the calling program.
XDAP	Reads or writes direct access volumes.

OS/MVS Macros for Assembly Only

In addition to the OS/MVS macros that CMS simulates, CMS includes many nonsimulated OS/MVS macros. The macros are contained in OSMACRO1 MACLIB.

These macros, which are listed below, are for assembly purposes only. Because CMS does **not** simulate these macros you should not use them in programs you intend to run on CMS.

For more information on these macros, see *OS/390® MVS Assembly Service* or *DFSMS Access Service Macros*.

ATSET	LOCASCB	SCHEDULE
AXEXT	LXFRE	SDUMP
AXFRE	LXRES	SEGLD
AXRES	MGCR	SEGWT
AXSET	MODESET	SETFRR
BLSABDPL	NIL	SETL
BLSQMDEF	NUCLKUP	SETLOCK
BLSRESSY	OIL	SETPRT
BUILD	PCLINK	SPOST
CALLDISP	PDAB	SRBSTAT
CALLRTM	PDABD	SRBTIMER
CHANGKEY	PGANY	STATUS
CIRB	PGFIX	SUSPEND
CPOOL	PGFIXA	SVCUPDTE
CPUTIMER	PGFREEA	SYNCH
CVT	PROTPSA	SYSEVENT
DATOFF	PRTOV	TCTL
DOM	PTRACE	TESTAUTH
DSGNL	PURGEDQ	TRUNC
DYNALLOC	QEDIT	VRADATA
ECVT	RACDEF	VSMLIST
ESETL	RACHECK	VSMLOC
ETCON	RACINIT	VSMREGN
ETCRE	RACLIST	WTL
ETDES	RACROUTE	XLATE
ETDIS	RACSTAT	
EVENTS	RACXTRT	
FESTAE	RELEX	
FRACHECK	RESERVE	
GQSCAN	RESUME	
IOSINFO	RISGNL	
IOSLOOK	RPSGNL	

Chapter 2. Preferred CMS Macro Instructions

This chapter describes the formats of the preferred CMS assembler language macros. These macros run in ESA, XA, and XC virtual machines and they are all capable of supporting 31-bit addressing. For more information on the CMS programming interface, see the discussion in [Chapter 1, “CMS Programming Interface,”](#) on page 3.

IBM recommends that you use these macros when you write assembler language programs to run in the CMS environment. To assemble a program using any of these macros, you must issue the GLOBAL command specifying MACLIB DMSGPI. This macro library is usually located on the system disk.

Note: References made to floating-point registers in this book refer to floating-point registers 0, 2, 4, and 6 unless indicated otherwise.

CMS Macro Coding Conventions

Coding conventions for CMS macros are the same as those for all assembler language macros. The macro format descriptions show optional operands in the format:



indicating that if you are going to use this operand, it must be preceded by a comma (unless it is the first operand coded). If a macro statement overflows to a second line, you must use a continuation character in column 72.

Note: No blanks may appear between operands.

When a macro offers a choice of operands, one and only one of which must be specified, the operands are stacked one per line and shown below the line of the syntax diagram.

Many operands can be specified with an argument in the form of either an expression or a register containing a value. When this is the case, the macro expects a register designation to begin with a left parenthesis. Therefore, specifying an expression that starts with a left parenthesis will produce unpredictable results, just as specifying a register without parentheses would.

Incorrect coding of any macro may result in assembler errors and MNOTES. MNOTES are unnumbered responses that can result from executing system generation macroinstructions or service programs. They are documented in logic listings only.

Where applicable, the end of a macro description contains a list of the possible error conditions that may occur during the execution of the macro, and the associated return codes. These return codes are always placed in register 15. The macros that produce these return codes have ERROR= operands that allow you to specify the address of an error handling routine that can check for particular errors during macro processing. If an error occurs during macro processing and no error address is provided, execution continues with the next sequential instruction following the macro.

CMS Macro Formats

CMS provides four macro formats:

Standard Format

is specified by omitting the MF macro form parameter. The standard format of the macro generates a series of assembler statements that declares the parameters inline and then calls the specified function. If the value of any parameter involves a substitution, then the macro generates nonreentrant code. If an optional parameter is omitted, the standard format substitutes the default value. If a required parameter is omitted, an MNOTE is issued and the parameter list is not valid.

MF=(E,addr)

MF=(E,(reg))

specifies the **execute format** of the macro, which generates code to execute the specified function. While you can use the execute format to fill in or modify a parameter list, you cannot use it to generate a parameter list. Before you issue the execute format of a macro, you must use the list or complex list format of the macro to create a valid parameter list—a parameter list that contains all required parameters with no conflicting options or extraneous parameters from previous macro calls. In many cases, you must reinitialize the parameter list before each new invocation of the execute format.

MF=L

specifies the **list format** of the macro, which generates an inline parameter list, substitutes default values for optional parameters you omit, and reserves a space for required parameters you omit. (To reserve a space, the list form issues the appropriate DC instruction with the correct length and data type.) The list format does **not** generate code to execute the specified function. To generate code to execute the specified function, use the execute format of the macro.

When you use the list format of the macro, do not use parameter forms that imply register use or indirect reference.

MF=(L,addr,label)

MF=(L,(reg),label)

specifies the **complex list format**, which generates a parameter list at the specified address. The complex list format does **not** substitute default values for parameters that are not specified, nor does it generate code to execute the specified function.

Specify the address of the parameter list as an assembler program label or as general register 1-12, enclosed in parentheses, that contains the address. If you code the optional parameter *label*, it is equated to the size of the parameter list.

Note: Because the complex list format produces executable code to move data into the specified area, you must invoke it before you invoke the execute format of the macro.

Note:

1. All parameters are optional **except** when you specify the standard format.
2. The MF=L format reserves space in the parameter list for required parameters; however, if you omit a required parameter on the MF=L macro format, you must use the MF=(L,addr) or MF=(E,addr) formats to specify the required value before you execute the function.
3. If you use a combination of the MF=(E,addr) and MF=(L,addr) formats of a macro, make sure that you specify a valid combination of parameters. The MF=(E,addr) and MF=(L,addr) formats change only the parameters specified on the macro invocation; they do not supply default values for parameters you omit. (Note that the standard and MF=L formats **do** supply default values for optional parameters you omit.)
4. The Standard and MF=(E,addr) forms of the macro alter the contents of registers 1 and 15. The MF=(L,addr) form alters register 1.
5. Not every CMS macroinstruction is available in each of these formats. Each, however, is available in the standard format.
6. The MF=(E,addr) format requires that the parameter list be in nonshared storage. Even if no parameters are specified on the macro invocation, the parameter list may be modified.

Using the Online HELP Facility

You can receive online information about the macros described in this book using the z/VM HELP Facility. For example, to display a menu of macros, enter:

```
help macro menu
```

To display information about a specific macro (ANCHOR in this example), enter:

```
help macro anchor
```

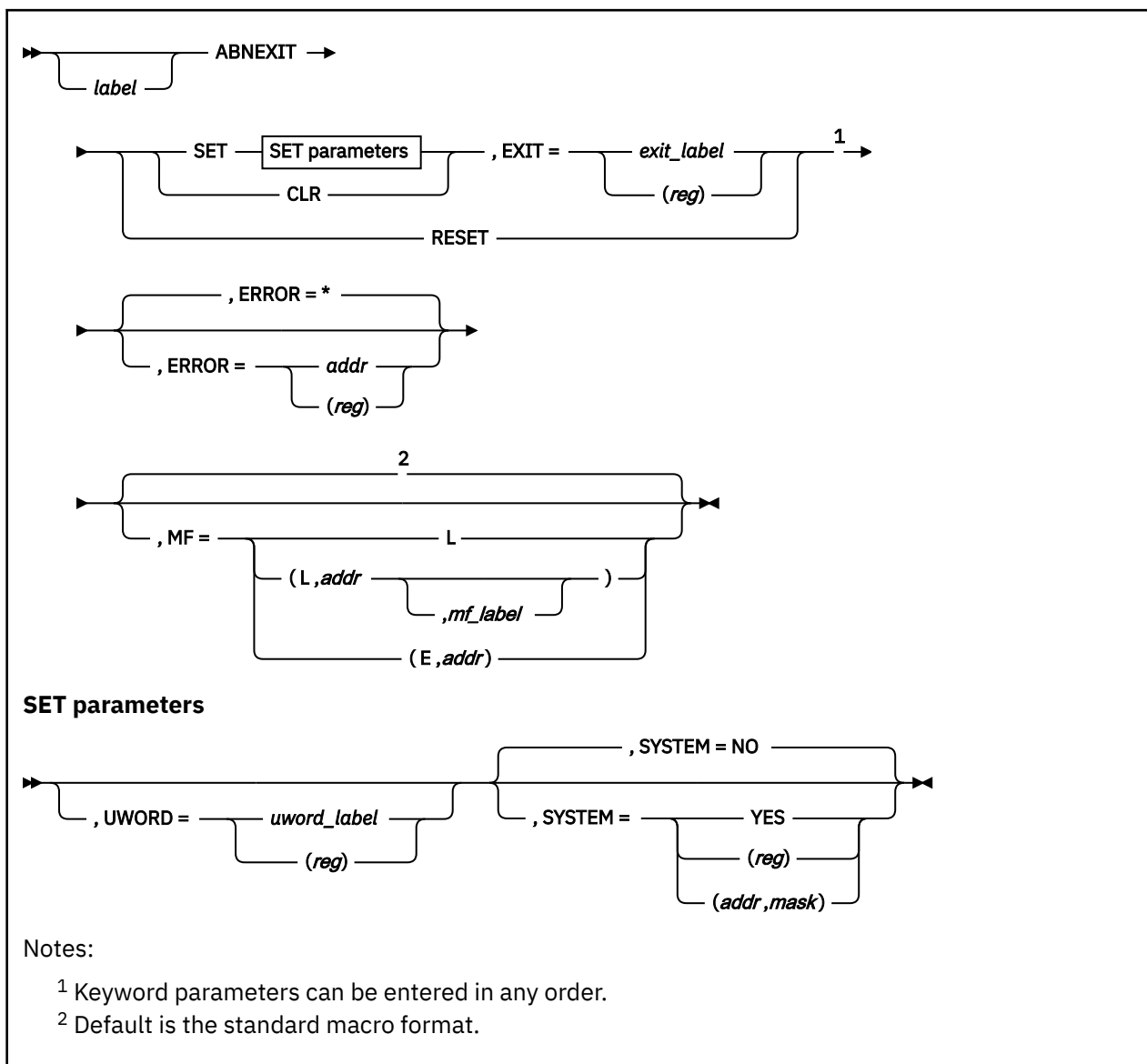
For more information about using the HELP Facility, see [z/VM: CMS User's Guide](#). To display the main HELP Task Menu, enter:

```
help
```

For more information about the HELP command, see [z/VM: CMS Commands and Utilities Reference](#) or enter:

```
help cms help
```

ABNEXIT



Purpose

Use the ABNEXIT macroinstruction to create (SET) or delete (CLR) an abend exit routine. The ABNEXIT RESET option allows your abend exit routine to be recalled if subsequent abends occur.

Parameters

Required Parameters:

SET

establishes an exit routine. This exit routine is added to the list of exit routines and becomes the current exit routine.

CLR

removes the specified exit routine from the list of exit routines. If it was the current exit routine, the previous exit routine on the list becomes the current exit routine. Exits can be cleared independently of their position in the list.

RESET

allows an abend exit routine to be recalled should a subsequent abend occur. In other words, CMS calls an abend exit routine only once **unless**, during its processing, the abend exit routine specifies ABNEXIT RESET. If the abend exit routine doesn't specify ABNEXIT RESET, CMS bypasses the abend exit routine should a subsequent abend occur.

Note: The RESET option can be specified from within an exit routine only.

EXIT=

specifies the address of the exit routine to be added or deleted. Acceptable values are:

exit_label

the assembler program label marks the address of the exit routine.

(reg)

the specified register contains the address of the exit routine.

Optional Parameters:

label

is an optional assembler label for the statement.

UWORD=

is an optional fullword that can be specified for any purpose you desire. When the exit routine gains control, the fullword is available to the exit, as described in Usage Note [“3” on page 20](#). Acceptable values are:

uword_label

specifies the address of the UWORD.

(reg)

specifies a register whose contents are stored as the UWORD.

SYSTEM=

specifies whether the abend exit routine survives CMS abend processing. Acceptable values are:

NO

specifies that the abend exit routine does not survive. This is the default value.

YES

specifies that the abend exit routine does survive. If you specify SYSTEM=YES, the abend exit routine must reside in storage that is not reclaimed during abend processing.

(reg)

specifies the register that contains the value for SYSTEM. The macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the SYSTEM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. To set the value at execution time, specify SYSTEM=*(reg)* or SYSTEM=*(addr,mask)*.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. CMS gives control to abend exit routines in the addressing mode of the program that issues the ABNEXIT macro.
2. In an XC virtual machine, your abend exit routine always receives control in primary space address translation mode and always must return control to CMS in primary space mode.
3. You must provide the proper entry and exit linkage for your abend exit routine. The abend exit routine receives control with the nucleus protect key and is disabled for interrupts. When your routine receives control, the register contents are as follows:

Register

Contents

R1

Address of an area of storage mapped by the CMSSDWA DSECT. To obtain the CMSSDWA expansion, call the DMSSDWA macro in the abend exit routine. UWORD can be found at the location SDWUWORD in the DMSSDWA DSECT.

R13

Address of an 18-fullword save area (for your use).

R14

Return address (see Usage Note [“5” on page 21](#)).

R15

Entry point address of your exit routine.

You can use the DMSSDWA macro, described on page [“Purpose” on page 173](#), to map the area pointed to by register 1.

4. In addition to register 1, the macro expansion for RESET uses registers 14 and 15. Your program must have a DSECT for NUCON when it uses ABNEXIT RESET or when it uses the MF=(E,addr) form of the macro and does not specify a function.

5. At completion, the abend exit routine can do one of the following:
 - Branch on register 14 to return to CMS. CMS calls any previous abend exits if they exist; if none exist, CMS continues with normal CMS abend recovery.
 - Load the PSW (or a modified version of the PSW) at the time of abend to return somewhere other than to CMS abend processing. Before it loads the PSW, the exit routine should issue an ABNEXIT RESET macro.
 6. Abend exit routines cannot clear or set other abend exit routines.
 7. If a program check occurs during an exit routine and ABNEXIT RESET has not been issued, control goes to the previous exit routine in the list. If there are no previous exits, CMS abend recovery occurs.
 8. Abend exit routines are disabled in two ways:
 - You can issue the ABNEXIT CLR macro at any time **except** from within an exit routine.
 - When CMS abend recovery occurs, CMS automatically clears all exit routines known to the system. That is, CMS exit routines defined without the SYSTEM attribute will have their definition structures cleaned up, therefore removing the linkage mechanism to the exit routine.
- Note:** Abend exits are not cleared at CMS end-of-command.
9. ABNEXIT processing is only one component of CMS abend recovery. Abend recovery can also be done through other facilities, including VMERROR event handlers. The following list summarizes the order of processing, organized by class of abend:

- Program Check
 - a. SPIE and ESPIE exits
 - b. STAE and ESTAE exits
 - c. VMERROR event handlers
 - d. VMERRORCHILD event handlers
 - e. ABNEXIT routines
- MVS ABEND macro
 - a. STAE and ESTAE exits
 - b. VMERROR event handlers
 - c. VMERRORCHILD event handlers
 - d. ABNEXIT routines
- DMSABN macro
 - a. VMERROR event handlers
 - b. VMERRORCHILD event handlers
 - c. ABNEXIT routines
 - d. STAE and ESTAE exits
- AbnormalEnd call
 - a. VMERROR event handlers
 - b. VMERRORCHILD event handlers
 - c. ABNEXIT routines
 - d. STAE and ESTAE exits

Only those STAE, ESTAE, SPIE, and ESPIE exits defined in the abending process are driven. ABNEXITs are driven regardless of the process that created them.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

ABNEXIT

Code	Meaning
8	No exit routines exist for the specified address.
12	ABNEXIT SET or CLR was issued from within an exit routine.
16	ABNEXIT RESET was issued from outside an exit routine.
24	Invalid function was specified.
104	Not enough storage is available to create the exit routine.

AMODESW

Purpose

Use the AMODESW macro to switch and, optionally, save a program's current addressing mode and switch addressing modes as a part of subroutine calls and returns.

Note: The AMODESW macro generates *dual-path* (System/370 or 370-XA) code. System/370 (370 mode) virtual machines are not supported. To eliminate the System/370 code path, use the MODE=NO370 option.

The general formats of the AMODESW macro are:

AMODESW CALL

Make a subroutine call with an appropriate mode switch.

AMODESW QRY

Determine current addressing mode.

AMODESW RETURN

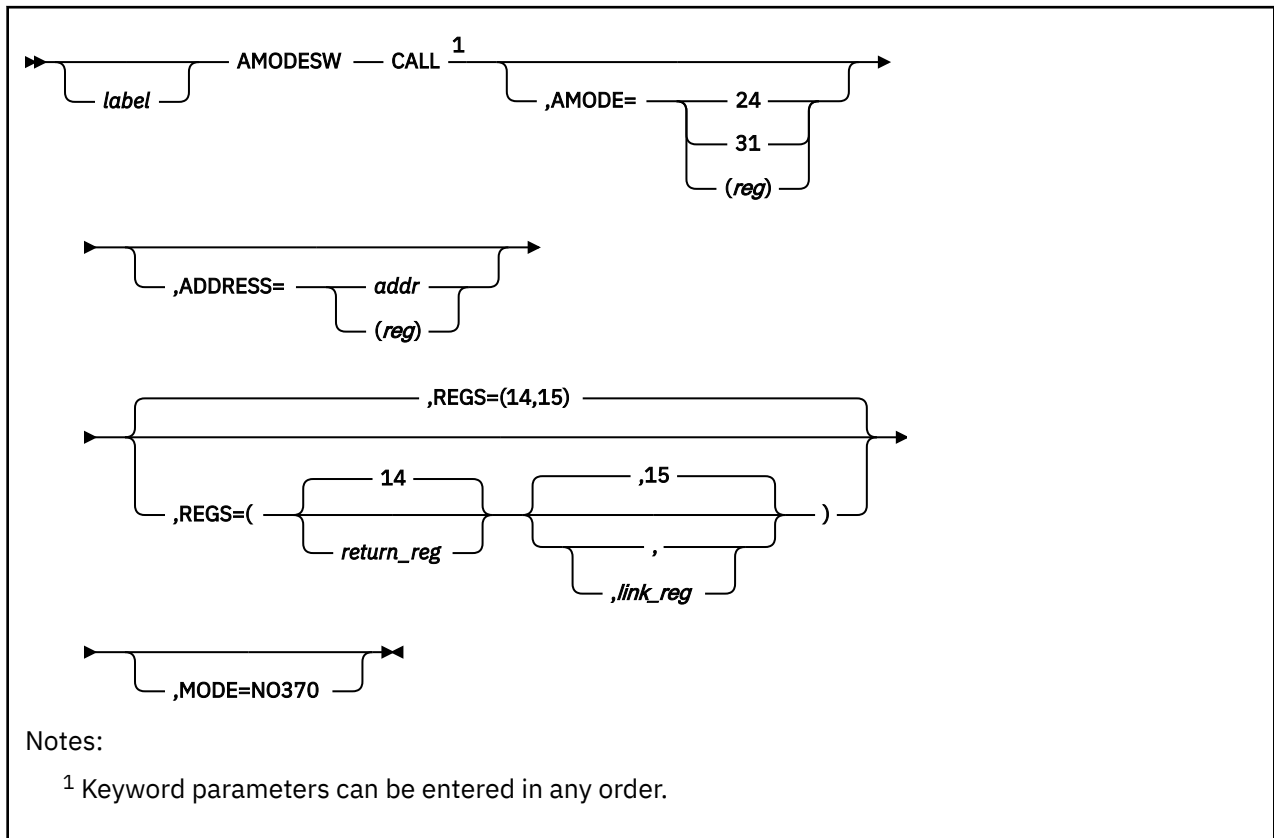
Return from a subroutine.

AMODESW SET

Switch addressing modes.

Note: While the AMODESW macro allows a program to switch addressing modes, the user must make sure programs follow 24-bit or 31-bit addressing conventions as appropriate. For more information on using 31-bit addressing and the AMODESW macro, see the [z/VM: CMS Application Development Guide for Assembler](#).

AMODESW CALL



Purpose

Use AMODESW CALL to make a subroutine call with an appropriate mode switch.

Parameters

Required Parameters:

CALL

calls a subroutine and makes the specified change in addressing mode.

Optional Parameters:

label

is an optional assembler label for the statement.

AMODE=

specifies the addressing mode of the called routine. Acceptable values are:

24

calls the routine in 24-bit addressing mode.

31

calls the routine in 31-bit addressing mode.

(reg)

sets the addressing mode according to the value of bit 0 of the specified register. A value of 0 gives you 24-bit addressing mode and a value of 1 gives you 31-bit addressing mode. The register specified must not be the same as the register used on the ADDRESS parameter or the register used as the return register.

If you do not specify AMODE, CMS sets the addressing mode as follows:

1. If you specify ADDRESS=, CMS obtains the new addressing mode from bit 0 of the address.
2. If you do not specify the AMODE or the ADDRESS parameter, CMS obtains the new addressing mode from bit 0 of the linkage register (see the description of the REGS parameter).

ADDRESS=

defines the location where control is transferred. Acceptable values are:

addr

specifies an address where control is transferred.

(reg)

specifies a register that contains the address where control is transferred. Valid registers are 1-15 enclosed in parentheses.

Note: If you do not specify ADDRESS, CMS passes control to the address in the linkage register (see the description of the REGS parameter).

REGS=

specifies the linkage registers for this call. Valid registers for *return_reg* or *link_reg* are 1-15. If you do not specify REGS, CMS uses register 15 for the *link_reg* and register 14 for the *return_reg* (REGS=(14,15)).

CMS uses the registers specified on the REGS parameter in the branch instruction for the subroutine call. AMODESW CALL issues a BASSM *return_reg*, *link_reg* instruction.

MODE=N0370

specifies that the macro should not create a System/370 code path.

Usage Notes

1. Users must restore their program's addressability (set up the proper base register) on return from the call. You can use the address CMS returns in the *return_reg* to set up program addressability.
2. AMODESW CALL and AMODESW RETURN allow you to call and return from subroutines. You can use them anywhere you can use a BALR and BR sequence. The sequence of instructions

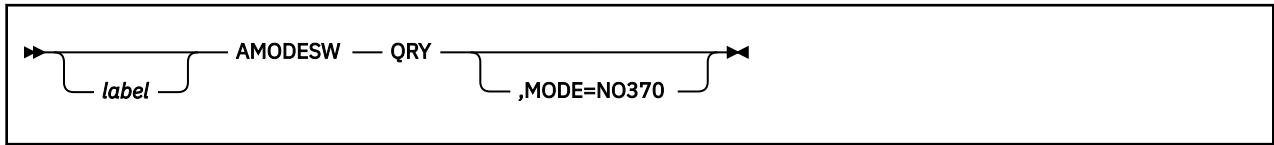
```

      .
      .
      .
      AMODESW CALL, ADDRESS=MYSUB, AMODE=31
      .
      .
MYSUB EQU *
      .
      .
      .
      AMODESW RETURN

```

(a) calls the subroutine at label MYSUB using a BASSM instruction, (b) switches to 31-bit addressing mode, and (c) saves the return address and the addressing mode in register 14 (the default value). The AMODESW RETURN instruction returns to the caller, restoring the addressing mode saved in register 14. You can use the REGS and REG parameters on CALL and RETURN to override the registers used for the BASSM linkage.

AMODESW QRY



Purpose

Use AMODESW QRY to determine the current addressing mode of a program.

Parameters

Required Parameters:

QRY

determines the virtual machine's current addressing mode. Upon completion, register 1 contains all 0's for 24-bit addressing mode or a nonzero (X'80000000') for 31-bit addressing mode.

Note: Register 1 is the only register AMODESW QRY alters.

Optional Parameters:

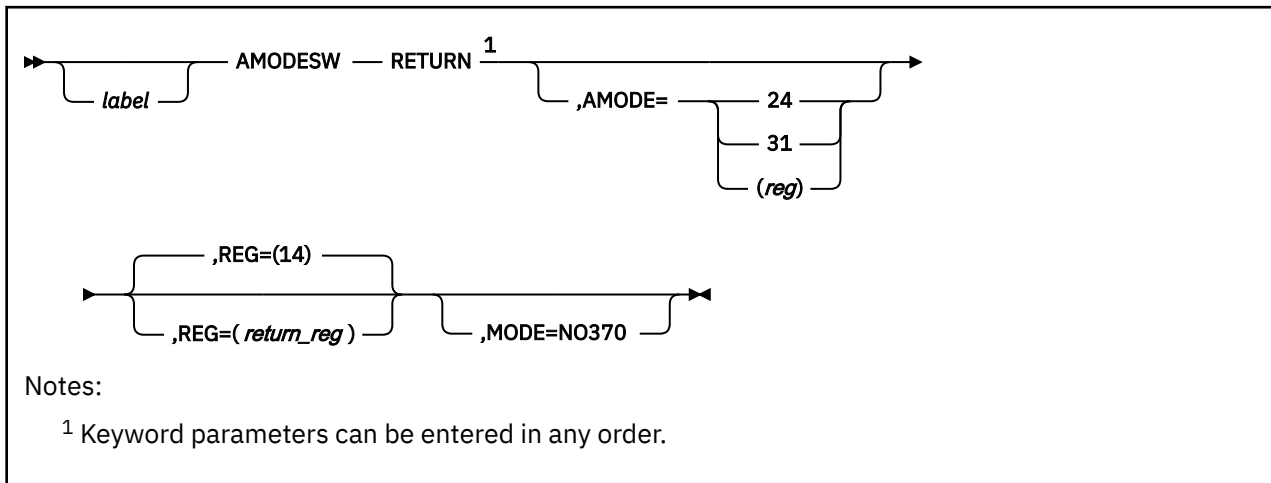
label

is an optional assembler label for the statement.

MODE=NO370

specifies that the macro should not create a System/370 code path.

AMODESW RETURN



Purpose

Use the AMODESW RETURN macro to return from a subroutine.

Parameters

Required Parameters:

RETURN

makes a return to the caller of the subroutine.

Optional Parameters:

label

is an optional assembler label for the statement.

AMODE=

specifies the new addressing mode to be set on the return. Acceptable values are:

24

returns to the caller in 24-bit addressing mode.

31

returns to the caller in 31-bit addressing mode.

(reg)

sets the addressing mode according to the value of bit 0 of the specified register. A value of 0 gives you 24-bit addressing mode and a value of 1 gives you 31-bit addressing mode. The register must be different from the one used as the return register (14 or the value specified on the REG parameter). When you specify AMODE=(reg) the specified register, (reg), is altered. For example, the macro call

```
AMODESW RETURN,AMODE=(2)
```

alters the contents of register 2.

If you do not specify AMODE, CMS sets the addressing mode according to the value of bit 0 of the return register (see the description of the REG parameter).

REG=

specifies the register that contains the address (and, optionally, the addressing mode) where control is returned. If you do not specify REG, CMS uses register 14 as the return register (REG=(14)).

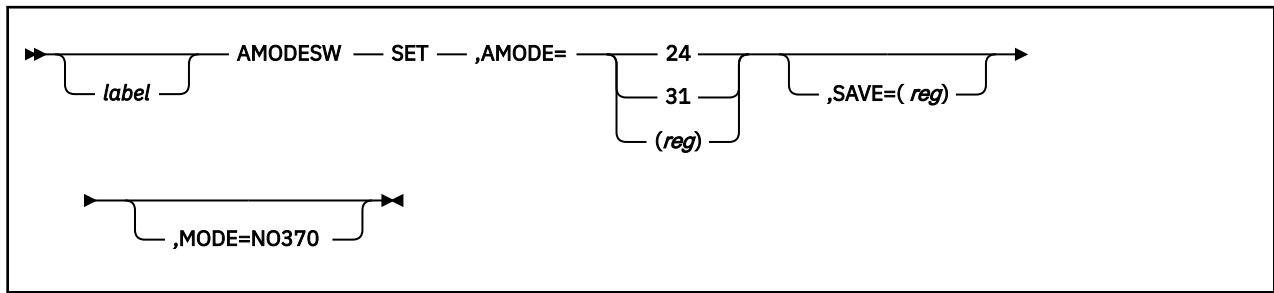
AMODESW RETURN

CMS uses the register specified on the REG parameter in the branch instruction for the subroutine return. AMODESW RETURN issues a BSM 0, *return_reg* instruction.

MODE=N0370

specifies that the macro should not create a System/370 code path.

AMODESW SET



Purpose

Use AMODESW SET to cause an inline switch in a program's addressing mode.

Parameters

Required Parameters:

SET

switches the addressing mode to a new value.

AMODE=

specifies the desired addressing mode. Acceptable values are:

24

switches to 24-bit addressing mode.

31

switches to 31-bit addressing mode.

(reg)

sets the addressing mode according to the value of bit 0 of the specified register. A value of 0 gives you 24-bit addressing mode and a value of 1 gives you 31-bit addressing mode.

Optional Parameters:

label

is an optional assembler label for the statement.

SAVE=(reg)

saves the current addressing mode in bit 0 of the specified register. If you do not specify SAVE, then the current mode is not saved. The valid registers are 1-14.

MODE=NO370

specifies that the macro should not create a System/370 code path.

Usage Notes

1. AMODESW SET alters register 15.
2. AMODESW SET switches a program's addressing mode without requiring a branch to a subroutine. For example, to switch the current addressing mode to 31-bit addressing, a program might use:

```
AMODESW SET,AMODE=31
```

3. To switch to a new mode from an unknown addressing mode **and** save the unknown mode for when you return, use the SAVE parameter. For example, the macroinstruction

```
AMODESW SET,AMODE=31,SAVE=(2)
```

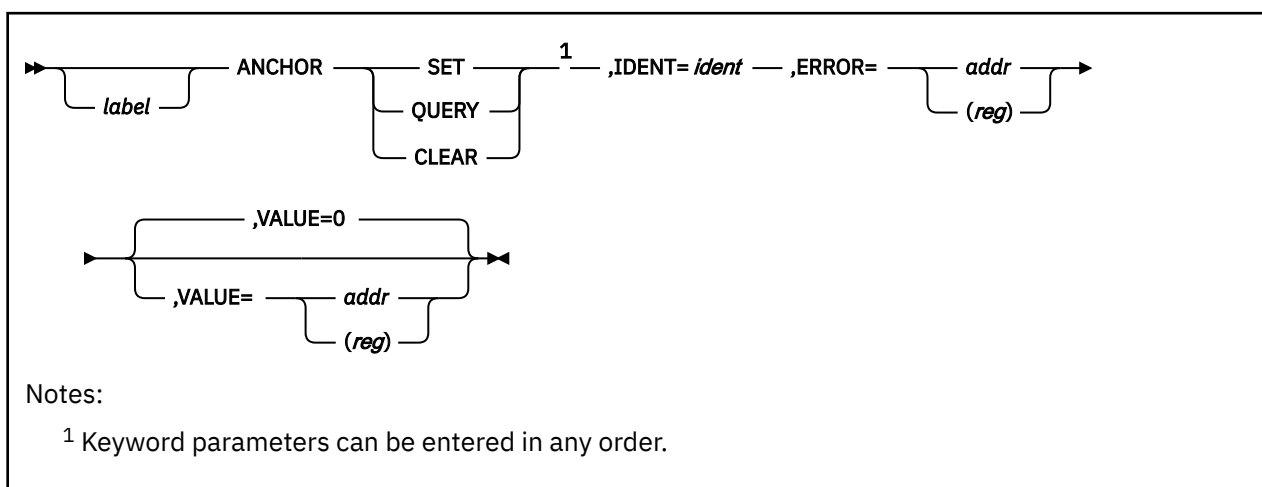
AMODESW SET

switches a program to 31-bit addressing mode and saves the current addressing mode as bit 0 of register 2. Only bit 0 of the SAVE register is altered.

You can then use the value set by the SAVE parameter on AMODESW SET to restore the original addressing mode:

```
AMODESW SET, AMODE=(2)
```

ANCHOR



Purpose

Use the ANCHOR macro to set, query, and clear a fullword that can be used by a program to save the address of its data between calls.

The ANCHOR macro is for programs with critical performance needs, such as one called multiple times per second. ANCHOR provides quick access to an anchor word, which is a fullword that points to one or more control blocks allocated in free storage by a program. This avoids the overhead of obtaining dynamic storage each time the program is called. This anchor word persists between calls to the program and persists after an abend occurs.

Before using ANCHOR, you must request an anchor identifier from IBM. This is necessary to ensure that your identifier is unique among all programs using the anchor facility.

To request your anchor identifier, complete the *ANCHOR Identifier Registration Form* in the back of this book and mail it to IBM. IBM will assign you an anchor identifier and notify you by mail.

Parameters

Required Parameters:

SET

initializes the anchor word to the value specified in VALUE. If VALUE is omitted, the anchor word is set to 0.

SET checks to see if an anchor slot has been assigned for your identifier. If an assignment has been made, the anchor word is updated with the data specified for VALUE. If an anchor slot has not been assigned, SET allocates one, fills in the anchor identifier, and sets the anchor word with the data specified for VALUE.

If there is not enough storage available to allocate the anchor block, or there are no anchor slots left in the anchor block, the routine specified in the ERROR parameter is run.

QUERY

returns the contents of the anchor word.

QUERY checks to see if an anchor slot has been assigned for your identifier. If it has, QUERY returns the anchor word in register 1.

If your program's anchor identifier was not found, the routine specified in the ERROR parameter is run.

QUERY is usually the first call of the Anchor facility from an application program.

CLEAR

sets the anchor entry in the anchor table to 0, making it available for other programs.

CLEAR checks to see if an anchor slot has been assigned for your identifier. If it has, CLEAR sets the anchor identifier and the anchor word to binary zeros.

If your program's anchor identifier was not found, the routine specified in the ERROR parameter is run.

IDENT=*ident*

is a 3-character anchor identifier that uniquely differentiates your program from other programs using the anchor facility. IDENT must be specified as an absolute expression. This identifier must be registered with IBM.

ERROR=

specifies an action taken if an error occurs. See Usage Note [“7” on page 33](#) for how an error is indicated. Acceptable values are:

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register. Valid registers are 2 through 12.

Optional Parameters:

label

is an optional assembler label for the statement.

VALUE=

specifies the data placed in the anchor word. Acceptable values are:

addr

specifies the address of the 4-byte storage area containing the anchor word data. This can be any valid assembler expression.

(reg)

specifies a register containing the anchor word data. Valid registers are 2 through 12.

If VALUE is omitted, the anchor word is set to 0. VALUE is ignored for the QUERY and CLEAR functions.

Usage Notes

1. The anchor facility keeps a list of 16 anchor words and their associated anchor identifiers. The number of anchor slots is limited to 16 for two reasons:
 - a. To reduce the time to search the list
 - b. Because it is highly unlikely that more than 16 performance-critical applications would be competing for execution at the same time during a CMS session.

Programs that do not have critical performance needs should use a nucleus extension to keep their anchor word. For more information on the description of user words in nucleus extensions, see the [z/VM: CMS Application Development Guide for Assembler](#).
2. The anchor identifier is passed to the anchor facility in register 0.
3. If you want to clear your anchor word after an abend to prevent reuse of possibly corrupted data areas, you can call the ABNEXIT macro or establish a nucleus extension that is called during abend cleanup processing. For more information on the nucleus extensions, see the [z/VM: CMS Application Development Guide for Assembler](#).
4. To obtain the fastest possible anchor word lookup time, ensure that your program is the first to set its anchor word after IPL of the virtual machine.
5. If you have a very large program that contains many subprograms needing anchors, you do not need multiple anchor identifiers. Instead, allocate storage for an array that points to each data area for a particular subprogram. The anchor word can then point to this array.
6. Anchor support does not save and restore translation mode (primary space or access register mode) or access registers.

7. Register 15 will not contain a return code after ANCHOR processing. Instead, a nonzero condition code will cause the routine specified in the ERROR parameter to be run.

Anchor Entry Conditions: When your program calls the anchor facility, the register contents are:

**Register
Contents**

R0

Your 3-character anchor identifier and a blank.

R1

During an ANCHOR SET, this register contains the anchor word data.

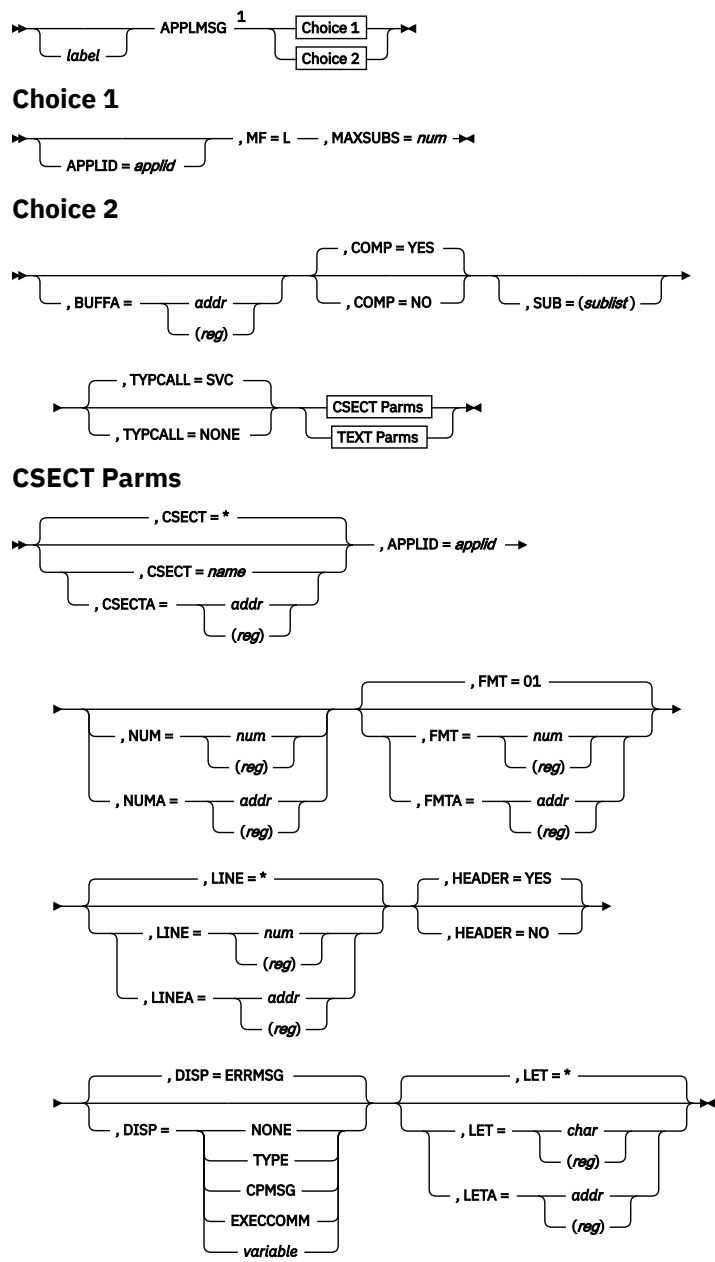
R14

Return address.

R15

Entry point address.

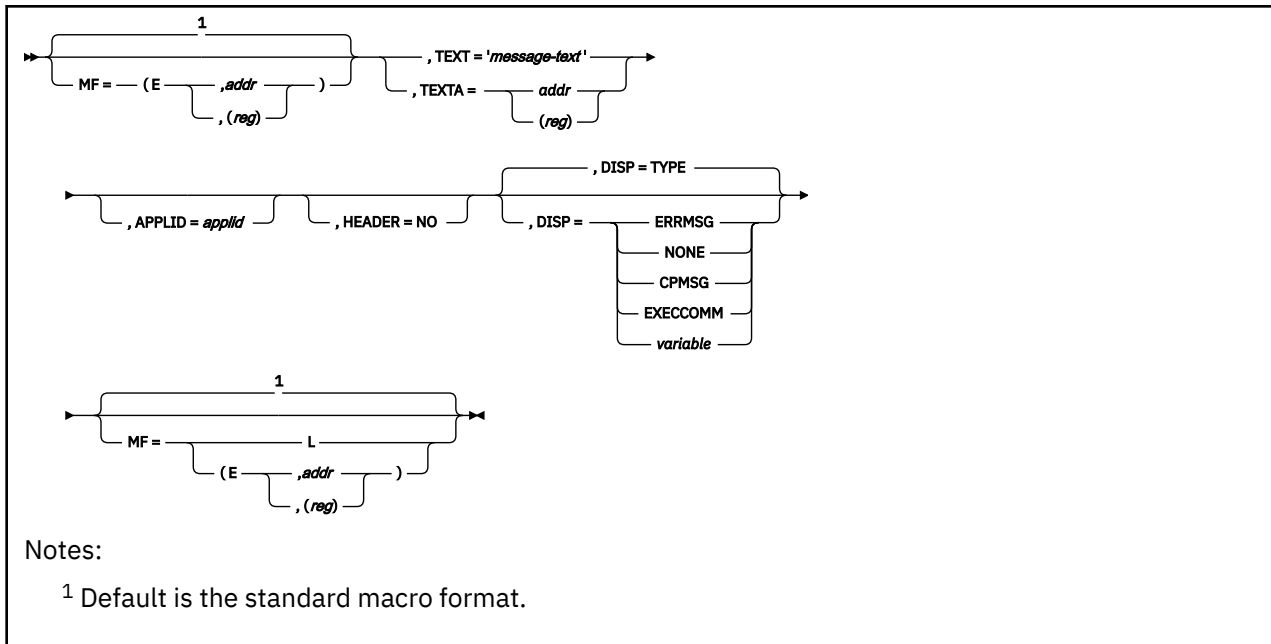
APPLMSG



Notes:

¹ Keyword parameters can be entered in any order.

TEXT Params



Purpose

Use the APPLMSG macro in an assembler program to retrieve a message from a message repository. (A message repository contains translated versions of system messages in the specified language.) You can optionally display the message at your terminal.

Parameters

APPLID=*applid*

specifies the name of the application that issues the message. CMS compares the 3-character application ID to the application ID in the repository information chain to retrieve the message from the proper repository. The application ID is also displayed in the message header.

Note: The APPLID parameter is optional when you specify TEXT or TEXTA unless you specify the SUB parameter with a *type* of DICT.

label

is an optional assembler label for the statement.

MF=

specifies the macro format. Using different macro formats, you can either code parameters directly in the macro call or put them at a place in the program where they can be referenced later.

The standard format (without the MF= operand) generates an inline operand list and invokes the message facility. This is the default format. You can specify a maximum of 20 substitutions with this macro format. Other acceptable formats are:

L

the list format is used only together with the MF=(E,...) macro format of APPLMSG. MF=L generates a storage area for the parameter list; this storage area later gets filled in when you use the execute form, MF=(E,...).

The size of the parameter list area you want to reserve depends on the number of substitutions to be made. Use the MAXSUBS operand to specify the size of this area. For example, the following would reserve space for a parameter list that can hold up to five substitutions.

```
MF=L, MAXSUBS=5, . . .
```

(E,addr)

generates code to fill in the parameter list at the address you specify, and invokes the message facility. For example:

```
MF=(E,label),...
```

(E,(reg))

generates code to fill in the parameter list at the address you specify (contained in (reg)) and invokes the message facility.

CSECT=

overrides the default CSECT identifier that goes in the message header. Acceptable values are:

specifies to use the default CSECT identifier. This is the default value.

name

specifies the name of the CSECT identifier.

By default, APPLMSG uses the first 3 characters of the module name if they are different from the application ID; if the 3 characters are the same as the application ID, then APPLMSG uses the next 3 characters of the module name.

You cannot specify CSECTA, TEXT, or TEXTA with CSECT.

Note: You cannot set CSECT to a variable. Use the CSECTA parameter to specify a variable CSECT identifier.

CSECTA=

overrides the default CSECT identifier that goes in the message header. Acceptable values are:

addr

specifies the address of a variable containing the name of the CSECT identifier. The length of the address or variable should be declared as a character length of 6.

(reg)

specifies a register containing the address of a variable which contains the CSECT identifier name.

For both forms, the length of the address or variable should be declared as a character length of 6. Neither form can be specified with MF=L.

You cannot specify CSECT, TEXT, or TEXTA with CSECTA.

BUFFA=

specifies the address of a buffer where APPLMSG copies the complete message. Acceptable values are:

addr

specifies the address.

(reg)

specifies a register that contains the address.

Use DISP=NONE when you want to copy the message to the buffer but not display it.

When text is copied into a buffer, the length of the message is in the first byte of the buffer, preceding the text. The message header (for example, DMSxxxxnnns) is also part of the copied information (unless you specify HEADER=NO).

Note: Store the length of the buffer, not including the length byte, in the first byte of the buffer before you call APPLMSG. This ensures that CMS does not overwrite any data immediately following the buffer.

COMP=

specifies whether multiple blanks in the message text are to be removed, including those preceding and following a substitution field. Acceptable values are:

YES

specifies to remove multiple blanks. This is the default value.

NO

specifies not to remove multiple blanks. If you specify COMP=NO without the SUB operand, the message, as defined in the message repository, is not scanned. Extra blanks are not removed and substitution indicators are not removed or replaced.

For example, if a message is defined in the repository with a substitution indicator of &1 and the message is invoked with no SUB operand and COMP=NO specified, then the &1 appears in the displayed message. To prevent the &1 substitution indicator from appearing in the message, a substitution must be specified on the message invocation. This can be done by coding a SUB operand for a single substitution. Specifying a null character as the substitution on the APPLMSG invocation causes the message text to be scanned to remove substitution indicators such as &1, &2, and so on.

When a double-byte character set (DBCS) language is being used (GENMSG is issued with the DBCS option), the message is always scanned.

DISP=

specifies the display format (disposition) of the message. Acceptable values are:

ERRMSG

specifies that the message line is displayed according to the CP EMSG setting. If EMSG is set to:

- ON - the entire message is displayed, header plus text
- OFF - no message is displayed
- TEXT - only the text portion is displayed
- CODE - only the 10- or 11-character header is displayed.

ERRMSG is the default DISP value unless you specify TEXT, TEXTA, or HEADER=NO. If you specify TEXT, TEXTA, or HEADER=NO, TYPE is the default DISP value.

NONE

specifies that no output occurs. DISP=NONE is useful with the BUFFA operand.

TYPE

specifies that the message is displayed on the terminal. This would be the same as DISP=ERRMSG with EMSG TEXT. TYPE is the default if you specify TEXT, TEXTA, or HEADER=NO; otherwise, ERRMSG is the default.

Note: If the message text wraps to a second line, a split can occur in the middle of a word.

CPMSG

specifies that the message is passed to CP to be issued as a CP message.

EXECCOMM

specifies that the message is returned to a variable in the calling exec. The complete message is copied into the variable 'MESSAGE', with the first line in 'MESSAGE.1', the second in 'MESSAGE.2', and so on. The number of lines in the message is copied into 'MESSAGE.0'. This is only used when the module issuing APPLMSG is called from an exec. If TEXT or TEXTA is also specified, only 'MESSAGE.1' is filled in and 'MESSAGE.0' is given a value of 1. This line is limited to 256 characters.

variable

specifies that a variable shows the message display format to be used. The variable must be 1 byte long, and the low-order 3 bits of the byte must be set to the desired disposition as follows:

```
ERRMSG = 000
TYPE   = 001
NONE   = 010
CPMSG  = 011
EXECCOMM = 100
```

HEADER=

specifies whether you want a header created for the message. The repository describes how many digits of the message number to display. Acceptable values are:

YES

specifies a header. You cannot specify HEADER=YES with the TEXT or TEXTA option. This is the default. If you specify DISP=ERRMSG, HEADER=YES is ignored.

NO

specifies no header. You cannot specify HEADER=NO with the DISP=ERRMSG option.

The standard header format of VM error messages is

```
xxxxmmnnnnns or xxxmmnnnnns
```

where:

- xxx is the application ID
- mmm is the CSECT name
- nnn or nnnn is the message number
- s is the severity code

The following is a list of the most commonly used severity codes:

Code**Message Type****E**

Error

I

Information

R

Response

S

Severe

T

Terminal

W

Warning

LET=

specifies a severity letter for the message. A default severity code letter is already provided in the message repository; you should use this parameter only when you want to override the provided severity.

You cannot specify LETA, TEXT, or TEXTA with LET. When TEXT or TEXTA is specified, DISP defaults to TYPE.

Acceptable values are:

specifies the default severity. This is the default value.

char

specifies the severity code letter.

(reg)

specifies the register that contains the severity code letter.

LETA=

specifies a severity letter for the message. Acceptable values are:

addr

specifies the address of the severity letter.

(reg)

specifies the register containing the address of the severity letter.

You cannot specify LET, TEXT, or TEXTA with LETA.

NUM=

specifies the number of the message you want. The message number is one to four digits and it locates the associated message text in the repository. This parameter is required with all formats except the list format.

You cannot specify NUMA, TEXT, or TEXTA with NUM. When TEXT or TEXTA is specified, DISP defaults to TYPE.

Acceptable values are:

num

specifies the message number.

(reg)

specifies the register containing the message number.

NUMA=

specifies the number of the message you want. The message number is one to four digits and it locates the associated message text in the repository. If NUMA is used, then the message number should be defined as a halfword. This parameter is required with all formats except the list format. Acceptable values are:

addr

specifies the address of the message number.

(reg)

specifies the register containing the address of the message number.

You cannot specify NUM, TEXT, or TEXTA with NUMA.

FMT=

specifies the message format number. The format number is a 1- or 2-digit number that identifies different versions of the same message which have the same message number. The formats are numbered from 01 to 99. The default is 01. A format of 00 is not allowed.

You cannot specify FMTA, TEXT, or TEXTA with FMT. When TEXT or TEXTA is specified, DISP defaults to TYPE.

Acceptable values are:

num

specifies the number.

(reg)

specifies the register containing the number.

FMTA=

specifies the message format number. The format number is a 1- or 2-digit number that identifies different versions of the same message which have the same message number. The formats are numbered from 01 to 99. A blank format defaults to 01. A format of 00 is not allowed. If FMTA is used, then the message format should be defined as one byte. Acceptable values are:

addr

specifies the address of the message format number.

(reg)

specifies the register containing the address of the message format number.

You cannot specify FMT, TEXT, or TEXTA with FMTA.

LINE=

specifies the line number of a message. The line number is a 1- or 2-digit number that identifies each line of a multi-line message.

Lines are numbered from 01 to 99.

You cannot specify LINEA, TEXT, or TEXTA with LINE. When TEXT or TEXTA is specified, DISP defaults to TYPE.

If BUFFA is not specified, the default for LINE is an asterisk (*). If BUFFA is specified, the default for LINE is 01. A line number of 00 is not allowed. Each line may be up to 240 characters long. Acceptable values are:

num

specifies the line number.

(reg)

specifies the register that contains the line number.

specifies that all lines for a certain message number and format are to be retrieved. You may only specify an asterisk with the LINE option (not with LINEA), and the asterisk must be hardcoded (not used in a register). You may not specify an asterisk for a line number if you use the BUFFA option.

LINEA=

specifies the line number of a message. The line number is a 1- or 2-digit number that identifies each line of a multi-line message. Lines are numbered from 01 to 99. If you use LINEA, you must define the line number as 1 byte. Acceptable values are:

addr

specifies the address of the line number.

(reg)

specifies the register that contains the address of the line number.

You cannot specify LINE, TEXT, or TEXTA with LINEA.

SUB= (*sublist*)

specifies the type of substitution to be performed on those portions of the message where substitutions are indicated.

Acceptable values for *sublist* are:

(type,(value,length))

specifies the type of data, its address, and the length of the substitution.

(type,value)

specifies a number used to retrieve the substitution information from the repository.

If you specify a length, you must enclose the value and length in parentheses. Otherwise, do not enclose the value in parentheses.

You can specify both the value and length using register notation. When you specify the length, it is interpreted to be the length of the input field, except when used with the HEX, HEXA, HEX4A, DEC and DECA parameters. For these parameters, the length represents the length of the converted result. Following are the possible values of *type*.

DICT,*number*

DICT,*(reg)*

indicates that the substitution is a dictionary item. The number of the dictionary item in the repository is specified by number or the value in *(reg)*.

You cannot specify a length with DICT. Also, it is recommended that you use only system keywords (for example, PROFILE, NOPROFILE, or XEDIT) to specify a dictionary item.

If you specify DICT, the APPLID parameter is required.

HEX,*expression*

HEX,*(reg)*

converts to graphic hexadecimal the expression or the value in the specified register. The length indicates the number of digits of the converted fullword to be displayed. The default length is 8 hexadecimal digits (4 bytes). The word is truncated from the left.

HEXA,*address***HEXA,(*reg*)**

converts to graphic hexadecimal the fullword at the specified address or indicated at the address in (*reg*). You may specify a length with type HEXA; the default is 8 hexadecimal digits (4 bytes). The length indicates the number of digits of the converted fullword to be displayed. The word is truncated from the left.

HEX4A,*address***HEX4A,(*reg*)**

converts to graphic hexadecimal the data at the specified address or at the address indicated in (*reg*). The value you specify is converted and substituted into the message text. Leading zeros are not suppressed. A blank character is inserted following every 4 bytes (8 characters of output). The data to be converted does not have to be on a fullword boundary.

The length field is required with type HEX4A. The length you specify indicates the number of bytes of the converted data to be displayed. This length does not include the blanks that are inserted following every 4 bytes. The data is truncated from the right.

DEC,*expression***DEC,(*reg*)**

converts to graphic decimal the expression or the value in the specified register.

You can specify a length with type DEC; the default is 15 digits (excluding the sign if the number is negative). The length indicates the number of digits of the converted fullword to be displayed, excluding the minus sign. The word is truncated from the left.

DECA,*address***DECA,(*reg*)**

converts to graphic decimal the fullword at the specified address or at the address in (*reg*). The value you specify is converted and substituted in the message text. Leading zeros are suppressed. If the number is negative, a leading minus sign is inserted.

You can specify a length with type DECA; the default is 15 digits (excluding the sign if the number is negative). The length indicates the number of digits of the converted fullword to be displayed, excluding the minus sign. The word is truncated from the left.

DEV,*expression***DEV,(*reg*)**

specifies a value, *expression*, or a register, *reg*, that contains a value, which can be up to a fullword of binary or hexadecimal data.

If the value is hexadecimal data, then no conversion is performed. If the value is binary data, then it is converted to a hexadecimal number.

If the hexadecimal number is greater than 4 digits, then the rightmost 4 digits are used. If the hexadecimal number is less than 4 digits long, then it is padded on the left with zeros to bring its length to 4 digits.

If the leftmost digit of the hexadecimal number is 0, then it is dropped and the rightmost 3 digits are used; otherwise, all 4 digits are used.

DEVA,*address***DEVA,(*reg*)**

specifies an address, *address*, or a register, *reg*, that contains an address, which points to a halfword in storage. The 2 bytes of data consist of 4 hexadecimal digits.

If the leftmost digit of the 4-digit group is 0, then it is dropped and the rightmost 3 digits are used; otherwise, all 4 digits are used.

DEVCA,*address***DEVCA,(*reg*)**

specifies an address, *address*, or a register, *reg*, that contains an address, which points to a fullword in storage that contains character data.

The character data at this fullword in storage is converted to its EBCDIC value, resulting in a 4-digit number.

If the leftmost digit of the 4-digit number is 0, then it is dropped and the rightmost 3 digits are used; otherwise, all 4 digits are used.

CHARA,*address*

CHARA,*(reg)*

substitutes into the message text the character data at the specified address or at the address indicated in *(reg)*.

The length field is mandatory with type CHARA.

CHAR8A,*address*

CHAR8A,*(reg)*

substitutes into the message text the character data at the specified address or at the address indicated in *(reg)* and inserts a blank character following each 8 characters of output.

The length field is mandatory with type CHAR8A. This length indicates the number of actual characters to be displayed, not including the blanks that are inserted after each 8 characters.

MAXSUBS=*num*

reserves program storage to build the parameter list. The number you specify is the maximum number of substitutions, and that determines the size of the area saved. It is used only with the MF=L macro form.

If you specify MAXSUBS and a SUB list, APPLMSG takes the maximum number of substitutions. That is, if you specify both MAXSUBS=1 and

```
SUB=(TYPE, (VALUE, LENGTH), TYPE, (VALUE, LENGTH))
```

then 2 is the value used. The number of substitutions is multiplied by the amount of space required for each substitution and added to the storage required for the remainder of the parameter list.

The maximum number of substitutions is 20.

TEXT='message-text'

directly specifies the message text to be used, instead of using a repository.

The substitution character defaults to '&' if you specify TEXT. A header is not created for the message, so the message header must be included as part of the text when the DISP=ERRMSG option is used with the TEXT or TEXTA option. The header option may not be specified when DISP=ERRMSG is used with TEXT or TEXTA. The HEADER=YES option may not be specified with TEXT or TEXTA.

You cannot specify CSECT, CSECTA, FMT, LET, LINE, NUM, or TEXTA with TEXT. When TEXT is specified, DISP defaults to TYPE.

The APPLID parameter is optional when you specify TEXT or TEXTA unless you specify the SUB parameter with a *type* of DICT.

If you code TEXT or TEXTA to display a message, that message always appears in the same language, even if your current language changes.

TEXTA=

directly specifies the message text to be used, instead of using a repository.

The substitution character defaults to '&' if you specify TEXTA. A header is not created for the message, so the message header must be included as part of the text when the DISP=ERRMSG option is used with the TEXT or TEXTA option. The header option may not be specified when DISP=ERRMSG is used with TEXT or TEXTA. The HEADER=YES option may not be specified with TEXT or TEXTA.

You cannot specify CSECT, CSECTA, FMT, LET, LINE, NUM, or TEXT with TEXTA. When TEXTA is specified, DISP defaults to TYPE.

The APPLID parameter is optional when you specify TEXT or TEXTA unless you specify the SUB parameter with a *type* of DICT.

If you code TEXT or TEXTA to display a message, that message always appears in the same language, even if your current language changes. Acceptable values are:

addr

specifies the address of the message text

(reg)

specifies the register containing the address of the message text

If you specify TEXTA, the first byte at the address specified must contain the length of the message text. For example:

```
APPLMSG TEXTA=MESSAGE
      .
      .
MESSAGE DC X'16'
         DC CL22'THIS IS A LINE OF TEXT'
```

TYPCALL=

specifies the type of call you want to generate. Acceptable values are:

SVC

the macro generates a CMSCALL to call the DMSMG module. This is the default value.

NONE

no call to DMSMG module is generated.

The processing that takes place in the macro depends on the value of the MF parameter and the TYPCALL parameter.

(MF= operand not specified)

generates a series of assembler statements that declare the parameters inline for use by the message processor module (DMSMG). If you use substitutions whose values and lengths do not use registers or are not hardcoded, then nonreentrant code is generated for this macro format. The macro then generates a call to DMSMG module depending on the value of the TYPCALL parameter.

MF=L

generates a DS assembler statement to reserve a storage area APPLMSG can use later. The size of the area of storage reserved depends on the value of the MAXSUBS parameter. No call to DMSMG is generated, and no parameter information is set up.

MF=E

generates a series of assembler statements that build a record in the specified buffer area. This record contains the parameters for use by DMSMG module.

The macro then generates a call to DMSMG module depending on the value of the TYPCALL parameter as described for the standard format.

Usage Notes

1. For more information on developing and managing message repositories, see the *z/VM: CMS Application Development Guide*. The *z/VM: CMS Application Development Guide for Assembler* also contains a sample program using the APPLMSG macro to access messages in a repository.
2. APPLMSG contains many of the functions of the LINEDIT macro; it also lets you specify just a message number rather than coding the entire message text. This allows for more flexibility, because a different repository can be in storage and the same message would come up, only it would be in a different language.
3. You should have a copy of the message repository you want to access. This way you can see the message numbers, formats, lines, and substitution slots.
4. When you use the DEV, DEVA, or DEVCA substitution types, the support device address must contain 4 digits.

5. For more information on installing a different system national language and on using other national languages supported by z/VM, see [z/VM: Installation Guide](#).

Examples

See [z/VM: CMS Application Development Guide for Assembler](#) for usage examples of the APPLMSG macro.

For the examples in this section, assume a message repository contains the following messages and dictionary items:

Messages:

```
08750101e Attempt to divide by &1 is invalid
08750201e Attempt to &2 by &1 is invalid
08760101e Error &X-1 rc = &X-3
08770101e This is a multi-line message.
      NOCOMP must be specified in
08770102e order to keep the
      return codes lined up on the next line.
```

```
08770103e          RC 1 = &X-1          RC 2 = &X-2
|
|
|-----severity code
|-----line of message
|-----format of message
|-----number of message
```

Dictionary items:

```
90250101  divide 90260101  reading from &2 90270101  tape
```

Sample code that displays error messages when it attempts to divide by zero:

```
SAMP CSECT
      ENTRY T
* TRY  SOME APPLMSG MACRO CALLS T      DS  0H
      LR  10,15
      USING SAMP,10
* SET  UP THE REGISTERS FOR THE DIVIDE
      L   3,=F'0'          R3=0
      L   4,=F'10'         R4=10
      L   5,=F'0'          R5=0
      CR  3,5              COMPARE REGISTER 5 TO 0
      BE  ERR0             IF REG 5 IS 0, ISSUE AN ERROR MESSAGE
      DR  4,5              OTHERWISE, DO THE DIVIDE
      B   DONE
      ----- issue error message; see cases below -----
      .
      .
      .
DONE  DS  0H
      BR  14
ERR1  APPLMSG MF=L,MAXSUBS=2
```

Case 1: This call accesses the repository to print CMS message 875, format 1. The parameter list for APPLMSG is set up inline. (The substitution is a 1-digit decimal number in register 5.)

```
ERR0  APPLMSG NUM=875,FMT=1,
      APPLID=CMS,COMP=YES,SUB=(DEC,((5),1)),
      DISP=TYPE,TYPCALL=SVC
```

Case 2: This call accesses the repository to print the message. The parameter list for APPLMSG is set up at ERR1. (Again, the substitution is a 1-digit decimal number in register 5.)

```
ERR0  APPLMSG MF=(E,ERR1),NUM=875,FMT=1,
      APPLID=CMS,COMP=YES,SUB=(DEC,((5),1)),
      DISP=TYPE,TYPCALL=SVC
```


Case 3: This call uses a dictionary item for the second substitution in the message.

```
ERR0  APPLMSG  MF=(E,ERR1),NUM=875,FMT=02,
              APPLID=CMS,COMP=YES,SUB=(DEC,((5),1),DICT,9025),
              DISP=TYPE,TYPICAL=SVC
```

Note: In this case, the dictionary item is a system keyword (DICT=DIVIDE).

Case 4: This call uses the TEXT parameter to print the message directly, without using the repository:

```
ERR0  APPLMSG  APPLID=CMS,COMP=YES,SUB=(DEC,((5),1)),
              DISP=TYPE,TYPICAL=SVC,
              TEXT='ATTEMPT TO DIVIDE BY &&1 IS INVALID'
```

To get the substitution character (&1) to appear in the message text, it is necessary to code two ampersands.

Note that this method was used prior to VM/SP Release 5, but is no longer recommended. Cases 1 through 3 show the preferred methods for text substitution.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

4

A message was produced, but the text was truncated because the:

- User buffer is too short to contain the message text
- Final message text with substitutions is longer than 240 characters.

Execution continues.

40

An invalid DISP value was received; APPLMSG macro terminates as a result of DISP parameter validation, the macro request was not performed, and processing continues with next sequential instruction.

104

EXECCOMM failed; APPLMSG macro terminates as a result of DISP parameter validation, the macro request was not performed, and processing continues with next sequential instruction.

BATLIMIT



Purpose

Use the BATLIMIT macro to generate a DSECT for the BATLSECT DSECT.

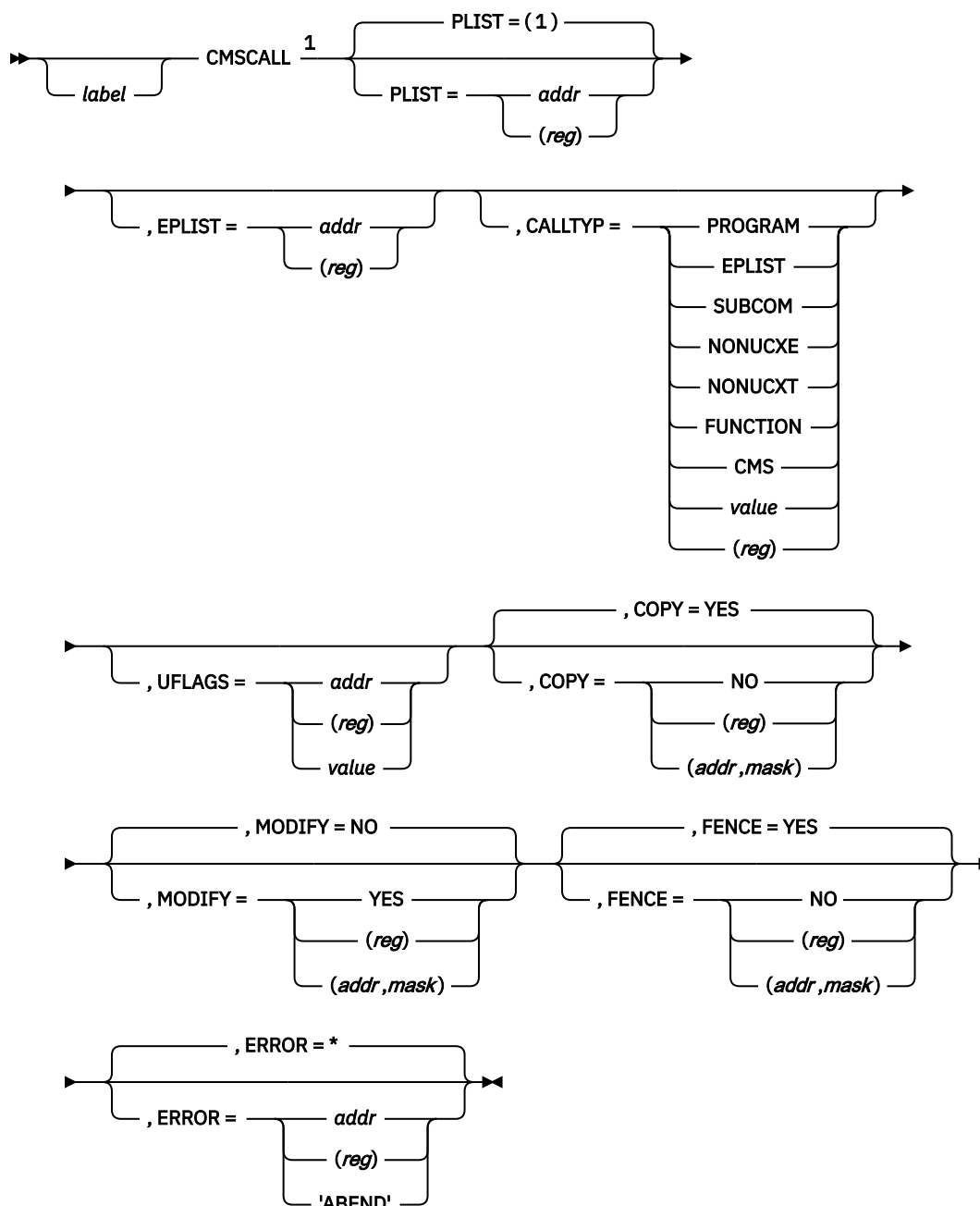
Usage Notes

- 1. For more information on the BATLIMIT macro, see [z/VM: CMS Planning and Administration](#).
- 2. The BATLIMIT macroinstruction expands as follows:

```

      BATLIMIT
BATLSECT DSECT
*
*
*      CMS BATCH USER JOB LIMITS
*
BATCPUL DC    F'131068' -    VIRT.CPU LIMIT (SEC.) - CAN BE RESET
BATCPUC DC    F'0'         -    CURRENT CPU COUNT   - DO NOT RESET
BATPRTL DC    F'131068' -    NO. PRINTED LINES LIMIT - CAN BE RESET
BATPRTC DC    F'0'         -    CURRENT LINE COUNT   - DO NOT RESET
BATPUNL DC    F'131068' -    NO. PUNCHED CARDS LIMIT - CAN BE RESET
BATPUNC DC    F'0'         -    CURRENT CARD COUNT   - DO NOT RESET
```

CMSCALL



Notes:

¹ Keyword parameters can be entered in any order.

Purpose

Use the CMSCALL macroinstruction to invoke a CMS command, CMS function, EXEC, or user MODULE. Your program must build the standard tokenized parameter list that the routine being invoked needs. The first token is the name of the routine CMSCALL invokes. The CMSCALL macro has only the standard macro form, which generates reentrant code.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

PLIST=

specifies the address of the tokenized parameter list for the command. CMSCALL loads the address into register 1. Acceptable values are:

(1)

specifies that register 1 contains the address of the tokenized parameter list. This is the default.

addr

specifies the address of the tokenized parameter list. This may be any valid assembler expression.

(*reg*)

specifies a general register (2-12) in parentheses which contains the address of the tokenized parameter list.

Note: You can use the SCAN macro to create the tokenized parameter list and the extended parameter list.

EPLIST=

specifies the address of an extended parameter list for the command. CMSCALL loads the address into register 0 and sets to 1 a bit (USEPLIST) in the user save area (USERSAVE). Acceptable values are:

addr

specifies the address of the extended parameter list. This may be any valid assembler expression.

(*reg*)

specifies a general register (0, 2-12) in parentheses which contains the address of the extended parameter list.

CALLTYP=

specifies the type of invocation for this call. These types correspond to the codes found in the high-order byte of SVC 202; they are available in the user save area, which register 13 points to on invocation of the program. Acceptable values are:

PROGRAM

X'00'—instructs the macro to pass a tokenized parameter list. This is the default value unless you specify the EPLIST parameter.

EPLIST

X'01'—instructs the macro to pass a tokenized parameter list and an extended parameter list. This is the default value **if** you specify the EPLIST parameter. A program invoked by REXX when 'ADDRESS COMMAND' is in effect will have a call type of X'01'.

SUBCOM

X'02'—instructs the macro to use the SUBCOM interface to make the call.

NONUCXE

X'03'—instructs the macro to pass an extended parameter list and then, during the command search, bypass the search of the list of nucleus extensions.

NONUCXT

X'04'—instructs the macro to pass a tokenized parameter list and then, during the command search, bypass the search of the list of nucleus extensions.

FUNCTION

X'05'—instructs the macro to call a REXX function or subroutine. This call type acts as if it was invoked using 'ADDRESS COMMAND' from REXX/VM.

CMS

X'0B'—instructs the macro to simulate invocation from a console and to pass a tokenized parameter list and an extended parameter list. A program invoked by REXX when 'ADDRESS CMS' is in effect will have a call type of X'0B'.

value

specifies a 1-byte constant that represents other call-type codes. The constant can be any 1-byte self-defining term, such as X'F2', C"2", or B'11110010'. Also, the constant must be an X, C, or B type data constant, it cannot use length modifiers, and it must not be greater than 1 byte in length.

(reg)

specifies a register in the range 2-12 enclosed in parentheses, that contains a call-type code in the low-order byte. Note that CMSCALL modifies the contents of the register.

If you specify the EPLIST parameter, the valid CALLTYPs are EPLIST, SUBCOM, FUNCTION, NONUCXE, CMS, *(reg)*, or *value*. If you do not specify the EPLIST parameter, the valid CALLTYPs are PROGRAM (the default), SUBCOM, NONUCXT, *(reg)*, or *value*. If you specify CALLTYP as *value* or *(reg)*, CMSCALL does not check the code for conflicts (this is because you may define your own call-type codes).

To determine what CALLTYP was made, the program being invoked can interrogate the field USECTYP in USERSAVE.

UFLAGS=

is an optional 1-byte parameter stored in the USEUFLG byte of the user save area (USERSAVE). (Upon invocation, register 13 points to USERSAVE.) Acceptable values are:

addr

specifies an address of a 1-byte field that contains the user flags.

(reg)

specifies a register in the range 2-12 that contains the flag information in the low-order byte. (If you specify a register, CMSCALL clears its contents after it executes.)

value

specifies the user flags as a 1-byte constant. If you specify a constant, it can be any 1-byte self-defining term, such as X'F2', C"2", or B'11110010'. It must be an X, C, or B type data constant, it cannot use length modifiers, and it must not be greater than 1 byte in length.

COPY=

specifies whether CMSCALL copies the extended and tokenized parameter lists if their addresses are above 16 MB and the called program has an addressing mode of 24. CMSCALL copies the extended parameter list only if it was provided using the EPLIST parameter. If it does copy the extended parameter list, CMSCALL copies the parameter block, the command verb and the argument string. CMSCALL alters the addresses in the first 3 words of the EPLIST parameter block, mapped by the EPLIST macro, to reflect the new addresses. The remainder of the parameter block is not changed. Acceptable values are:

YES

specifies that CMSCALL copy the extended and tokenized parameter lists. This is the default value.

NO

specifies that CMSCALL does not copy the lists.

(reg)

the macro checks the value of the specified register and, if it is 0, sets COPY to NO. If the register contains a nonzero value, the macro sets COPY to YES.

(addr,mask)

defines a single bit in storage that sets the value of the COPY parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then COPY is set to NO. If the bit is 1, then COPY is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the COPY parameter as

```
COPY=(APPFLAG,X'80')
```

To set the value of the COPY parameter at assembly time, specify COPY=YES or COPY=NO. To set the value at execution time, specify COPY=(*reg*) or COPY=(*addr,mask*). If, at execution time, CMSCALL determines that COPY=NO, it ignores the MODIFY and FENCE parameters.

MODIFY=

specifies whether the tokenized parameter list is to be modified by the called program. If the called program modifies the parameter list, CMSCALL makes the same modifications to the original parameter list. The MODIFY parameter is valid only when COPY=YES. It is not valid if COPY=NO was specified at assembly time; it is ignored if COPY=NO is set at execution time. Acceptable values are:

NO

specifies the tokenized parameter list is not modified. This is the default value.

YES

specifies the tokenized parameter list is modified.

(*reg*)

the macro checks the value of the specified register and, if it is 0, sets MODIFY to NO. If the register contains a nonzero value, the macro sets MODIFY to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the MODIFY parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then MODIFY is set to NO. If the bit is 1, then MODIFY is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the MODIFY parameter as

```
MODIFY=(APPFLAG,X'80')
```

To set the value of the MODIFY parameter at assembly time, specify MODIFY=YES or MODIFY=NO. To set the value at execution time, specify MODIFY=(*reg*) or MODIFY=(*addr,mask*).

FENCE=

indicates whether the last token in the tokenized parameter list is the standard fence, which has a doubleword value of X'FF'. If FENCE=NO, CMSCALL copies the 68 doublewords beginning at the address of the tokenized parameter list. If FENCE=YES, CMSCALL copies everything up to and including the fence in the tokenized parameter list.

The FENCE parameter is valid only when COPY=YES. It is not valid if COPY=NO was specified at assembly time; it is ignored if COPY=NO is set at execution time. Acceptable values are:

YES

specifies the last token in the tokenized list as the standard fence. This is the default value.

NO

specifies the last token in the tokenized list is not the standard fence.

(*reg*)

the macro checks the value of the specified register and, if it is 0, sets FENCE to NO. If the register contains a nonzero value, the macro sets FENCE to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the FENCE parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then FENCE is set to NO. If the bit is 1, then FENCE is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the FENCE parameter as

```
FENCE=(APPFLAG,X'80')
```

To set the value of the FENCE parameter at assembly time, specify FENCE=YES or FENCE=NO. To set the value at execution time, specify FENCE=(*reg*) or FENCE=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

'ABEND'

abends the program.

Usage Notes

Call Charts: Table 8 on page 51 and Table 9 on page 51 summarize how SVC 202 and CMSCALL work. Note that CMSCALL always treats the address of the tokenized parameter list as a 31-bit address; SVC 202 always treats it as a 24-bit address.

Table 8. CMSCALL Call Chart		
Parameter List Location	AMODE of Program Being Called	Action to Parameter List
Below 16 MB	24	CMS copies the information specified on the CALLTYP parameter of CMSCALL into the high-order byte of register 1. This allows CMSCALL to call a routine that has not been changed (that is, a routine that expects information about the call in the high-order byte of register 1 instead of in the user save area).
Above 16 MB	24	Unless you code the COPY=NO parameter on the CMSCALL macro, CMS copies the parameter list below the 16 MB line. If you do not specify COPY=NO, CMS copies the information specified on the CALLTYP parameter of CMSCALL into the high-order byte of general register 1. If you do code the COPY=NO parameter on the CMSCALL macro, the program terminates with an abend code of X'1CC'.
Anywhere	31, ANY	Leave intact. If the caller is AMODE 24 and the callee is AMODE ANY, CMS copies the information specified on the CALLTYP parameter of CMSCALL into the high-order byte of general register 1.
Note: CMS passes register 0, which may contain the address of the extended parameter list, intact to the caller. It does not check to determine what type of address you pass unless you specify the COPY parameter on the CMSCALL macro.		

Table 9. SVC 202 Call Chart		
Callers Location	AMODE of Program Being Called	Action to Parameter List
Below 16 MB	24	Leave intact.
Above 16 MB	24, 31, ANY	Abend code X'1CA'—SVC 202 does not work from above 16 MB.

<i>Table 9. SVC 202 Call Chart (continued)</i>		
Callers Location	AMODE of Program Being Called	Action to Parameter List
Below 16 MB	31, ANY	CMS stores 0's in the high-order byte of register 1 in order to pass a 31-bit address.

PSW Settings When A Called Routine Starts: The following table shows how the PSW is set up when the called routine is entered.

<i>Table 10. PSW Settings When a Called Routine Starts</i>		
Call Mechanism - Target Program	Interrupts	Storage Key
CMSCALL - Nucleus Extension Module	Defined by NUCEXT macro	Defined by NUCEXT macro
CMSCALL - Transient Area Module	Disabled	Defined by GENMOD or SET PROTECT command
CMSCALL - User Area Module	Enabled	Defined by GENMOD or SET PROTECT command
SVC 202 - Nucleus Extension Module	Defined by NUCEXT macro	Defined by NUCEXT macro
SVC 202 - Transient Area Module	Disabled	Defined by GENMOD or SET PROTECT command
SVC 202 - User Area Module	Enabled	Defined by GENMOD or SET PROTECT command
User-defined	Disabled	User
Note: When a user defined SVC interrupt handler is invoked, the interrupt mask is disabled.		

Register Contents When A Called Routine Starts: The following table shows how the general registers are set up when the called routine is entered.

<i>Table 11. Register Contents When a Called Routine Starts</i>							
Type	Reg 0-1	Reg 2	Reg 3-11	Reg 12	Reg 13	Reg 14	Reg 15
CMSCALL	Same as caller	See note	Not defined	Address of called routine	Address of user save area	Return address	Address of called routine
SVC 202	Same as caller	See note	Not defined	Address of called routine	Address of user save area	Return address	Address of called routine
Other SVCs	Same as caller	Same as caller	Same as caller	Address of called routine	Address of user save area	Return address	Same as caller
Note: If the called routine is a nucleus extension or subcommand processor, then register 2 has the address of the SCBLOCK and the bit USESCBLK in USERSAVE is set to 1.							

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code
Meaning

-0015

A multitasking program was invoked while CMS/DOS mode was active.

-0014

SVC resulted in an implicitly created process that abended before completion.

-0006

An attempt was made to invoke a CMS function or macro from the command line or from a REXX EXEC with ADDRESS CMS or an EXEC 2 EXEC with &PRESUME &SUBCOMMAND. The function should be invoked from a program using SVC 202 or CMSCALL with a proper parameter list.

-0005

A LOADMOD was attempted with the wrong environment (for example, the module was generated by the GENMOD command with the OS option and LOADMOD was attempted with DOS=ON specified).

-0004

The LOADMOD failed (for example, there was an error with the module).

-0003

No CMS command, using name passed in parameter list, was found.

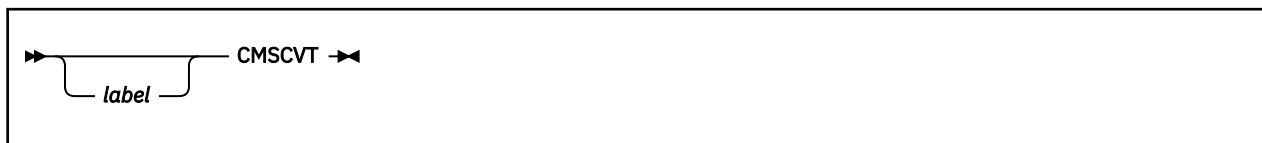
-0002

Error 32 on LOADMOD.

CMSCALL Abend Codes: [Table 12 on page 53](#) describes the CMSCALL abend codes.

<i>Table 12. ABEND Codes Specific to CMSCALL</i>			
ABEND code	Module name	Cause of ABEND	Action
0F0	DMSITS	Insufficient free storage is available to allocate a save area for a CMSCALL or SVC 202 call. Insufficient free storage is available to copy the parameter lists.	Define more storage
1CA	DMSITS	A program residing above 16 MB issued an SVC 202.	Change program to use CMSCALL, or move program below 16 MB.
1CB	DMSITS	A program residing above 16 MB issued an SVC 203.	Change program to use CMSCALL, or move program below 16 MB.
1CC	DMSITS	CMSCALL was used to invoke an AMODE 24 program with a parameter list above 16 MB.	Move the parameter list below 16 MB. This can be done using the COPY parameter on the CMSCALL macro.

CMSCVT



Purpose

Use the CMSCVT macro to generate a DSECT for the communications vector table.

Parameters

Optional Parameter:

label

is an optional assembler label for the statement. The first statement in the CMSCVT macro expansion is labeled CVTSECT.

Usage Notes

1. The CVTFLAG2 field indicates whether or not Data Compression Services and the hardware instruction (CMPSC) are supported. If Data Compression Services are supported, the CVTCMPSC bit will be on. If the machine supports hardware compression, the CVTCMPSH bit will be on.
2. The CMSCVT macroinstruction expands as follows:

```

CVTSECT  DSECT
*
***      COMMUNICATION VECTOR TABLE AS SUPPORTED BY CMS
*
          DC      H'0' -      RESERVED
CVTMDDL  DC      H'0' -      CPU MODEL ID
          DC      CL4'CSPR'    VM SYSTEM PRODUCT RELEASE
*
***      END OF CVT PREFIX AREA      **
*
CMSCVT   DS      0D -      CVT START
          DC      V(DMSNUCEL)  Simulated CEL Anchor linkage
          DC      F'-1' -      NOT SUPPORTED
CVTLINK   DC      F'-1' -      RESERVED
          DC      11F'-1' -    NOT SUPPORTED
CVTDATE   DC      PL4'0' -    CURRENT DATE IN PACKED DECIMAL
          DC      3F'-1' -    NOT SUPPORTED
          DC      A(0) -      NOT SUPPORTED
CVTVPRM   DS      0F -      VECTOR FACILITY PARAMETERS
CVTVSS    DC      H'0' -      VECTOR SECTION SIZE
CVTVPSM   DC      H'0' -      VECTOR PARTIAL SUM NUMBER
CVTEXIT   DC      XL2'0A03' -  AN SVC 3 INSTRUCTION (EXIT)
CVTBRET   DC      XL2'07FE' -  A BCR 15,14 INSTRUCTION
          DC      8F'-1' -    NOT SUPPORTED
CVTDCB    DC      AL1(CVTMVSE+CVT1SSS+CVTOSEXT) System is XA+CMS
CVTMVSE   EQU     X'80' -      S/370-XA mode execution
CVT1SSS   EQU     X'40' -      Option 1 (PCP) SSS also CMS
CVT2SPS   EQU     X'20' -      Option 2 (MFT) or VSE on VM
CVTOSEXT   EQU     X'08' -      indicator that the CVT0SLVL area
*                                     is present and may be referenced.

```

```

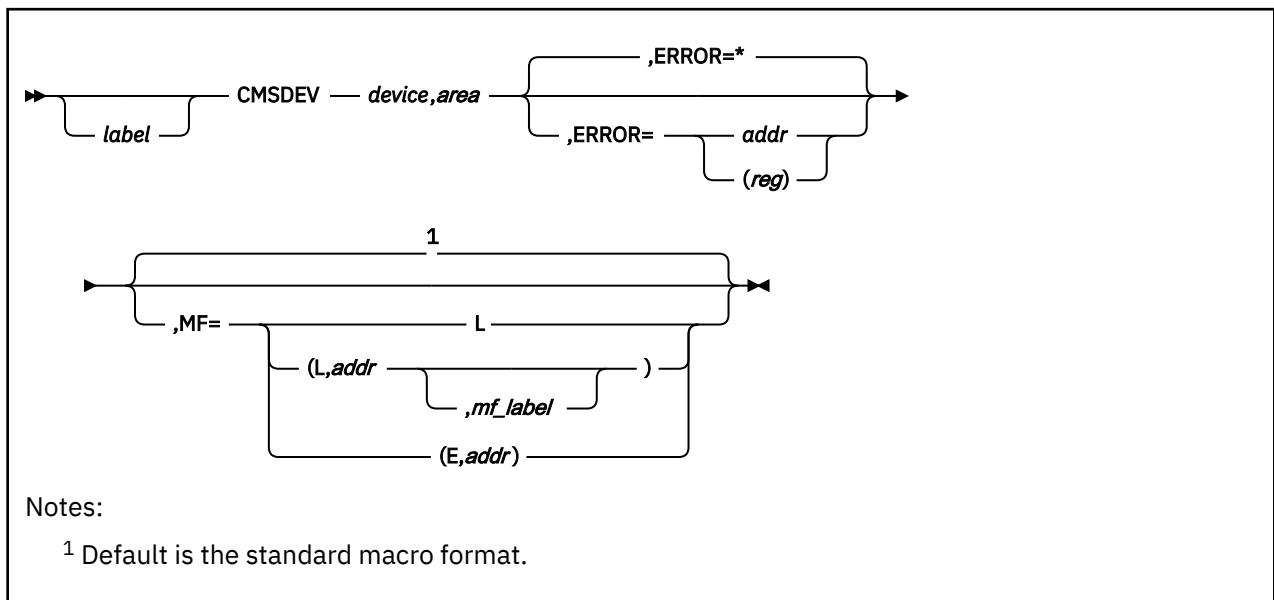
CVTR13    DC      FL3'-1' -    NOT SUPPORTED
          DC      F'0' -      R13 SAVED DURING 'OPEN'
          DC      F'-1' -    NOT SUPPORTED
CVTNUCB    DC      A(0) -      RESERVED
          DC      2F'-1' -    NOT SUPPORTED
CVTECVT    DC      A(0) -      ADDR OF EXTENDED CVT
          DC      5F'-1' -    NOT SUPPORTED
CVTMZ00    DC      A(0) -      HIGHEST STORAGE ADDRESS IN MACHINE
          DC      3F'-1' -    NOT SUPPORTED
          DC      XL2'00' -    NOT SUPPORTED
CVTOPTA    DC      XL2'00' -    BIT 7 - EXT-PREC FP HRDWRE IN CPU

```

	DC	2F'-1' -	NOT SUPPORTED
	DC	2A(0) -	NOT SUPPORTED
CVTABEND	DC	V(CMSSCVT)	ADDR OF SECONDARY CVT
CVTUSER	DC	F'0' -	FIELD AVAILABLE TO USER
	DC	7F'-1' -	NOT SUPPORTED
CVTGTf	DC	F'-1' -	GENERALIZED TRACE FACILITY
	DC	2F'-1' -	NOT SUPPORTED
CVTSAF	DC	A(0)	ADDR OF SAF VECTOR TABLE
	DC	F'-1' -	NOT SUPPORTED
CVTACBM	DC	V(DMSCBM)	ADDR OF CBMM ROUTINE
	DC	11F'-1' -	NOT SUPPORTED
CVTTZ	DC	XL4'00'	DIFFERENCE BETWEEN LOCAL TIME
*			AND GREENWICH MEAN TIME
	DC	5F'-1' -	-NOT SUPPORTED-
CVTEXT2	DC	A(0)	ADDR OF OS/VS2 COMMON EXTENSION
	DC	11F'-1' -	-NOT SUPPORTED-
CVTFLAGS	DS	0A -	SYSTEM GLOBAL FLAGS
CVTFLAG1	DC	AL1(0) -	-NOT SUPPORTED-
CVTFLAG2	DC	AL1(0) -	FLAG BYTE 2
CVTCMPSC	EQU	X'80' -	Compression Services supported
CVTCMPSH	EQU	X'40' -	'CMPSC' HW instruction available
CVTFLAG3	DC	X'00' -	-NOT SUPPORTED-
CVTFLAG4	DC	X'00' -	-NOT SUPPORTED-
	DC	23F'-1' -	-NOT SUPPORTED-
*			
CVTSTCK	DC	F'0'	TSO STACK ADDR
	DC	74F'-1' -	-NOT SUPPORTED-
CVTXSFT	DC	A(0)	ADDR OF SYSTEM FUNCTION TABLE
	DC	92F'-1' -	-NOT SUPPORTED-
CVTSREGN	DC	A(0)	ADDR OF VSM REGION SIZE ROUTINE
	DC	17F'-1' -	-NOT SUPPORTED-
CVTDFA	DC	A(0)	ADDR OF DFP ID TABLE
	DC	11F'-1' -	-NOT SUPPORTED-

*			
*			
CVTOSLVL	DS	0XL16	SYSTEM LEVEL INDICATORS
*			BYTE 0 OF CVTOSLVL has...
CVTOSLV0	DC	AL1(CVTXAX+CVTCADS)	System ESA & Data Spaces
CVTXAX	EQU	X'80'	'EXTENDED XA' = ESA/370 SUPPORTED
CVTHIPER	EQU	X'10'	HIPERSPACES ARE SUPPORTED
CVTCADS	EQU	X'04'	COMMON DATA SPACES SUPPORTED
*			
CVTOSLV1	DC	AL1(0)	BYTE 1 OF CVTOSLVL
CVTOSLV2	DC	AL1(0)	BYTE 2 OF CVTOSLVL
CVTOSLV3	DC	AL1(0)	BYTE 3 OF CVTOSLVL
CVTOSLV4	DC	AL1(0)	BYTE 4 OF CVTOSLVL
CVTOSLV5	DC	AL1(0)	BYTE 5 OF CVTOSLVL
CVTOSLV6	DC	AL1(0)	BYTE 6 OF CVTOSLVL
CVTOSLV7	DC	AL1(0)	BYTE 7 OF CVTOSLVL
CVTOSLV8	DC	AL1(0)	BYTE 8 OF CVTOSLVL
CVTOSLV9	DC	AL1(0)	BYTE 9 OF CVTOSLVL
CVTOSLVA	DC	AL1(0)	BYTE 10 OF CVTOSLVL
CVTOSLVB	DC	AL1(0)	BYTE 11 OF CVTOSLVL
CVTOSLVC	DC	AL1(0)	BYTE 12 OF CVTOSLVL
CVTOSLVD	DC	AL1(0)	BYTE 13 OF CVTOSLVL
CVTOSLVE	DC	AL1(0)	BYTE 14 OF CVTOSLVL
CVTOSLVF	DC	AL1(0)	BYTE 15 OF CVTOSLVL
*			OS/VS2 COMMON EXTENSION
*			ADDRESS OF EXTENSION IS IN CVTREXT2
CVTXTNT2	DSECT		
CVT2R000	DC	CL4'EXT2'	RESERVED - EYECATCHER
	DC	13F'-1' -	-NOT SUPPORTED-
CVTLDT0	DS	0D	LOCAL TIME/DATE OFFSET
CVTLDT0L	DC	F'0'	HIGH WORD
CVTLDTOR	DC	F'0'	LOW WORD
	DC	17F'-1' -	-NOT SUPPORTED-

CMSDEV



Purpose

Use the CMSDEV macroinstruction to obtain the characteristics of a virtual device. CMS returns the results to a specified storage area.

Parameters

Required Parameters:

device

specifies the virtual device whose characteristics CMSDEV obtains. It may be one of the following:

CONS

a virtual console.

PRT

the virtual printer.

RDR

the virtual reader.

PUN

the virtual punch.

TAP n

a tape device attached to your virtual machine. Valid values for n are X'0' to X'1F'.

vdev

a hexadecimal address of a virtual device attached to your virtual machine.

(reg)

a register (2-12) containing the device address in the low-order two bytes.

area

is the name of a 12-byte storage area to contain the device information. It may be one of the following:

addr

an assembler program label for the address of the storage area.

(reg)

a specified register (2-12) containing the address of the storage area.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. Use the CMSDEV macro with the PRINTL macro to obtain the device characteristics of the virtual printer. This avoids the need to perform either a DIAGNOSE code X'24' or a DIAGNOSE code X'210' each time you want to write to the same virtual printer. The CMSDEV macro is an easier way to get the information generated by the Diagnose instructions.
2. When the CMSDEV macro completes, the defined 12-byte storage area contains the device characteristics.

If the virtual device exists, the first 4 bytes contain:

Bytes

Virtual Device Information

0

Type class

1

Type

2

Status

3

Flags

If the virtual device is associated with a local real device, bytes 4 through 7 contain:

Bytes

Local Real Device Information

- 4** Type class
- 5** Type
- 6** Model number
- 7** Current device line length for a virtual console, or the device feature code for other devices.

If the virtual device is associated with a remote real device, bytes 4 through 7 contain:

Bytes

Remote Real Device Information

- 4** Type class
- 5** Type for a remote 3270 console
- 6** Model number for a remote 3270 console
- 7** Current device line length for a remote virtual console.

If the virtual device is a local virtual console or a remote 3270 virtual console (*device* specified as CONS), bytes 8 through 11 contain:

Byte

Information

- 8** The terminal code bits defining the type of virtual console and the translate table the console uses.
- 9** Reserved
- 10-11** Virtual device number

For virtual devices other than CONS, bytes 8 through 11 contain:

Bytes

Information

- 8** Reserved
- 9** Reserved
- 10-11** Virtual device number

For more information on DIAGNOSE codes X'24' or X'210', and device information see [z/VM: CP Programming Services](#).

Return Codes

When the CMSDEV macro completes, register 15 contains one of the following return codes:

Code

Meaning

- 0** The virtual device is attached and a real device is associated with it.

- 1** The virtual device is attached and a real device is **not** associated with it. This is normal for spooled devices.
- 2** The virtual device is not attached or an invalid device address was specified.

CMSECVT



Purpose

Use the CMSECVT macro to generate a DSECT for the extended communication vector table.

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the CMSECVT macro expansion is labeled ECVTSECT.

Usage Notes

1. The CMSECVT macroinstruction expands as follows: ECVTSECT DSECT

```
*
*** Extended Communication Vector Table (ECVT) as supported by CMS
*
CMSECVT DS    0D          ECVT start
          DC    CL4'ECVT' Eyecatcher
          DC    59F'-1'   Not supported - reserved for IBM use
ECVTOCVT DS    0A          Anchor for OpenMVS CVT
ECVTOMVS DC    XL1'1'     OpenMVS Feature Bit
          DC    XL3'0'
ECVTOEXT DC    A(0)       Anchor for OpenMVS External data
ECVTCMPS DC    A(0)       Addr of Compression Services routine
*                               Either -
*                               A(DMSCMSSH + X'80000000') Hardware co
*                               A(DMSCMSSS + X'80000000') Software co
          DC    17F'-1'   Not supported - reserved for IBM use
ECVTLENB EQU    (*-ECVTSECT) Length in bytes of ECVTSECT
ECVTLEND EQU    ((ECVTLENB+7)/8) Length in dwords of ECVTSECT
```

2. The system compression routine is found through the CMSCVT and CMSECVT tables. Once the CSRCMPSC macro is invoked, it branches to the correct service entry point to do the data compression or expansion.
3. If Data Compression Services are supported, the pointer entry for ECVTCMPS (for the compression services routine address) will be set to either:
 - DMSCMSSH, if the hardware instruction is supported or
 - DMSCMSSS, for the software simulation of the hardware compression feature, since the hardware instruction is not supported.

An example of this is:

```
A(DMSCMSSH + X'80000000') - Hardware compression
A(DMSCMSSS + X'80000000') - Software compression
```


CMSIUCV

Purpose

Use the CMSIUCV macro to start or end communications with another program in an IUCV (Inter-User Communications Vehicle) or APPC/VM (Advanced Program-to-Program Communications/VM) environment.

The basic functions of the CMSIUCV macro are:

CMSIUCV ACCEPT

Accepts the connection from a requesting program to complete a path, and notifies CMS.

CMSIUCV CONNECT

Establishes and reserves a path to communicate with another program, and lets CMS know about the connection.

CMSIUCV QCMSWID

Gets the current CMS work unit identifier associated with a path.

CMSIUCV RESOLVE

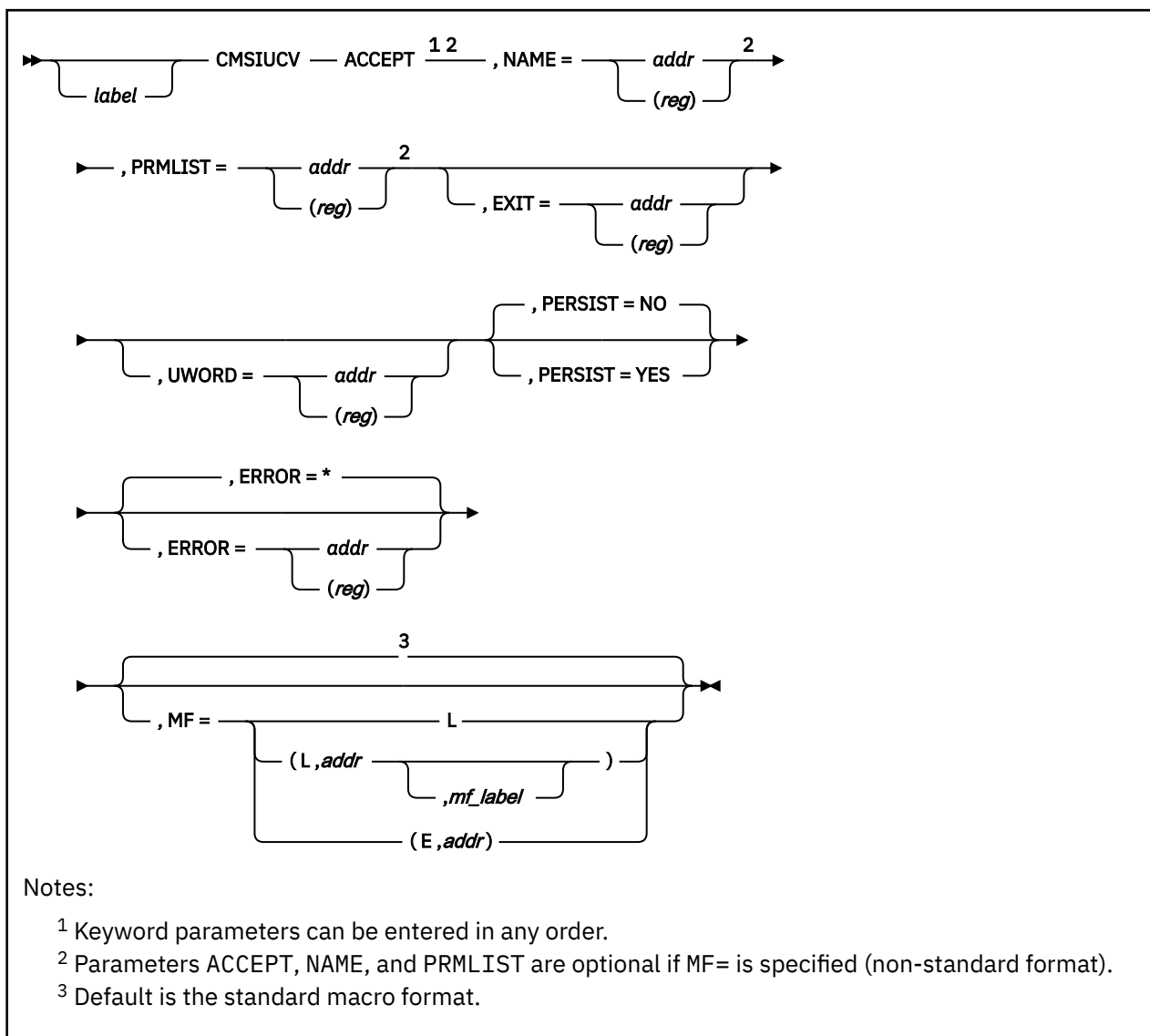
Gets values from a CMS communications directory file for examination.

CMSIUCV SEVER

Ends communications with another program, and lets CMS know about it.

For more information on how to use the CMSIUCV macro, see the [*z/VM: CMS Application Development Guide for Assembler*](#).

CMSIUCV ACCEPT



Purpose

Use the `ACCEPT` function of the `CMSIUCV` macro to request that CMS perform an `ACCEPT`.

Before issuing this function, an `IUCV ACCEPT` parameter list must be set up by the program and passed to CMS using the `MF=L` operand on the `IUCV ACCEPT` macro.

Parameters

Required Parameters:

ACCEPT

Accepts the connection from a requesting program to complete a path, and notifies CMS.

NAME=

specifies the name that identifies the program associated with this path. A program with this name must have previously issued an `HNDIUCV SET` function to identify itself as an `APPC/VM` program to CMS.

addr

specifies the address of an 8 character symbolic name.

(reg)

specifies a register that contains the address of the 8 character symbolic name.

If the program requests an ACCEPT for a specific path and the NAME specified does not correspond with the owner of that path, the ACCEPT is not permitted.

PRMLIST=

specifies the storage address that contains the IUCV ACCEPT parameter list. Your program must prepare this parameter list before it issues the CMSIUCV ACCEPT. To prepare the parameter list, your program must use the list form (MF=L) of the IUCV ACCEPT macro. (This lets you set up the IUCV parameter list using macro keyword parameters instead of storing information with IPARML DSECT labels.) The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

addr

specifies the address of the program's IUCV parameter list.

(reg)

specifies a register that contains the address of the program's IUCV parameter list.

Optional Parameters:

label

is an optional assembler label for the statement.

EXIT=

specifies the address of an exit routine to receive control whenever an APPC/VM external interrupt occurs on this APPC/VM path. If you do not specify EXIT, the exit address defaults to the address specified on the HNDIUCV macro for this program.

APPC/VM exit routines are called in the addressing mode (24- or 31-bit) of the program that issues CMSIUCV ACCEPT.

addr

specifies an assembler program label as the address of the exit routine.

(reg)

specifies a register that contains the address of the exit routine.

When the program's APPC/VM external interrupt routine is given control, all interrupts are disabled. The exit routine is responsible for providing proper entry and exit linkage for its APPC/VM external interrupt handling routine. The exit routine:

- Should not enable itself for any type of interrupts.
- Should not perform any I/O operations, because all interrupts are disabled.
- Must return control to the address in register 14.

When the routine receives control, the significant registers contain:

Register	Contents
0	UWORD Field

Register	Contents																												
1	<p>If the pending interrupt is for a private resource connection, register 1 contains a X'00'.</p> <p>If a connection to a global or local resource, register 1 points to a SAVEAREA in this format:</p> <table><tr><th>Label</th><th colspan="2">Displacement</th><th>Contents</th></tr><tr><td></td><th>Dec</th><th>Hex</th><td></td></tr><tr><td>GRS</td><td>0</td><td>0</td><td>General purpose registers 0-15 at the time of the interrupt.</td></tr><tr><td>FRS</td><td>64</td><td>40</td><td>Floating point registers 0-7 at the time of the interrupt.</td></tr><tr><td>PSW</td><td>96</td><td>60</td><td>External Old PSW at the time of the interrupt.</td></tr><tr><td>UAREA</td><td>104</td><td>68</td><td>Register save area for exit routine's use.</td></tr><tr><td>END</td><td>176</td><td>B0</td><td>End of save area.</td></tr></table>	Label	Displacement		Contents		Dec	Hex		GRS	0	0	General purpose registers 0-15 at the time of the interrupt.	FRS	64	40	Floating point registers 0-7 at the time of the interrupt.	PSW	96	60	External Old PSW at the time of the interrupt.	UAREA	104	68	Register save area for exit routine's use.	END	176	B0	End of save area.
Label	Displacement		Contents																										
	Dec	Hex																											
GRS	0	0	General purpose registers 0-15 at the time of the interrupt.																										
FRS	64	40	Floating point registers 0-7 at the time of the interrupt.																										
PSW	96	60	External Old PSW at the time of the interrupt.																										
UAREA	104	68	Register save area for exit routine's use.																										
END	176	B0	End of save area.																										
2	Address of the APPC/VM External Interrupt Buffer																												
3	Address of the connection pending extended data (if the exit is driven by a connection pending interrupt), or the address of the connection complete extended data (if the exit is driven by a connection complete interrupt).																												
4	Address of the PIP variable (if the exit is driven by a connection pending interrupt).																												
13	Points to the register save area at label UAREA for use by the exit routine. (If register 1 contains a X'00', register 13 points to a standard register save area.)																												
14	Return address																												
15	Entry point address																												

UWORD=

specifies an optional fullword that the invoking program can pass to the exit routine for any purpose desired. When the exit routine receives control, register 0 contains either an address (if **UWORD=addr**) or the value of the register (if **UWORD=(reg)**). If **UWORD** is not specified here, it is set to zero.

If you do not specify **UWORD** here, the **UWORD** value defaults to the value specified on the **HNDIUCV** macro for this program.

addr

specifies the address where the **UWORD** value is stored.

(reg)

specifies a register that contains the **UWORD** value.

PERSIST=

lets you specify whether an APPC/VM conversation is deallocated when work unit processing completes. Work unit processing completes at end-of-command, end-of-subset, or when **DMSPURWU** (purge work unit) or **DMSRETWU** (return work unit) routines are issued.

NO

specifies that CMS will automatically deallocate the APPC/VM conversation when work unit processing completes.

YES

specifies that CMS will not automatically deallocate the APPC/VM conversation when work unit processing completes. This cannot be used with protected conversations (**SYNCLVL=SYNCPT**).

Note: When you use **MF=(E,addr)**, the execute format of **CMSIUCV ACCEPT**, there is no default for the **PERSIST** parameter.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

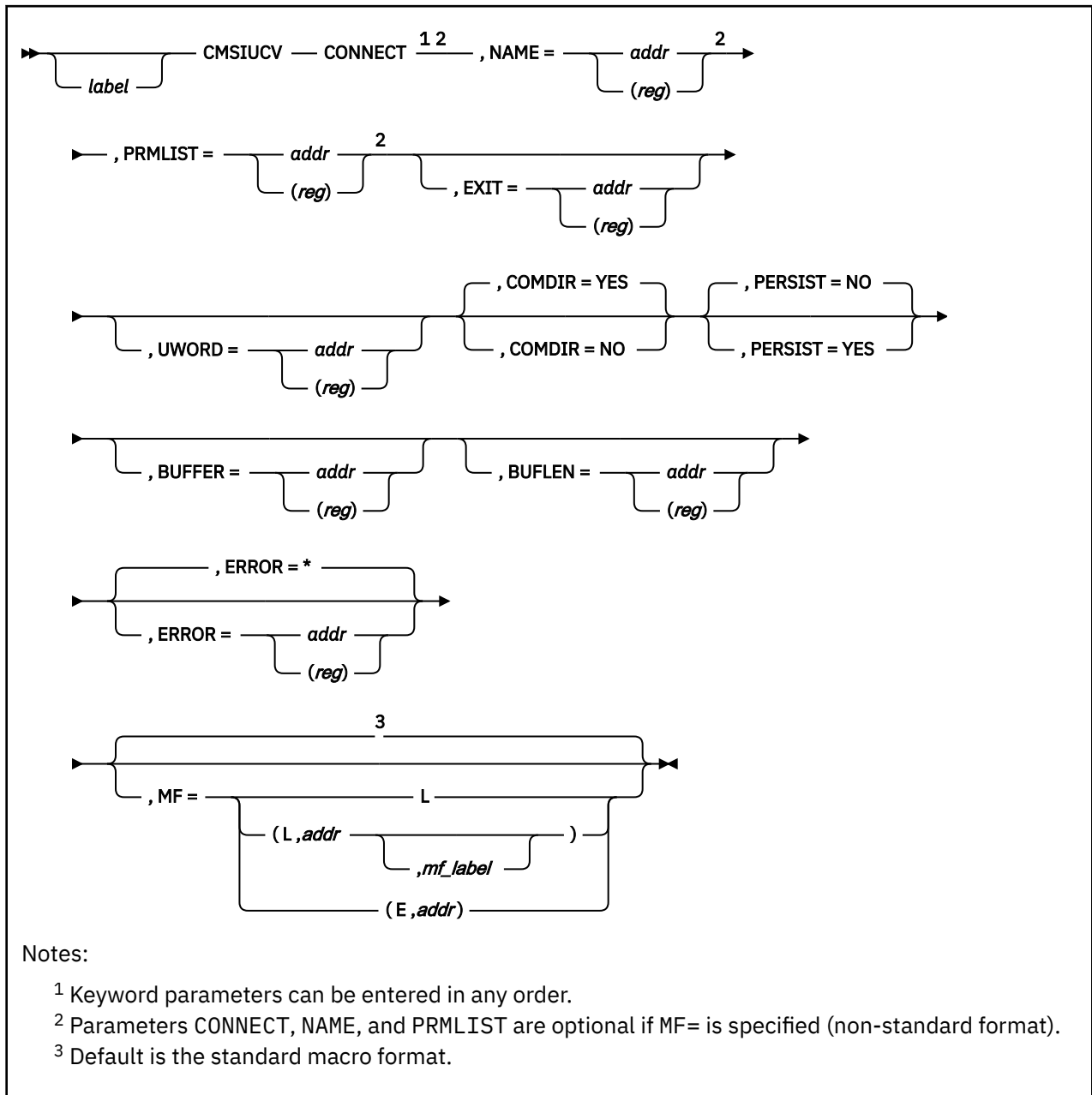
Upon completion of the CMSIUCV ACCEPT function, register 15 contains either:

- A 5-digit reason code returned by a CSL routine that was called by CMSIUCV ACCEPT processing. These are described in *z/VM: CMS and REXX/VM Messages and Codes*, or
- One of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	CMSIUCV ACCEPT completed successfully. (For a protected conversation this path may have been previously accepted by CMS on behalf of the application. Only the path's exit address and user word were updated (if appropriate). The actual ACCEPT was not reflected to CP.)
X'02'	2	An IUCV parameter list was passed as input to the CMSIUCV ACCEPT, and the ACCEPT completed immediately. The function complete information is in the parameter list. The user's path-specific exit is not called because CP does not reflect an interrupt to the virtual machine. (This reflects a CC=2 for the IUCV ACCEPT.)
X'08'	8	No HNDIUCV SET has been issued for this program.
X'0C'	12	The program does not own the path.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'18'	24	The PRMLIST parameter was not specified or its address is equal to 0.

Hex Code	Decimal Code	Meaning
X'28'	40	An invalid CMSIUCV function was specified; must be CONNECT, ACCEPT, SEVER, RESOLVE, or QCMSWID.
X'46'	70	PERSIST=YES was specified for a SYNCLVL=SYNCPT conversation.
X'3E8' + xxx	1ddd	Indicates that an APPC/VM error occurred. The xxx is the IPRCODE field that was returned by the IUCV ACCEPT macro to aid in diagnosing the error. The ddd is the decimal equivalent of this IPRCODE value. For more information on the IUCV ACCEPT return codes, see z/VM: CP Programming Services .

CMSIUCV CONNECT



Purpose

Use the CONNECT function of the CMSIUCV macro to request that CMS perform a CONNECT.

Before issuing this function, the program must set up an APPCVM CONNECT parameter list and pass it to CMS.

Parameters

Required Parameters:

CONNECT

Establishes and reserves a path to communicate with another program, and lets CMS know about the connection.

NAME=

specifies the program name associated with this connection path. A program with this name must have previously issued an HNDIUCV SET to identify itself as an APPC/VM program to CMS.

addr

specifies the address of an 8 character program name.

(reg)

specifies a register that contains the address of the 8-character program name.

PRMLIST=

specifies the storage address that contains the APPCVM CONNECT parameter list. Your program must prepare this parameter list before it issues the CMSIUCV CONNECT. To prepare the parameter list, your program must use the list form (MF=L) of the APPCVM CONNECT macro. (This lets you set up the APPC/VM parameter list using macro keyword parameters instead of storing information with IPARML DSECT labels.) The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

addr

specifies the address of the program's APPC/VM parameter list.

(reg)

specifies a register that contains the address of the program's APPC/VM parameter list.

Optional Parameters:

label

is an optional assembler label for the statement.

EXIT=

specifies the address of an exit routine to receive control whenever an APPC/VM external interrupt occurs on this APPC/VM path. If you do not specify EXIT, the exit address defaults to the address specified in the HNDIUCV macro for this program.

APPC/VM exit routines are called in the addressing mode of the program that issues this CMSIUCV CONNECT.

addr

specifies an assembler program label as the address of the exit routine.

(reg)

specifies a register that contains the address of the exit routine.

When the program's APPC/VM external interrupt routine is given control, all interrupts are disabled. The exit routine is responsible for providing proper entry and exit linkage for its APPC/VM external interrupt handling routine. The exit routine:

- Should not enable itself for any type of interrupts.
- Should not perform any I/O operations, because all interrupts are disabled.
- Must return control to the address in register 14.

When the routine receives control, the significant registers contain:

Register	Contents
0	UWORD Field

Register	Contents																												
1	<p>If the pending interrupt is for a private resource connection, register 1 contains a X'00'.</p> <p>If a connection to a global or local resource, register 1 points to a SAVEAREA in this format:</p> <table><tr><th>Label</th><th colspan="2">Displacement</th><th>Contents</th></tr><tr><td></td><th>Dec</th><th>Hex</th><td></td></tr><tr><td>GRS</td><td>0</td><td>0</td><td>General purpose registers 0-15 at the time of the interrupt.</td></tr><tr><td>FRS</td><td>64</td><td>40</td><td>Floating point registers 0-7 at the time of the interrupt.</td></tr><tr><td>PSW</td><td>96</td><td>60</td><td>External Old PSW at the time of the interrupt.</td></tr><tr><td>UAREA</td><td>104</td><td>68</td><td>Register save area for exit routine's use.</td></tr><tr><td>END</td><td>176</td><td>80</td><td>End of save area.</td></tr></table>	Label	Displacement		Contents		Dec	Hex		GRS	0	0	General purpose registers 0-15 at the time of the interrupt.	FRS	64	40	Floating point registers 0-7 at the time of the interrupt.	PSW	96	60	External Old PSW at the time of the interrupt.	UAREA	104	68	Register save area for exit routine's use.	END	176	80	End of save area.
Label	Displacement		Contents																										
	Dec	Hex																											
GRS	0	0	General purpose registers 0-15 at the time of the interrupt.																										
FRS	64	40	Floating point registers 0-7 at the time of the interrupt.																										
PSW	96	60	External Old PSW at the time of the interrupt.																										
UAREA	104	68	Register save area for exit routine's use.																										
END	176	80	End of save area.																										
2	Address of the APPC/VM External Interrupt Buffer																												
3	Address of the connection pending extended data (if the exit is driven by a connection pending interrupt), or the address of the connection complete extended data (if the exit is driven by a connection complete interrupt).																												
4	Address of the PIP variable (if the exit is driven by a connection pending interrupt).																												
13	Points to the register save area at label UAREA for use by the exit routine. (If register 1 contains a X'00', register 13 points to a standard register save area.)																												
14	Return address																												
15	Entry point address																												

UWORD=

specifies a fullword (user word) containing information that the invoking program can specify. CMS passes this user word to the exit routine when an interrupt is presented for this APPC/VM path. The exit routine can use this information if it desires to do so. When the exit routine receives control, register 0 contains either an address where the user word is stored (if **UWORD=addr**) or the value of a register that contains the user word (if **UWORD=(reg)**).

If you do not specify **UWORD** here, the user word value defaults to the value specified on the **HNDIUCV SET** macro for this program name.

addr

specifies the address where the user word value is stored.

(reg)

specifies a register that contains the user word value.

COMDIR=

indicates whether you want communications directory resolution to be performed.

YES

indicates that communications directory resolution will be performed conditionally, depending on the setting specified by the **SET COMDIR** command. For more information on **SET COMDIR**, see the *z/VM: CMS Commands and Utilities Reference*. This causes the **RESID** parameter on the **APPCVM CONNECT** to map to values in a CMS communications directory file. CMS uses these values to fill in the connection parameter list extension for you, transparently.

If the connection parameter list extension contains allocate data (**FMH5** is specified on the **APPCVM CONNECT**), communications directory resolution is disabled.

If you omit the COMDIR= parameter, it defaults to YES. If you are using the execute form (MF=E) of the CMSIUCV CONNECT macro and omit COMDIR=, the macro will not use YES if NO is stored in the COMDIR field of the CMSIUCV parameter list.

See Usage Note “2” on page 71, which describes the process used by CMS to perform communications directory resolution.

NO

indicates that communications directory resolution is not performed.

Note: When you use the MF=(E,*addr*), the execute format of CMSIUCV CONNECT, there is no default for the COMDIR parameter.

PERSIST=

lets you specify whether an APPC/VM conversation is deallocated when work unit processing completes. Work unit processing completes at end-of-command, end-of-subset, or when DMSPURWU (purge work unit) or DMSRETWU (return work unit) routines are issued.

NO

specifies that CMS will automatically deallocate the APPC/VM conversation when work unit processing completes.

YES

specifies that CMS will not automatically deallocate the APPC/VM conversation when work unit processing completes. This cannot be used with protected conversations (SYNCLVL=SYNCPT).

Note: When you use the MF=(E,*addr*), the execute format of CMSIUCV CONNECT, there is no default for the PERSIST parameter.

BUFFER=

specifies the address of the user supplied location where connection complete extended data (CCED) will be placed when APPCVM CONNECT completes with CC=2 or CC=3.

addr

specifies a label in storage for the address.

(*reg*)

specifies a register that contains the address where the CCED data will be moved.

BUFLN=

specifies the address of the user-supplied location that contains the length of BUFFER.

addr

specifies the label of the fullword containing the length.

(*reg*)

specifies a register that contains the address of the location containing the BUFFER length.

If the BUFLN value is smaller than the length of the CCED, as much of the CCED data as will fit is copied into the buffer. If the CCED does not fill the BUFFER area allocated for it, the unused portion is undefined.

Note:

1. If either BUFFER or BUFLN is specified, the other must also be specified.
2. If neither BUFFER nor BUFLN is specified and APPCVM CONNECT completed with CC=2 or CC=3, the application will not get the CCED.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The first 8 characters in the IPUSER field of the IUCV parameter list must contain the name specified in the target virtual machine's HNDIUCV SET. For more information, see the [z/VM: CMS Application Development Guide for Assembler](#).
2. When CMS resolves a symbolic destination name (the RESID= parameter on APPCVM CONNECT), the user-level communications directory file (if it exists) is checked first. If the user-level communications directory does *not* contain the specified symbolic destination name, CMS searches the system-level communications directory. If the same symbolic destination name is defined in both the user and system levels, the information in the system-level file is ignored. Lastly, when a symbolic destination name is found in neither the user-level file nor the system-level file, the IBM-level communication directory file, ICOMDIR NAMES, is searched.

Note: The communications directory entries in the IBM-level file are intended for use by IBM and should not be changed.

If the resource identified in the connection request does *not* match a symbolic destination name defined in any of the CMS communications directories, then the connection request is processed using the specified resource ID as the name of a resource located in the same TSAF or CS collection as the user program issuing CMSIUCV CONNECT.

Return Codes

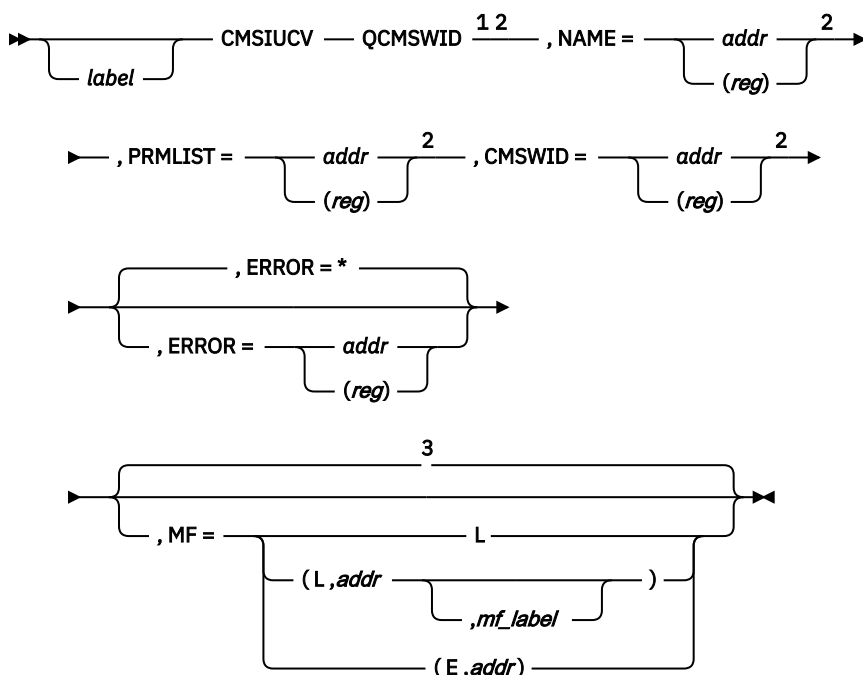
Upon completion of the CMSIUCV ACCEPT function, register 15 contains either:

- A 5-digit reason code returned by a CSL routine that was called by CMSIUCV ACCEPT processing. These are described in [z/VM: CMS and REXX/VM Messages and Codes](#), or
- One of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	For an IUCV CONNECT, your function completed normally. For an APPCVM CONNECT, the CONNECT started successfully, but has not completed. For more information on IUCV CONNECT and APPCVM CONNECT, see the <i>z/VM: CP Programming Services</i> (This corresponds to a CC=0 for either APPCVM CONNECT or IUCV CONNECT.)
X'02'	2	An APPC/VM parameter list was passed as input to the CMSIUCV CONNECT, and the APPCVM CONNECT completed immediately. The function complete information is in the parameter list. The user's path-specific exit is not called because CP does not reflect an interrupt to the virtual machine. (This corresponds to a CC=2 for the APPCVM CONNECT.)
X'03'	3	An APPC/VM parameter list was passed as input to the CMSIUCV CONNECT, and the APPCVM CONNECT completed immediately. CP stored error information related to PIP data in the IPAUDIT field of the CP APPC/VM parameter list and/or there was truncation of the CCED. The user's path-specific exit is not called because CP does not reflect an interrupt to the virtual machine. (This corresponds to a CC=3 for the APPCVM CONNECT.)
X'08'	8	No HNDIUCV SET has been issued for this program.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'18'	24	The PRMLIST parameter was not specified or its address is equal to 0.
X'1E'	30	The LUWID could not be obtained because of a CSL error (for SYNCLVL=SYNCPT).
X'26'	38	The work unit is not in a valid state for issuing CMSIUCV CONNECT (for SYNCLVL=SYNCPT).
X'28'	40	An invalid CMSIUCV function was specified; it must be CONNECT, ACCEPT, SEVER, RESOLVE, or QCMSWID.
X'2E'	46	Either the BUFFER or BUFLen parameter was specified without the other.
X'32'	50	Initialization for CSL support for CMS communication failed.
X'34'	52	The APPCVM CONNECT parameter list is invalid—only the reserved username, !CMS, can specify CONTROL=YES. (!CMS is a reserved name for CMS. CMS uses !CMS as a user ID so it can use its own APPC/VM support.)
X'36'	54	SYNCLVL=SYNCPT is not allowed on a control path.
X'46'	70	PERSIST=YES was specified for a SYNCLVL=SYNCPT conversation.
X'4A'	74	CP support for coordinated resource recovery is not available.
X'5C'	92	An invalid security tag field was found in the communications directory.

Hex Code	Decimal Code	Meaning
X'68'	104	Out of storage.
X'3E8' + xxx	1ddd	Indicates that an APPC/VM or IUCV error occurred. The xxx is the IPRCODE field that was returned by APPCVM CONNECT or IUCV CONNECT to aid in diagnosing the error; the ddd is the decimal equivalent of this IPRCODE value. (This corresponds to a CC=1 for the APPC/VM Connect.) For more information on APPCVM CONNECT and IUCV CONNECT return codes, see the z/VM: CP Programming Services .

CMSIUCV QCMSWID



Notes:

- ¹ Keyword parameters can be entered in any order.
- ² Parameters `QCMSWID`, `NAME`, `PRMLIST`, and `CMSWID` are optional if `MF=` is specified (non-standard format).
- ³ Default is the standard macro format.

Purpose

Use the `QCMSWID` function of the `CMSIUCV` macro to query the CMS work unit ID associated with the path ID given in `PRMLIST`.

Parameters

Required Parameters:

QCMSWID

Gets the current CMS work unit identifier associated with a path.

NAME=

specifies the program name associated with this path. A program with this name must have previously issued an `HNDIUCV SET` to identify itself as an `APPC/VM` program to CMS.

addr

specifies the address of an 8 character program name.

(reg)

specifies a register that contains the address of the 8 character program name.

PRMLIST=

specifies the address of the storage area containing the `APPC/VM` or `IUCV` parameter list. Your program must prepare this parameter list before it issues the `CMSIUCV QCMSWID`. (This is the parameter list your program should have prepared using the list form (`MF=L`) of the `APPCVM CONNECT` or `IUCV ACCEPT` macro.) The address must be a guest real address, that is, the address

must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

addr

specifies the address of the program's parameter list.

(reg)

specifies a register that contains the address of the program's parameter list.

CMSWID=

specifies the address of the 4-byte user-defined location where the CMS work unit ID for this path ID will be moved into. For an outbound connection, this CMS work unit ID is the current work unit ID when the CONNECT function was initiated for this path. For an inbound connection, this CMS work unit ID is the work unit ID obtained during connection pending interrupt processing for this path.

addr

specifies the label in storage of the location where the CMS work unit ID will be stored.

(reg)

specifies a register that contains the address of the location where the CMS work unit ID will be stored.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

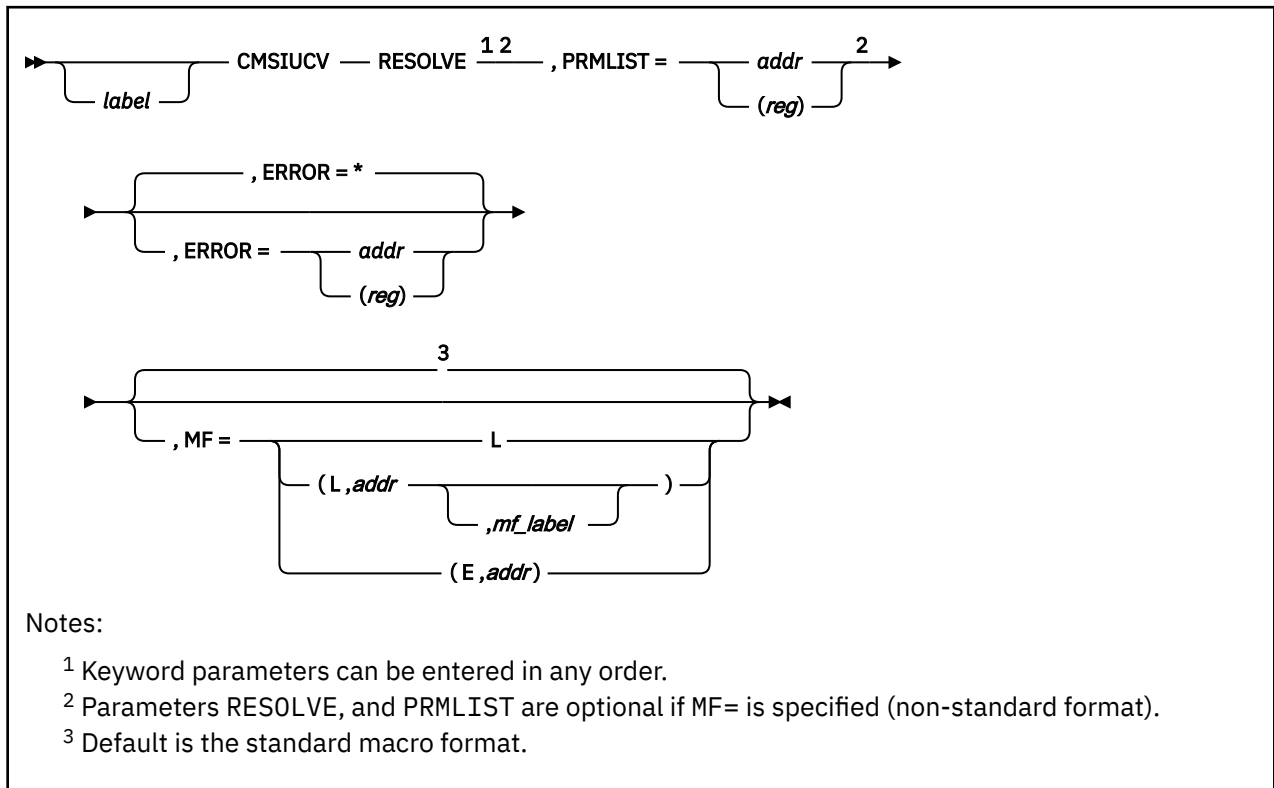
specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

Upon completion of the CMSIUCV QCMSWID function, register 15 contains one of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	The CMS work unit ID has been placed at the address specified by CMSWID.
X'08'	8	No HNDIUCV SET has been issued for this program.
X'0A'	10	The path ID specified is invalid.
X'0C'	12	The program does not own the path.
X'0E'	14	The path ID specified is not associated with a CMS work unit.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'18'	24	The PRMLIST parameter was not specified or its address is equal to 0.
X'28'	40	An invalid CMSIUCV function was specified; it must be ACCEPT, CONNECT, QCMSWID, SEVER, or RESOLVE.

CMSIUCV RESOLVE



Purpose

Use the RESOLVE function to let an application get the results of a CMS communications directory symbolic destination name resolution without connecting to the resource. The result of the RESOLVE function is placed in the PRMLIST and connection parameter list extension so that the application can examine it. Before applications can issue RESOLVE, you or the system administrator should set up a CMS communications directory file and enable communications directory processing.

When CMS resolves the symbolic destination name, the user-level directory (if it exists) is checked first. If the user-level communication directory file does *not* contain the specified symbolic destination name, CMS searches the system-level communications directory file. If the same symbolic destination name is defined in both the user-level and system-level communication directory files, the information in the system-level file is ignored. Lastly, when a symbolic destination name is found in neither the user-level file nor the system-level file, the IBM-level communication directory file, ICOMDIR NAMES, is searched.

Note: The communications directory entries in the IBM-level file are intended for use by IBM and should not be changed.

For more information on how to set up and control your CMS communications directories using the SET COMDIR command, see the [z/VM: CMS Commands and Utilities Reference](#). For more information on the contents of the CMS communications directory files, see [z/VM: Connectivity](#).

Parameters

Required Parameters:

RESOLVE

Gets values from a CMS communications directory file for examination.

PRMLIST=

specifies the address of the block of storage that contains the APPCVM CONNECT parameter list. Your program must prepare this parameter list before it issues the CMSIUCV RESOLVE macro. The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

Your application should issue an APPCVM CONNECT that includes the following operands:

- RESID, along with the symbolic destination name, that will map to the communications directory file values
- BUFFER, along with the address that will contain the connection parameter list extension
- BUFLen=any valid length for an APPCVM CONNECT connection parameter list extension greater than 120 bytes
- FMH5=NO
- MF=L to only format the parameter list.

Note: The CMS communications directory does not fill the connection parameter list extension fields that relate to PIP data or the logical unit of work ID.

addr

specifies the address identified by the APPCVM CONNECT PRMLIST.

(reg)

specifies a register that contains the address identified by the APPCVM CONNECT PRMLIST.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

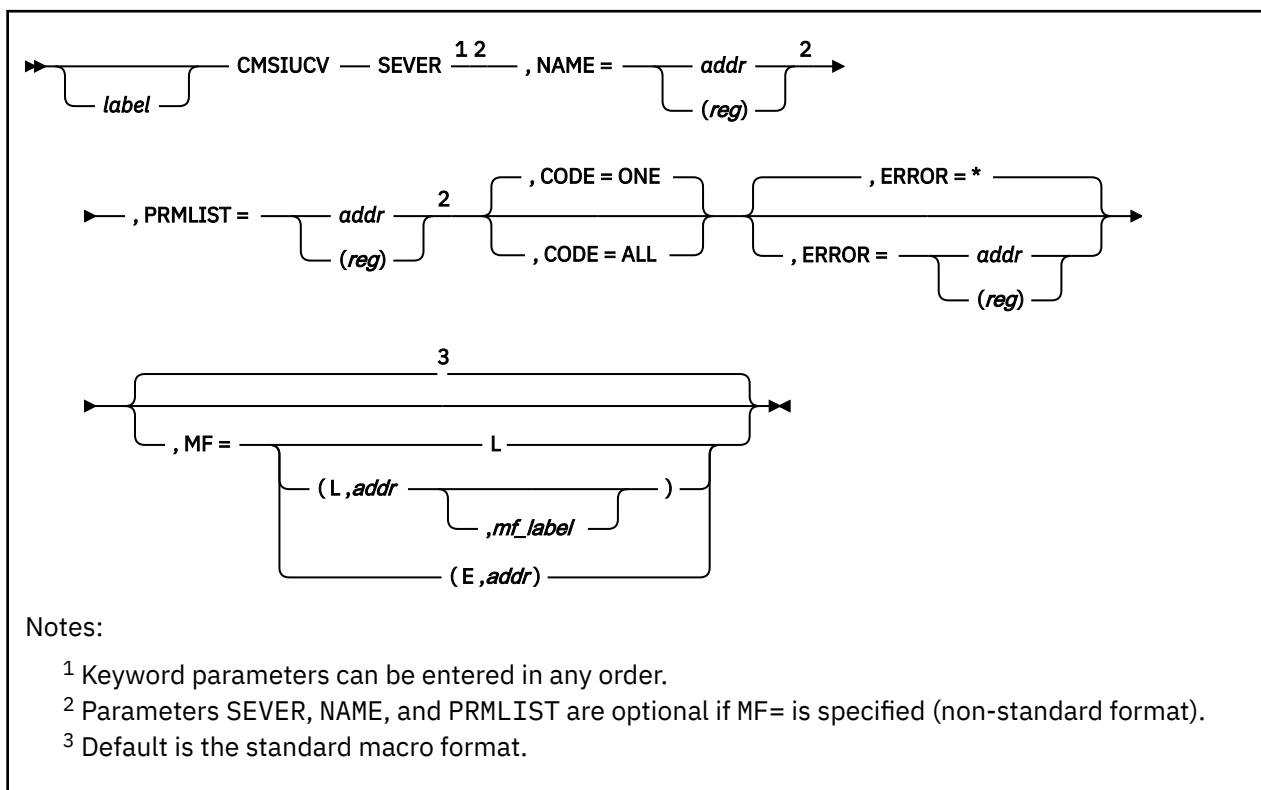
specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

Upon completion of the CMSIUCV RESOLVE function, register 15 contains one of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	CMSIUCV RESOLVE completed successfully.
X'18'	24	The PRMLIST parameter was not specified or its address is equal to 0.
X'28'	40	An invalid CMSIUCV function was specified; must be CONNECT, ACCEPT, SEVER, RESOLVE, or QCMSWID.
X'50'	80	No communications directory entry was found because SET COMDIR OFF was in effect.
X'54'	84	SET COMDIR ON was in effect but no entry was found in the communications directory for the specified symbolic destination name.
X'58'	88	<p>This return code can result for any of the following reasons:</p> <ul style="list-style-type: none"> • The connection parameter list passed to CMSIUCV CONNECT was not an APPC parameter list. • The connection parameter list extension length specified as BUFLN= parameter on APPCVM Connect was less than 120 bytes or was not valid. See the <i>z/VM: CP Programming Services</i> for valid BUFLN= values. • Communications directory resolution was disabled because a connection parameter list extension was provided in FMH5 format.
X'5C'	92	An invalid security tag field was found in the communications directory.
X'7D0' + xxx	2ddd	<p>CMSIUCV was unable to complete the RESOLVE function because an error was encountered in the NAMEFIND routine. The xxx is the hexadecimal return code received from NAMEFIND; the ddd is the decimal equivalent of this return code.</p> <p>Check the communications directory files you are using. If you are not sure of the names of these files, you can find out by issuing the QUERY COMDIR command; the response from that command will indicate what files the system is currently using to perform COMDIR resolution.</p> <p>You should then ensure that those files contain correct information. For more information about NAMEFIND return codes, see the z/VM: CMS Commands and Utilities Reference.</p> <p>After you are satisfied that the content of the files is correct (and the disk(s) they are on have been reaccessed if necessary), you should issue the command SET COMDIR RELOAD. If the problem persists, consult your system administrator. For more information on communications directory files, see z/VM: Connectivity.</p>

CMSIUCV SEVER



Purpose

Use the SEVER function to request that CMS perform a SEVER.

Before issuing this function, a program must set up an APPCVM SEVER parameter list and pass it to CMS. CMS severs any exit routines established for the path.

Parameters

Required Parameters:

SEVER

Ends communications with another program, and lets CMS know about it.

NAME=

specifies the symbolic name that identifies the program associated with this path. A program with this name must have previously issued an HNDIUCV macro to identify itself as an APPC/VM program to CMS.

addr

specifies the address of an 8 character symbolic name.

(reg)

specifies a register that contains the address of the 8-character symbolic name.

If the program requests a SEVER for a specific path and the NAME specified does not correspond with the owner of that path, the SEVER is not permitted.

PRMLIST=

specifies the storage address that contains the IUCV SEVER or APPCVM SEVER parameter list. Your program must prepare this parameter list before it issues the CMSIUCV SEVER. To prepare the parameter list, your program must use the list form (MF=L) of the IUCV SEVER or APPCVM SEVER

macro. (This lets you set up the APPC/VM parameter list using macro keyword parameters instead of storing information with IPARML DSECT labels.) The address must be a guest real address, that is, the address must be within the virtual machine's real address space (guest=real). Also, the parameter list must be on a doubleword boundary.

addr

specifies the address of the program's APPC/VM parameter list.

(reg)

specifies a register that contains the address of the program's APPC/VM parameter list.

Optional Parameters:

label

is an optional assembler label for the statement.

CODE=

specifies whether one or all paths owned by the program are severed.

ALL

severs all APPC/VM paths owned by the program.

If the program requests a SEVER function with CODE=ALL, all APPC/VM paths owned by that program are severed. The IPUSER field of the APPCVM SEVER parameter list is set to binary 1's. The CP parameter list passed as input on the CMSIUCV SEVER, CODE=ALL should be an IUCV parameter list.

ONE

severs only one APPC/VM path, which was specified with the PATHID parameter on APPCVM SEVER or IUCV SEVER. This is the default value unless MF=(E,addr) is specified.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If you issue a CMSIUCV SEVER on a protected (SYNCLVL=SYNCPT) conversation, you should roll back the work unit associated with that conversation before doing any other processing on that work unit. For more information on when the work unit may need to be rolled back, see *Synchronizing Updates to Multiple Resources* section in *z/VM: CMS Application Development Guide for Assembler*. For information

about protected conversations and the Coordinated Resource Recovery (CRR) facility in CMS, see [z/VM: CMS Application Development Guide](#).

Return Codes

Upon completion of the CMSIUCV SEVER function, register 15 contains either:

- A 5-digit reason code returned by a CSL routine that was called by CMSIUCV SEVER processing. These are described in [z/VM: CMS and REXX/VM Messages and Codes](#), or
- One of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	For an IUCV SEVER, your function completed normally. For an APPCVM SEVER, the SEVER started successfully, but has not completed. For more information on IUCV SEVER and APPCVM SEVER, see the z/VM: CP Programming Services (This corresponds to a CC=0 for either APPCVM SEVER or IUCV SEVER.)
X'02'	2	An APPC/VM parameter list was passed as input to CMSIUCV SEVER, and the APPCVM SEVER completed immediately. The function complete information is in the parameter list. The user's path-specific exit is not called because CP does not reflect an interrupt to the virtual machine. (This reflects a CC=2 for the APPCVM SEVER.)
X'03'	3	An APPCVM SEVER function was requested, and completed immediately. CP stored error information in the IPAUDIT field of the CP APPC/VM parameter list. The user's path-specific exit is not called because CP does not reflect an interrupt to the virtual machine. (This reflects a CC=3 for the APPCVM SEVER.)
X'08'	8	No HNDIUCV SET has been issued for this program.
X'0C'	12	The program does not own the path.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'18'	24	The PRMLIST parameter was not specified or its address is equal to 0.
X'1C'	28	An IUCV SEVER with ALL=YES is not allowed.
X'27'	39	An IUCV SEVER with KEEP=YES is not allowed as input on a CMSIUCV SEVER with CODE=ALL.
X'28'	40	An invalid CMSIUCV function was specified; must be CONNECT, ACCEPT, SEVER, RESOLVE, or QCMSWID.
X'34'	52	The APPCVM CONNECT parameter list is invalid—only the reserved username, !CMS, can specify CONTROL=YES. (!CMS is a reserved name for CMS. CMS uses !CMS as a user ID so it can use its own APPC/VM support.)
X'3A'	58	An allocation error sever (sever code = X'1xx') to a SYNCLVL=SYNCPT path was not allowed because there are other resources registered in the same work unit.
X'44'	68	An APPC/VM parameter list is not allowed as input on a CMSIUCV SEVER, CODE=ALL.

Hex Code	Decimal Code	Meaning
X'C8' + xx	2dd	An error was encountered in getting CMS free storage. The xx is the hexadecimal return code from CMSSTOR. The dd is the decimal equivalent of this return code.
X'3E8' + xxx	1ddd	Indicates that an APPC/VM or IUCV error occurred. The xxx is the IPRCODE field returned by the APPCVM SEVER or IUCV SEVER macro to aid in diagnosing the error. The ddd is the decimal equivalent of this IPRCODE value. For more information on the APPCVM SEVER or IUCV SEVER return codes, see z/VM: CP Programming Services .

CMSLEVEL

➤ CMSLEVEL ➤

Purpose

Use the CMSLEVEL macroinstruction with the CMS command QUERY CMSLEVEL to map the release level of the CMS you are running on.

Usage Notes

- 1. If you just want to obtain the register contents, you may want to suppress the response associated with the CMS command QUERY CMSLEVEL. You can suppress the typing of the response by:
 - Issuing the SET CMSTYPE HT command before issuing QUERY CMSLEVEL and issuing SET CMSTYPE RT afterward
 - Using the Extract/Replace CSL routine (described in the [z/VM: CMS Callable Services Reference](#)) to set HT using the NO_TYPE_HT information name.
- 2. After issuing QUERY CMSLEVEL from your assembler language program, register 0 contains the fullword at USERLVL in NUCON. This field is reserved for the user. Register 1 contains:

bit	description
0-7	Reserved
8-15	Release number
16-31	Service level

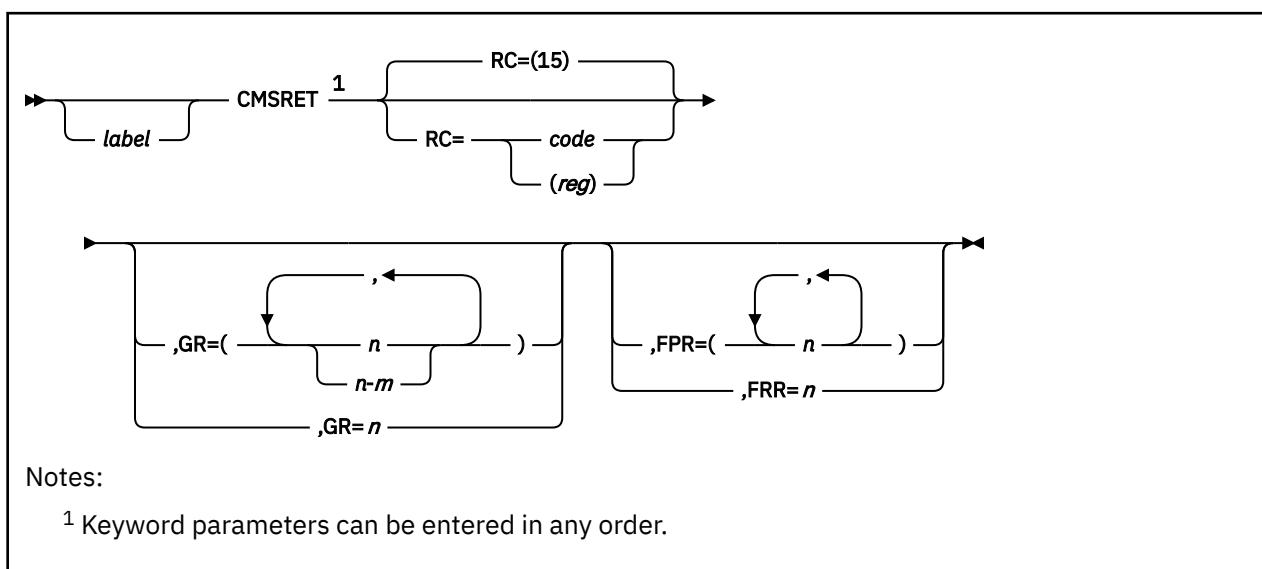
The service level is a halfword field in binary format.

- 3. The CMSLEVEL mapping macro expands as follows:

```
MACRO
CMSLEVEL
*
* THE CODE FOR RELEASE IS DEFINED AS:
*
VMR6 EQU X'00' - VM/370 RELEASE 6
VMBSEP EQU X'01' - VM/BSEP RELEASE 2
VMSEP EQU X'02' - VM/SEP RELEASE 2
VMSP1 EQU X'03' - VM/SP RELEASE 1
VMSP2 EQU X'04' - VM/SP RELEASE 2
VMSP3 EQU X'05' - VM/SP RELEASE 3
VMSP4 EQU X'06' - VM/SP RELEASE 4
VMSP5 EQU X'07' - VM/SP RELEASE 5
VMSP55 EQU X'08' - VM/SP RELEASE 5.5
VMSP56 EQU X'08' - VM/SP RELEASE 5.6
VMSP6 EQU X'09' - VM/SP RELEASE 6
CMS7 EQU X'0A' - VM/ESA RELEASE 1 and 1.5; CMS Level 7
CMS8 EQU X'0B' - VM/ESA Release 1.1; CMS Level 8
CMS9 EQU X'0C' - VM/ESA Release 2; CMS Level 9
CMS10 EQU X'0D' - VM/ESA Release 2.1; CMS Level 10
CMS11 EQU X'0E' - VM/ESA Release 2.2; CMS Level 11
CMS12 EQU X'0F' - VM/ESA CMS Level 12 and later
*
VMPC EQU X'10' - VM/PC VERSION 1.00
VMPC2 EQU X'20' - VM/PC VERSION 2.00
* CMSLEVEL is frozen at X'0F' for CMS 12 and above.
* Use DMSQEFL macro or DMSQEFL CSL routine instead.
```


4. You can also use the DMSQEFL CSL routine to return information about the level of CMS to a program.
5. A value of X'0F' indicates CMS 12 or higher. To determine CMS level on VM/ESA® Version 2 Release 1 or later, use the DMSQEFL CSL routine or DMSQEFL macroinstruction.

CMSRET



Purpose

Use the CMSRET macro to return to the caller from a program which was invoked by SVC 202 or CMSCALL.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

RC=

specifies the return code to be placed in register 15. If you do not specify RC=, CMS returns register 15 unchanged to the caller. Acceptable values are:

code

specifies the actual return code.

(reg)

specifies the register containing the return code.

GR=

lists the general registers CMS passes unchanged to the caller. Specify registers as decimal numbers with no leading zeros and separated by commas. You can list them in any order. Specify a range as *n-m*, where *n* and *m* are decimal numbers. If you specify only one register, the parentheses are not required. CMS restores all other general registers to their values at entry, except register 15, which CMS always uses to pass the return code.

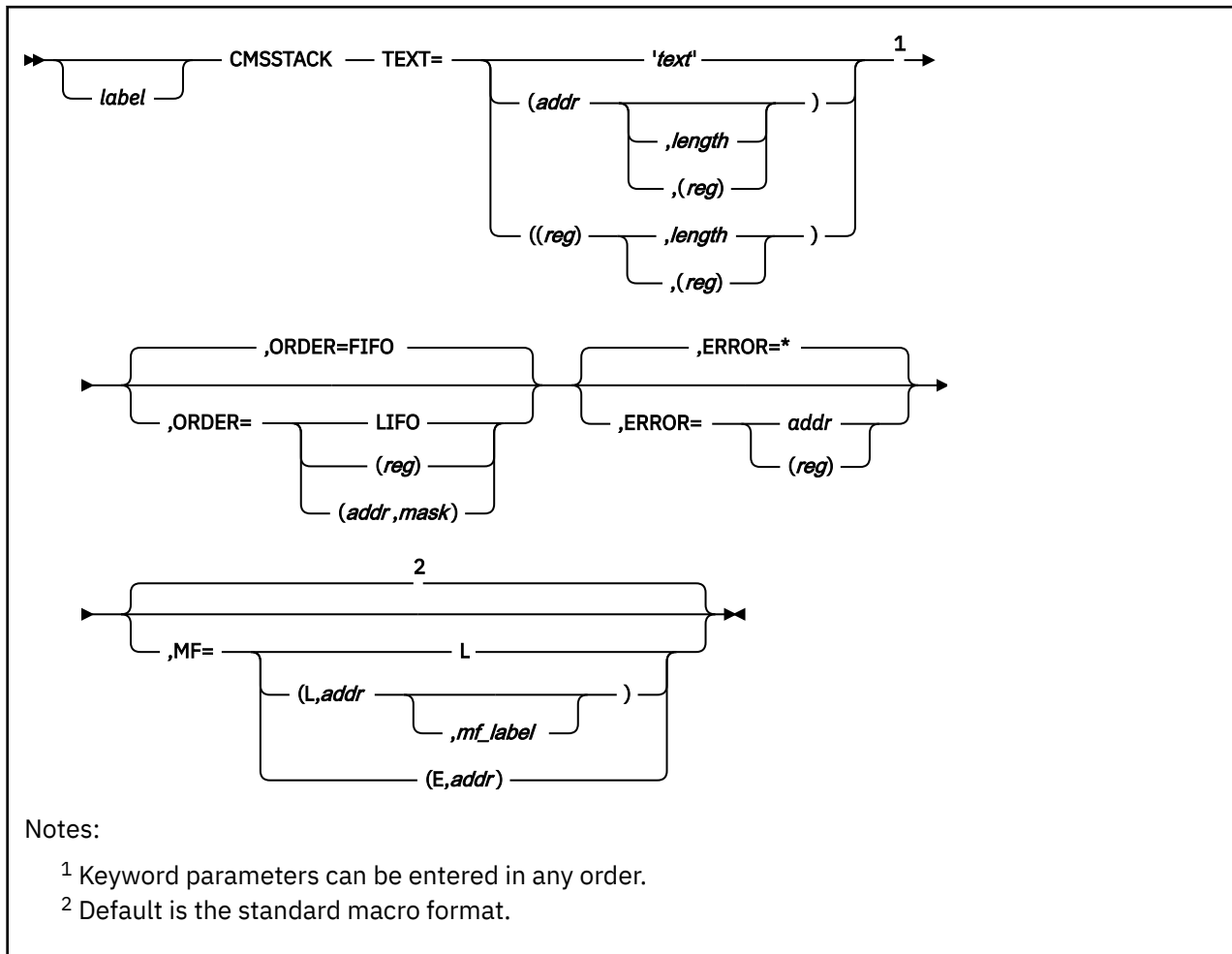
FPR=

lists the floating point registers CMS passes unchanged to the caller. Specify the registers as decimal numbers with no leading zeros and separated by commas. You can list them in any order. If you specify only one register, the parentheses are not required. Floating point registers other than 0, 2, 4, and 6 cannot be used with this parameter.

Usage Notes

1. Immediate commands must use BR 14 rather than CMSRET to return control. Using CMSRET may cause the program that invoked the immediate command to end, rather than causing just the immediate command itself to end.

CMSSTACK



Purpose

Use the CMSSTACK macro to place data on the program stack.

Parameters

Required Parameters:

TEXT=

specifies the data to be stacked. Acceptable values are:

'text'

explicitly defines the data to be stacked. If 'text' contains mixed DBCS data, CMS will not validate the data if it is truncated.

(addr, length)

specifies the address of the data as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address of the data as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((*reg*),*length*)

specifies a register that contains the address of the data and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((*reg*),(*reg*))

specifies a register that contains the address of the data and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ORDER=

specifies the order CMS uses to operate on records in the stack. Acceptable values are:

FIFO

instructs CMS to treat the stack as a queue (first in first out). FIFO is the default value.

LIFO

instructs CMS to treat it as a push down stack (last in first out).

(*reg*)

instructs the macro to check the value of the specified register and, if it is 0, sets ORDER to FIFO. If the register contains a nonzero value, the macro sets ORDER to LIFO.

(*addr*,*mask*)

defines a single bit in storage that sets the value of the ORDER parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then ORDER is set to FIFO. If the bit is 1, then ORDER is set to LIFO. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the ORDER parameter as

```
ORDER=(APPFLAG,X'80')
```

To set the value of the ORDER parameter at assembly time, specify ORDER=FIFO or ORDER=LIFO.

To set the value at execution time, specify ORDER=(*reg*) or ORDER=(*addr*,*mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E, *addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
------	---------

CMSSTOR

Purpose

Use the CMSSTOR macro to allocate and release free storage. CMSSTOR has two functions:

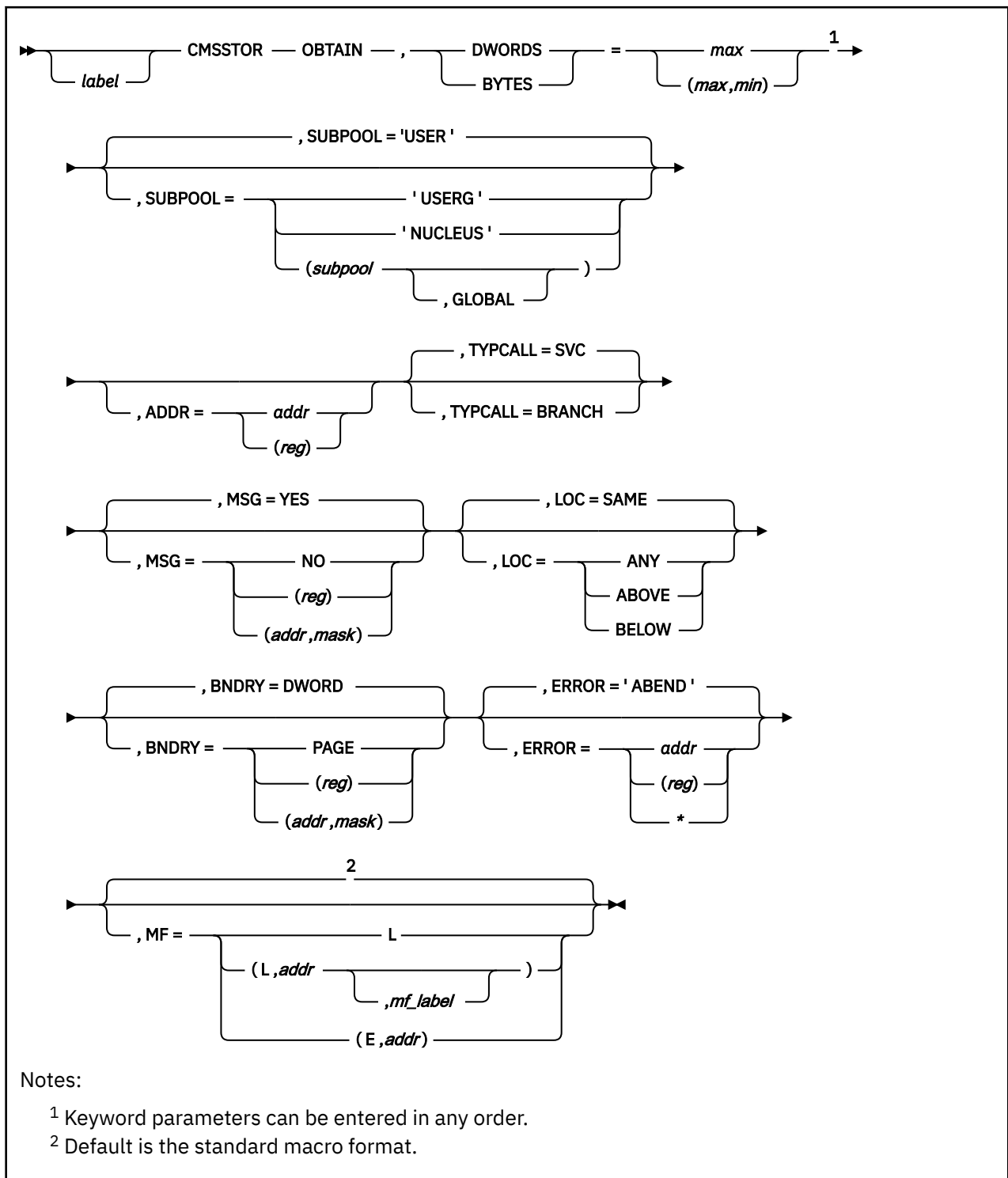
CMSSTOR OBTAIN

allocates free storage

CMSSTOR RELEASE

releases free storage.

CMSSTOR OBTAIN



Purpose

Use the CMSSTOR OBTAIN macro to allocate free storage.

Parameters

Required Parameters:

OBTAIN

allocates CMS free storage.

DWORDS=

requests free storage in doublewords. DWORDS and BYTES are mutually exclusive parameters; they cannot be specified on the same macro call. Acceptable values are:

max

is the number of doublewords of free storage you request. Specifying *max* without *min*, indicates a fixed request for free storage.

Note: If you use the standard macro form, you must specify *max*.

min

indicates a variable free storage request. If *max* number of doublewords is not available, CMS obtains the largest block of storage greater than or equal to the minimum, *min*.

Both *max* and *min* must be greater than 0. Specify *max* and *min* as any valid assembler expression or as a register that contains the number. Valid registers are 2-12; register 0 may be specified for *max* only. In addition, register 1 may be specified for *min* only when using the standard or execute macro form. Specifying register 1 for *min* when using the standard format results in the generation of nonreentrant code.

The possible combinations of the DWORDS parameters for requesting free storage are as follows:

Fixed request (only *max* specified)

```
DWORDS=n
DWORDS=(reg)
```

Variable request (*max* and *min* specified)

```
DWORDS=(n,n)
DWORDS=(n,(reg))
DWORDS=(reg,n)
DWORDS=(reg,(reg))
```

BYTES=

requests free storage in bytes. DWORDS and BYTES are mutually exclusive parameters; they cannot be specified on the same macro call. For BYTES, CMS rounds the amount up to the next doubleword multiple if it is not already a doubleword value.

max

is the number of bytes of free storage you request. Specifying *max* without *min*, indicates a fixed request for free storage.

Note: If you use the standard macro form, you must specify *max*.

min

indicates a variable free storage request. If *max* number of bytes is not available, CMS obtains the largest block of storage greater than or equal to the minimum, *min*.

Both *max* and *min* must be greater than 0. Specify *max* and *min* as any valid assembler expression or as a register that contains the number. Valid registers are 2-12; register 0 may be specified for *max* only. In addition, register 1 may be specified for *min* only when using the standard or execute macro form. Note that specifying register 1 for *min* when using the standard format results in the generation of nonreentrant code.

The possible combinations of the BYTES parameters for requesting free storage are as follows:

Fixed request (only *max* specified)

```
BYTES=n
BYTES=(reg)
```

Variable request (*max* and *min* specified)

```
BYTES=(n,n)
BYTES=(n,(reg))
BYTES=((reg),n)
BYTES=((reg),(reg))
```

Optional Parameters:

label

is an optional assembler label for the statement.

SUBPOOL=

indicates the subpool from where CMS obtains the free storage. For more information on specifying subpool names, see the usage notes. Acceptable values are:

'USER'

obtains storage from the USER subpool, which has storage protect key X'E'. This is the default.

'USERG'

when z/CMS is running in the virtual machine, obtains storage above 2 GB from the USERG subpool, which has storage protect key X'E'. See usage note [“7”](#) on page 98.

'NUCLEUS'

obtains storage from the NUCLEUS subpool, which has storage protect key X'F'.

subpool

indicates the user-identified subpool name.

'name'

obtains storage from a named user storage subpool. '*name*' must be from 1 to 8 characters in length. If '*name*' is less than 8 characters, it is padded on the right with blanks (as are 'USER', 'USERG', and 'NUCLEUS').

addr

obtains storage from the subpool named at the specified address. This may be any assembler expression.

(*reg*)

obtains storage from the subpool named at the address contained in the specified register. The valid registers are 2-12 enclosed in parentheses.

('name', GLOBAL)

obtains storage from the GLOBAL subpool with the specified name. '*name*' must be from 1 to 8 characters in length. If '*name*' is less than 8 characters, it is padded on the right with blanks.

(*addr*, GLOBAL)

obtains storage from the GLOBAL subpool named at the specified address. The variable *addr* may be any assembler expression.

((*reg*), GLOBAL)

obtains storage from the GLOBAL subpool named at the address contained in the specified register. The valid registers are 2-12 enclosed in parentheses.

ADDR=

specifies that CMSSTOR must allocate storage from the specified address. If CMSSTOR cannot allocate the requested amount of storage from the specified address, it generates an out-of-storage condition. ADDR must be aligned on a doubleword boundary. Acceptable values are:

addr

allocates storage from the address specified by *addr*. This can be any assembler expression other than a label.

(reg)

allocates storage from the address contained in the specified register. The valid registers are 2-12 enclosed in parentheses.

Note: Specifying BNDRY and ADDR or LOC and ADDR on the same macro call causes an error. If specified on separate macro calls to build a single parameter list, ADDR is honored and BNDRY or LOC ignored.

TYPCALL=

indicates how control is passed to CMSSTOR. Because CMSSTOR is a nucleus resident routine, use TYPCALL=BRANCH if the calling routine is nucleus resident. Use TYPCALL=SVC if the calling routine is not nucleus resident. Acceptable values are:

SVC

indicates the calling routine is not nucleus resident. This is the default value.

BRANCH

indicates the calling routine is nucleus resident.

Note: The calling routine must make sure it calls CMSSTOR with the proper storage key and with interrupts disabled.

MSG=

indicates whether CMS displays an error message if it cannot allocate sufficient storage to satisfy the request. Acceptable values are:

YES

indicates CMS displays an error message. This is the default value.

NO

indicates CMS does not display an error message.

(reg)

CMSSTOR checks the value of the specified register and, if it is 0, sets MSG to NO. If the register contains a nonzero value, the macro sets MSG to YES.

(addr,mask)

defines a single bit in storage that sets the value of the MSG parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then MSG is set to NO. If the bit is 1, then MSG is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the MSG parameter as

```
MSG=(APPFLAG,X'80')
```

To set the value of the MSG parameter at assembly time, specify MSG=YES or MSG=NO. To set the value at execution time, specify MSG=(reg) or MSG=(addr,mask).

LOC=

indicates from where CMS can allocate storage. Acceptable values are:

SAME

allocates storage based on the current addressing mode. If the caller is running in 31-bit addressing mode, LOC=SAME allocates storage from above the 16 MB line if available. If storage is not available above 16 MB, CMSSTOR allocates it from below 16 MB. If the caller is running in 24-bit addressing mode, LOC=SAME allocates storage from below the 16 MB line. LOC=SAME is the default.

ANY

allocates storage from above or below the 16 MB line. If possible, CMS obtains storage from above the 16 MB line; if none is available, CMS allocates it from below the 16 MB line. Note that a 24-bit mode program **can** allocate storage from above 16 MB.

ABOVE

allocates storage from above the 16 MB line.

BELOW

allocates storage from below the 16 MB line.

Note: Specifying LOC and ADDR on the same call causes an error. If specified on separate macro calls to build a single parameter list, ADDR is honored and LOC ignored.

BNDRY=

indicates the type of boundary alignment required for the storage. Acceptable values are:

DWORD

aligns storage on a doubleword boundary. This is the default value.

Note: Specifying BNDRY=DWORD does not ensure that successive calls to CMSSTOR OBTAIN will obtain adjacent areas of storage even if the adjacent area is free.

PAGE

aligns storage on a 4 KB page boundary.

(reg)

CMSCALL checks the value of the specified register and, if it is 0, sets BNDRY to DWORD. If the register contains a nonzero value, the macro sets BNDRY to PAGE.

(addr,mask)

defines a single bit in storage that sets the value of the BNDRY parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then BNDRY is set to DWORD. If the bit is 1, then BNDRY is set to PAGE. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the BNDRY parameter as

```
BNDRY=(APPFLAG,X'80')
```

To set the value of the BNDRY parameter at assembly time, specify BNDRY=PAGE or BNDRY=DWORD. To set the value at execution time, specify BNDRY=(reg) or BNDRY=(addr,mask).

Note: Specifying BNDRY and ADDR on the same call causes an error. If specified on separate macro calls to build a single parameter list, ADDR is honored and BNDRY ignored.

ERROR=

specifies an action to be taken if an error occurs. Acceptable values are:

'ABEND'

abends the program. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register. Registers 2-12 are valid.

passes control to the next sequential instruction.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E, *addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If CMSSTOR OBTAIN is successful, it stores:
 - a. A 0 return code in register 15
 - b. The number of BYTES/DWORDS allocated in register 0 (the number of BYTES/DWORDS is a doubleword multiple).
 - c. The address of the storage allocated in register 1. This address replaces the address of the parameter list passed to the CMSSTOR macro upon invocation. If an error occurs during CMSSTOR processing and control returns to the caller, register 1 is not changed; it still points to the parameter list.
2. CMSSTOR OBTAIN places *max* (the number of BYTES or DWORDS of storage requested) into register 0 before the invocation of the storage management system.
3. The rules for subpool naming are:
 - a. Subpool names can use any characters. (Note that this implies that subpool names are case sensitive and SUBPOOL='XYZ' is not the equivalent of SUBPOOL='xyz'.)
 - b. The subpool names DMSxxxxx are reserved for system use, and the names USER, USERG, and NUCLEUS are for reserved system subpools; otherwise, there are no restrictions on the names you give subpools.
 - c. You cannot use CMSSTOR OBTAIN to create SHARED subpools; you can use it to create PRIVATE and GLOBAL subpools.
 - d. If you specify a named subpool that has not been previously created, CMS creates a new subpool. If you specify GLOBAL, CMS creates a global subpool. Otherwise CMS creates a private subpool.
 - e. To create named subpools for nucleus key storage, you must use the SUBPOOL macro; you cannot use CMSSTOR OBTAIN. To obtain storage in nucleus key, you can specify the NUCLEUS subpool; however, this subpool is shared with all applications running in your virtual machine.

For more information on subpools, see [“SUBPOOL” on page 401](#).

4. CMSSTOR always treats the address on the ADDR= parameter as a 31-bit address. Therefore, if a program specifies a 24-bit address for ADDR=, the program must make sure that bits 1-7 are 0s. Bit 0 is always ignored.
5. Specifying the LOC parameter returns free storage as follows:

Specification	24-bit Addressing	31-bit Addressing	
		AMODE	
		24	31
LOC=SAME	BELOW	BELOW	ABV/BEL (see LOC=ANY)
LOC=BELOW	BELOW	BELOW	BELOW
LOC=ABOVE	ERROR	ABOVE	ABOVE
LOC=ANY	BELOW	ABOVE/BELOW - if sufficient storage exists ABOVE the 16 MB line, otherwise BELOW	

6. Reentrant code can be produced with the standard macro form or a combination of MF=L and MF=(E, *addr*) forms of macro expansions for CMSSTOR OBTAIN. To build a parameter list without involving parameter substitutions, the following restrictions must be met:

On the standard macro form the:

- DWORDS/BYTES *max* parameter can be any valid value and is always loaded into register 0.

- DWORDS/BYTES *min* parameter, if specified, must be a register form or constant. Do not use register 1 for this when using the standard macro form.
- ADDR= parameter, if specified, must be a register form or constant. Do not use register 1 for this when using the standard macro form.
- SUBPOOL= parameter, must be a quoted constant, for example, SUBPOOL='XYZ' or SUBPOOL=('XYZ',GLOBAL).
- BNDRY= and MSG=, if specified, cannot be the (*addr,mask*) form.

For example:

```
CMSSTOR OBTAIN,BYTES=( (R3) , 50) ,ADDR=(R2) ,MSG=NO ,ERROR=*
```

On the MF=(E,*addr*) form:

The only parameter you can specify in variable form is the *max* portion of the BYTES/DWORDS parameter, which is loaded into register 0 before invoking the storage management system. If you specify this parameter, you must specify *max* in register notation with the MF=(E,*addr*) macro format. If *max* were specified as a number, it would be stored into the parameter list before being loaded into register 0. As a result, the previously-set default for *max* would be replaced by the specified number.

This means that:

- Only *max*, TYPICAL=, and ERROR= can be specified on the MF=(E,*addr*) form.
- Defaults are not recognized on the MF=(E,*addr*) form, and to specify a default would cause storing into the parameter list. For example, specifying SUBPOOL='USER' on the MF=(E,*addr*) form is nonreentrant. To allow the combination of MF=L and MF=(E,*addr*) macro formats to create inline reentrant parameter lists, the MF=(E,*addr*) format does not store default values into the parameter list. For example, while other CMS macros with an MF=(E,*addr*) format store the name of the function being called, CMSSTOR does not store the function name into the parameter list. The MF=(L,*addr*) macro format can be used to store the function name and other values into an uninitialized parameter list.

The action taken by the macro for the *max* portion of the BYTES/DWORDS parameter depends upon the macro form and the specification of the parameter:

	Standard	MF=L	MF=(L, <i>addr</i>)	MF=(E, <i>addr</i>)
Register	Loaded	Error	Stored	Loaded
Number	Inline Loaded	Inline	Stored	Stored Loaded
Unspecified	Error	Skipped	Skipped	Loaded

The action taken by the macro for the *min* portion of the BYTES/DWORDS parameter and the ADDR= parameter depends upon the macro form and the specification of the parameter:

	Standard	MF=L	MF=(L, <i>addr</i>)	MF=(E, <i>addr</i>)
Register	Inline “6.a” on page 97	Error	Stored	Stored
Number	Inline	Inline	Stored	Stored
Unspecified	Skipped	Skipped	Skipped	Skipped
Note: a. If register 1 is used, the action taken by the macro will be stored.				

Where:

Inline

The value or register equate number is generated as an inline constant within the parameter list. The register is not manipulated.

Stored

The value is stored into the parameter list. If the parameter is *max* it may be loaded into R0 on the same macro invocation, see below.

Loaded

Only applies to the *max* portion of the BYTES/DWORDS parameter and indicates the value is placed into R0 from the source. If the source is a register, the parameter list is not stored into or read from, even if a value was placed there by a previous macro invocation.

If the source is not a register, the value is read from the parameter list. The value may have been placed into the parameter list on this macro call, or a previous one.

Note: This mechanism provides the ability to store a default value into a parameter list on an MF=L call, and override the value if required by invoking an MF=(E,*addr*) call with a register specification. The register value is used and the value in the parameter list ignored. If a call is made with an MF=(E,*addr*) form without any specification for BYTES/DWORDS, the default value would be loaded from the parameter list into register 0.

Skipped

Indicates that no action is taken whatsoever.

Error

Indicates that an MNOTE is generated.

7. z/CMS does not directly exploit storage above 2 GB. However, z/CMS can be IPLed in a virtual machine with more than 2 GB of storage, and programs can use SUBPOOL='USERG' to allocate storage above 2 GB. The number of pages of storage to be allocated is specified using the BYTES= parameter of CMSSTOR OBTAIN. For example, the following statement will allocate 1 page of storage above 2 GB:

```
CMSSTOR OBTAIN,BYTES=1,SUBPOOL='USERG'
```

The 64-bit address of the allocated storage is returned in general-purpose register 1. All storage allocated above 2 GB is aligned on a page boundary and the BNDRY= parameter is ignored. In addition, the ADDR=, TYPCALL=, and LOC= parameters should not be specified. TYPCALL=SVC is the only TYPCALL value supported.

SUBPOOL='USERG' is supported only under z/CMS. Before assembling a program that uses SUBPOOL='USERG', you must issue the GLOBAL MACLIB command to specify the DMSZGPI macro library ahead of the DMSGPI library. For example:

```
global maclib dmszgpi dmsgpi
```

8. Although CMSSTOR and OS/MVS macro calls for storage management are implemented via the same CMS storage management subsystem, mixing these calls is not recommended. The default subpool name in which CMSSTOR obtains storage is USER, while the default subpool name used by the OS/MVS macro calls (GETMAIN/FREEMAIN) is DMSOS000. Also, GETMAIN automatic storage cleanup might be affected by the CMS STORECLR setting.

Return Codes

When CMSSTOR OBTAIN completes, register 15 contains one of the following return codes:

Code**Meaning****0**

Normal completion. Register 0 contains the amount of storage allocated and register 1 contains the address of the storage.

1

Insufficient storage space is available to satisfy the request for free storage. In the case of the variable request, even the minimum request could not be satisfied.

If the ADDR parameter was specified, this error indicates that insufficient storage was available to satisfy the request at the specified address; there may still be sufficient amounts of free storage at other locations.

If the BNDRY=PAGE parameter was specified, sufficient storage may exist to satisfy the request, however, all pages of free storage have been at least partially used and storage to satisfy the request cannot be found on a page boundary.

2

USER key storage pointers destroyed.

3

NUCLEUS key storage pointers destroyed.

4

An invalid size was requested. This error is taken if the requested size is not greater than 0. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. (However, the latter error is not detected if CMSSTOR is able to satisfy the maximum request.)

7

The address given for an OBTAIN is not doubleword aligned or the specified address plus the amount of storage requested would cross either the 16 MB boundary or the storage size of the virtual machine.

9

Unexpected and unexplained error in the storage management routine.

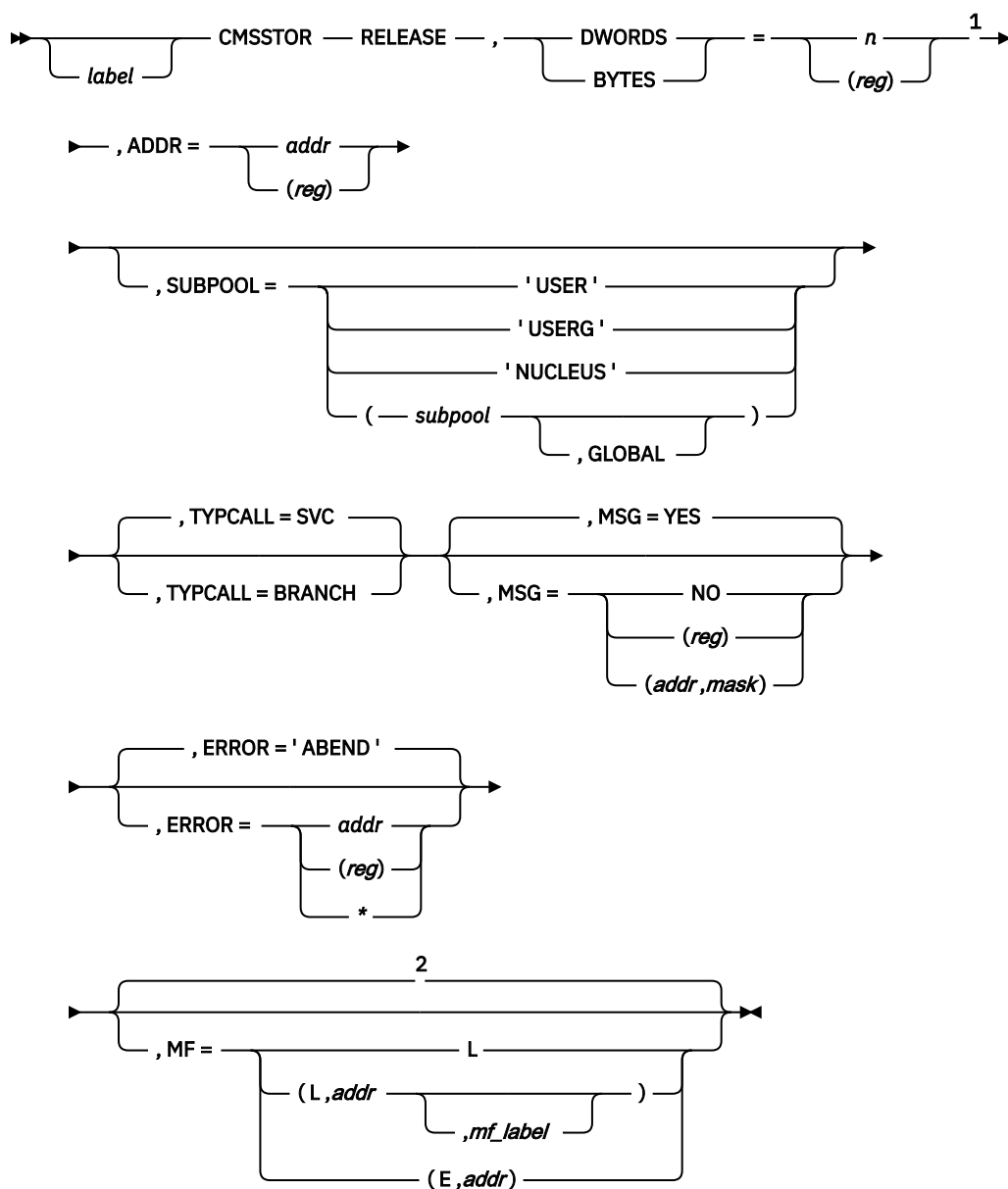
11

A register specified for either the *min* portion of BYTES/DWORDS or the ADDR= parameter is not in the range 2-12.

12

The subpool name USERG is allowed only when running in z/CMS.

CMSSTOR RELEASE



Notes:

¹ Keyword parameters can be entered in any order.

² Default is the standard macro format.

Purpose

Use CMSSTOR RELEASE to release free storage.

Parameters

Required Parameters:

RELEASE

releases free storage previously allocated by CMSSTOR OBTAIN.

DWORDS=

is the number of doublewords of free storage to be released. DWORDS and BYTES are mutually exclusive parameters; you cannot specify both on the same macro call. Acceptable values are:

n

specifies the number of doublewords to be released.

(reg)

releases the number of doublewords specified in the register. Valid registers are 0 and 2-12 enclosed in parentheses.

Note: If you use the standard macro form, you must specify either BYTES or DWORDS.

BYTES=

is the number of bytes of free storage to be released. DWORDS and BYTES are mutually exclusive parameters; you cannot specify both on the same macro call. When you specify BYTES, the value is rounded up to the next doubleword multiple if not already a doubleword value. Acceptable values are:

n

specifies the number of bytes to be released.

(reg)

releases the number of bytes specified in the register. Valid registers are 0 and 2-12 enclosed in parentheses.

Note: If you use the standard macro form, you must specify either BYTES or DWORDS.

ADDR=

specifies the address of the storage to be released. Acceptable values are:

addr

releases storage at the specified location.

(reg)

releases storage from the address in the specified register. Registers 2-12 are valid; register 1 can also be specified when using the standard or execute macro form. Note that specifying register 1 when using the standard format results in the generation of nonreentrant code.

Note: If you use the standard macro form, you must specify ADDR.

Optional Parameters:

label

is an optional assembler label for the statement.

SUBPOOL=

the subpool from which the storage was allocated. Using SUBPOOL= allows CMSSTOR to verify that the subpool specified is the one from which the storage was obtained. If the storage specified by ADDR= is not within the subpool specified, no release occurs; an error code is returned. If you omit SUBPOOL=, this verification is not done. The purpose of using SUBPOOL is to reduce the possibility of releasing the wrong storage.

Acceptable values are:

'USER'

returns storage obtained from the USER subpool, which has a storage protect key X'E'.

'USERG'

when z/CMS is running in the virtual machine, returns storage above 2 GB obtained from the USERG subpool, which has a storage protect key X'E'. See usage note [“6” on page 104](#).

'NUCLEUS'

returns storage obtained from the NUCLEUS subpool, which has a storage protect key X'F'.

subpool

indicates the user-identified subpool name. Acceptable values are:

'name'

returns storage obtained from a named user storage subpool. If *'name'* is less than 8 characters in length, it is padded on the right with blanks (as are 'USER', 'USERG', and 'NUCLEUS').

addr

returns storage obtained from the subpool named at the specified address. *addr* may be any assembler expression.

(reg)

returns storage obtained from the subpool named at the address contained in the specified register. The valid registers are 2-12 enclosed in parentheses.

('name', GLOBAL)

returns storage obtained from the GLOBAL subpool with the specified name. If *'name'* is less than 8 characters in length, it is padded on the right with blanks.

(addr, GLOBAL)

returns storage obtained from the GLOBAL subpool named at the specified address. *addr* may be any assembler expression.

((reg), GLOBAL)

returns storage obtained from the GLOBAL subpool named at the address contained in the specified register. The valid registers are 2-12 enclosed in parentheses.

Note: Private subpools are not hidden across SVC levels for calls to CMSSTOR RELEASE as they are for CMSSTOR OBTAIN. You cannot use CMSSTOR OBTAIN to allocate storage from a private subpool that was created at a different SVC level. If storage is allocated to a private subpool at a different SVC level and you specify the name of that subpool on the CMSSTOR RELEASE SUBPOOL parameter, CMSSTOR releases the storage, even though it is at a different SVC level.

TYPICAL=

indicates how control is passed to CMSSTOR. Because CMSSTOR is a nucleus resident routine, use TYPICAL=BRANCH if the calling routine is nucleus resident. Use TYPICAL=SVC if the calling routine is not nucleus resident. Acceptable values are:

SVC

indicates the calling routine is not nucleus resident. This is the default value.

BRANCH

indicates the calling routine is nucleus resident.

Note: The calling routine must make sure it calls CMSSTOR with the proper storage key and with interrupts disabled.

MSG=

indicates whether CMS displays an error message if it cannot release the storage as requested. Acceptable values are:

YES

indicates CMS displays an error message. This is the default value.

NO

indicates CMS does not display an error message.

(reg)

CMSSTOR checks the value of the specified register and, if it is 0, sets MSG to NO. If the register contains a nonzero value, the macro sets MSG to YES.

(addr,mask)

defines a single bit in storage that sets the value of the MSG parameter. *addr* is the address of a byte in storage and *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then MSG is set to NO. If the bit is 1, then MSG is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the MSG parameter as

```
MSG=(APPFLAG,X'80')
```

To set the value of the MSG parameter at assembly time, specify MSG=YES or MSG=NO. To set the value at execution time, specify MSG=(*reg*) or MSG=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. Acceptable values are:

'ABEND'

abends the program. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register. Registers 2-12 are valid.

passes control to the next sequential instruction.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr,mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If CMSSTOR RELEASE is successful, it stores a 0 return code in register 15 and leaves in register 0 the number of BYTES/DWORDS released. Note that the number of BYTES/DWORDS in register 0 is a doubleword multiple.
2. Subpool naming:
 - a. Subpool names can use any characters. (Note that this implies that subpool names are case sensitive and SUBPOOL='XYZ' is not the equivalent of SUBPOOL='xyz'.)
 - b. The subpool names DMSxxxxx are reserved for system use, and the names USER, USERG, and NUCLEUS are for reserved system subpools; otherwise, there are no restrictions on the names you give subpools.
3. CMSSTOR always treats the address on the ADDR= parameter as a 31-bit address. Therefore, if a program specifies a 24-bit address for ADDR=, it (the program) must make sure that bits 1-7 are 0s. Bit 0 is always ignored.
4. You can produce reentrant code with the standard macro form or with a combination of the MF=L and MF=(E,*addr*) forms of CMSSTOR RELEASE. To build a parameter list without using parameter substitutions, you must meet the following restrictions:

On the standard macro form:

- The DWORDS/BYTES parameter can be any valid value and is always loaded into register 0.
- The ADDR= parameter must be a register form or constant. Do not use register 1 for this when using the standard macro form.
- The SUBPOOL= parameter, if specified, must be a quoted constant (for example, SUBPOOL='XYZ' or SUBPOOL=('XYZ',GLOBAL)).

For example,

```
CMSSTOR RELEASE, BYTES=(R0), ADDR=(R3), ERROR=*
```

On the MF=(E,addr) form:

- The only parameters you can specify are BYTES/DWORDS, TYPICAL=, and ERROR=. The only parameter you can specify in variable form is the DWORDS/BYTES parameter because the value is stored into register 0 before invoking the storage management routines. If you specify this parameter, you must specify its value in register notation with the MF=(E,addr) macro format. If specified as a number, it would be stored into the parameter list before being loaded into register 0. As a result, the previously-set default for DWORDS/BYTES would be replaced by the specified number.
 - Defaults are not recognized on the MF=(E,addr) form; specifying a default causes storing into the parameter list. For example, specifying MSG=YES on the MF=(E,addr) form would not be reentrant.
 - To allow the combination of MF=L and MF=(E,addr) macro formats to create inline reentrant parameter lists, the MF=(E,addr) format does not store default values into the parameter list. For example, while other CMS macros with an MF=(E,addr) format store the name of the function being called, CMSSTOR does not store the function name into the parameter list. The MF=(L,addr) macro format can be used to store the function name and other values into an uninitialized parameter list.
5. The CMSSTOR RELEASE macro can release specific blocks of storage allocated by the CMSSTOR OBTAIN macro. In addition, a specific subpool of such storage may be released or deleted as follows:

Table 13. Releasing Storage Allocation

Action	USER or USERG Subpool	NUCLEUS Subpool	USER/ Shared	GLOBAL Subpool SYSTEM= YES	GLOBAL Subpool SYSTEM= NO
SVC 202 or CMSCALL Termination	retain	retain	delete	retain	retain
abend recovery	release	retain	delete	retain	delete
SUBPOOL DELETE	error	error	delete	delete	delete
SUBPOOL RELEASE	error	error	release	release	release

Note:

- SYSTEM=YES—as specified on the SUBPOOL macro, the GLOBAL subpool is to survive abend processing.
 - SYSTEM=NO—as specified on the SUBPOOL macro, the GLOBAL subpool is not to survive abend processing.
 - Error—indicates the action is not allowed for the particular type of subpool.
 - Retain—the subpool is not affected by the action.
 - Release—the subpool is RELEASED.
 - Delete—the subpool is DELETED.
6. To release storage above 2 GB when z/CMS is running in the virtual machine, you must specify SUBPOOL='USERG'. The ADDR= parameter must contain the 64-bit address of the storage to be released and the BYTES= parameter must contain the number of pages to be released. For example, the following statement will release 1 page of storage at the 64-bit address specified in register 10:

```
CMSSTOR RELEASE, BYTES=1, ADDR=(R10), SUBPOOL='USERG'
```

The DWORDS= and TYPICAL= parameters should not be specified. TYPICAL=SVC is the default, and it is the only TYPICAL value supported when releasing storage above 2 GB.

SUBPOOL='USERG' is supported only under z/CMS. Before assembling a program that uses SUBPOOL='USERG', you must issue the GLOBAL MACLIB command to specify the DMSZGPI macro library ahead of the DMSGPI library. For example:

```
global maclib dmszgpi dmsgpi
```

7. Although CMSSTOR and OS/MVS macro calls for storage management are implemented via the same CMS storage management subsystem, mixing these calls is not recommended. The default subpool

name in which CMSSTOR obtains storage is USER, while the default subpool name used by the OS/MVS macro calls (GETMAIN/FREEMAIN) is DMSOS000. Also, GETMAIN automatic storage cleanup might be affected by the CMS STORECLR setting.

Return Codes

When CMSSTOR RELEASE completes, register 15 contains one of the following return codes:

Code

Meaning

0

Normal completion. Register 0 specifies the amount of storage released.

2

The USER key storage pointers were destroyed.

3

The NUCLEUS key storage pointers were destroyed.

5

The size value specified on the BYTES or DWORDS parameter was invalid. This error occurs if the specified value is not positive.

6

The specified block of storage was never allocated by CMSSTOR OBTAIN. This error can occur because the block:

- Does not lie entirely within the virtual machine's free storage area.
- Crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS type storage.
- Overlaps another block already on the free storage chain.

7

The specified address is not doubleword aligned.

9

An unexpected and unexplained error occurred in the storage management routine.

10

The block specified on the CMSSTOR RELEASE SUBPOOL parameter does not match the subpool name specified on the CMSSTOR OBTAIN macro. No storage is released.

11

A register specified for the ADDR= parameter is not in the range 2-12.

COMPSWT



Purpose

Use the COMPSWT macroinstruction to turn the compiler switch (COMPSWT) flag on or off.

Parameters

Required Parameters:

ON

turns the COMPSWT flag on. When this flag is on, any program called by a LINK, LOAD, XCTL, or ATTACH macroinstruction must be a module file with a file type of MODULE; CMS first searches private storage for that entry point already loaded. If it is not found then CMS uses the LOADMOD command to load it.

OFF

turns the COMPSWT flag off. When this flag is off, any program called by a LINK, LOAD, XCTL, or ATTACH macroinstruction must be a relocatable object module residing in a file with a file type of TEXT, LOADLIB, or TXTLIB; CMS uses the INCLUDE command to load it.

Note: COMPSWT is initially set to OFF by the compiler.

Optional Parameter:

label

is an optional assembler label for the statement.

CONSOLE

Purpose

Use the CONSOLE macroinstruction to access CMS full-screen console services. The CONSOLE macro performs 3270 I/O operations, including building the Channel Command Word (CCW), issuing the DIAGNOSE code X'58', SSCH instruction, or SIO instruction, waiting for the I/O to complete, and checking any error status from the device. Applications must construct a valid 3270 data stream to write to the screen and a 3270 data stream is returned when a CONSOLE READ is performed. For line-mode I/O operations or for 3215-type devices, use the LINERD and LINEWRT macros.

The CONSOLE macro allows programs to open 'paths' to a display device. The CONSOLE macro coordinates screen use by indicating to an application doing a write that the screen was last updated by another 'path' and that the screen must be reformatted. Full-screen applications thus do not have to rewrite the entire screen every time a write is done.

The basic functions of the CONSOLE macro are:

CONSOLE CLOSE

Closes a specific path to a device.

CONSOLE EXCP

Lets you specify your own channel program to read or write I/O. Note that CONSOLE EXCP requires the user to distinguish between dedicated devices and the virtual console, because DIAGNOSE code X'58' CCWs must be provided for I/O to the console.

CONSOLE MODIFY

Changes the exit address, user word, or RESET parameter setting without closing and reopening the path.

CONSOLE OPEN

Opens a specific path to a device.

CONSOLE QUERY

Gets information about the device attributes (DIAGNOSE code X'24' and X'8C' information) or about a specific path and its associated device (if the path is open).

CONSOLE READ

Reads information from the display device.

CONSOLE WAIT

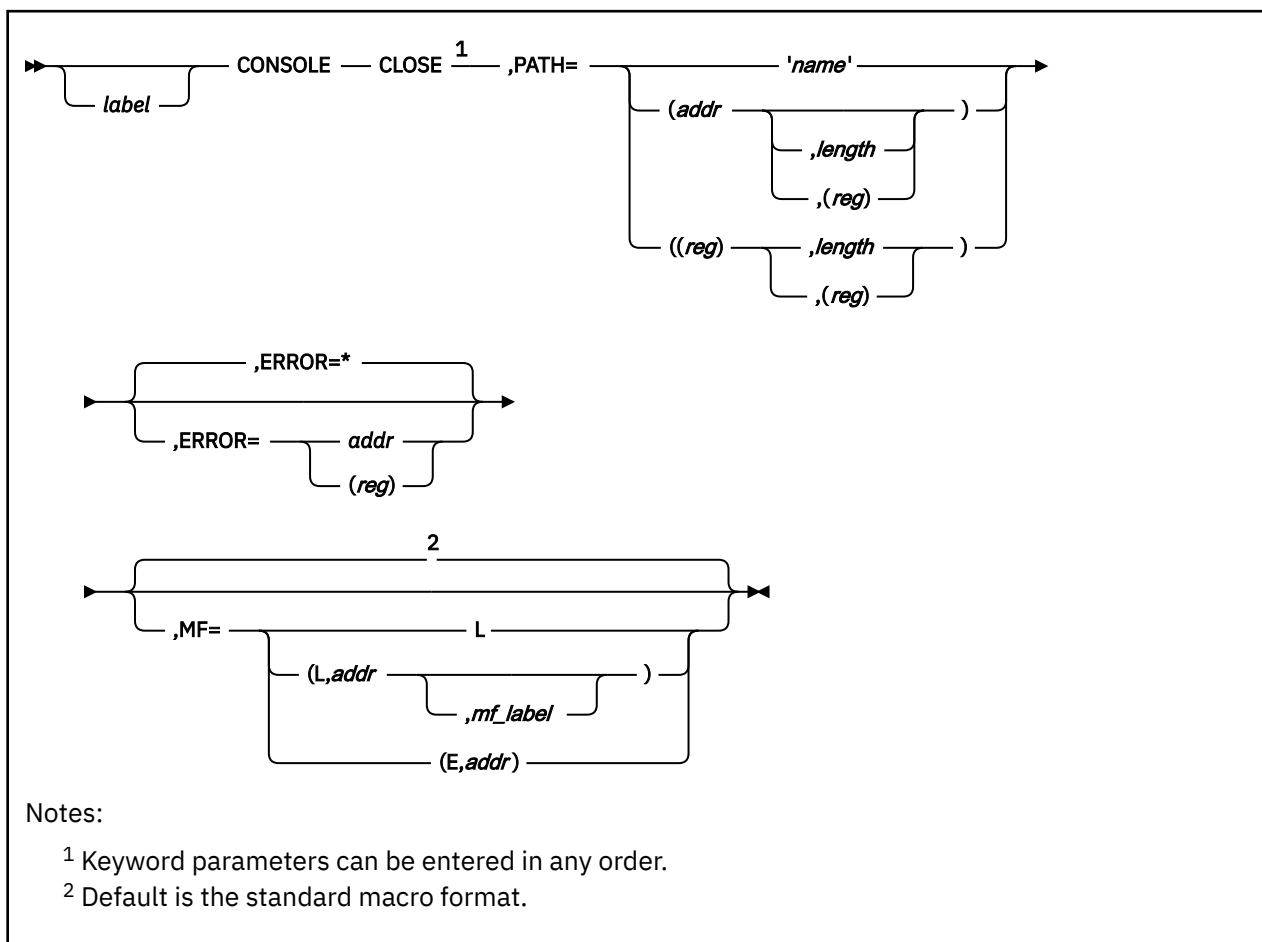
Waits for an interrupt (for example, an I/O interrupt from the console device).

CONSOLE WRITE

Writes buffers that have 3270 data streams built by the application.

For more information on how to use the CONSOLE macro, see the [*z/VM: CMS Application Development Guide for Assembler*](#).

CONSOLE CLOSE



Purpose

Use **CONSOLE CLOSE** to close a path to a device.

Parameters

Required Parameters:

CLOSE

closes a specific path to a device.

PATH=

specifies the path to be closed.

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the **CONSOLE** request is processed.

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If RESET=YES was specified when the path was opened or modified, a CP RESET command is issued when you close the last path to a dedicated 3270 device.

Return Codes

Upon completion of the CONSOLE CLOSE function, register 15 contains one of the following return codes:

Code

Meaning

0

The path is closed.

3

The requested path has been closed, but other paths to the associated device are still open.

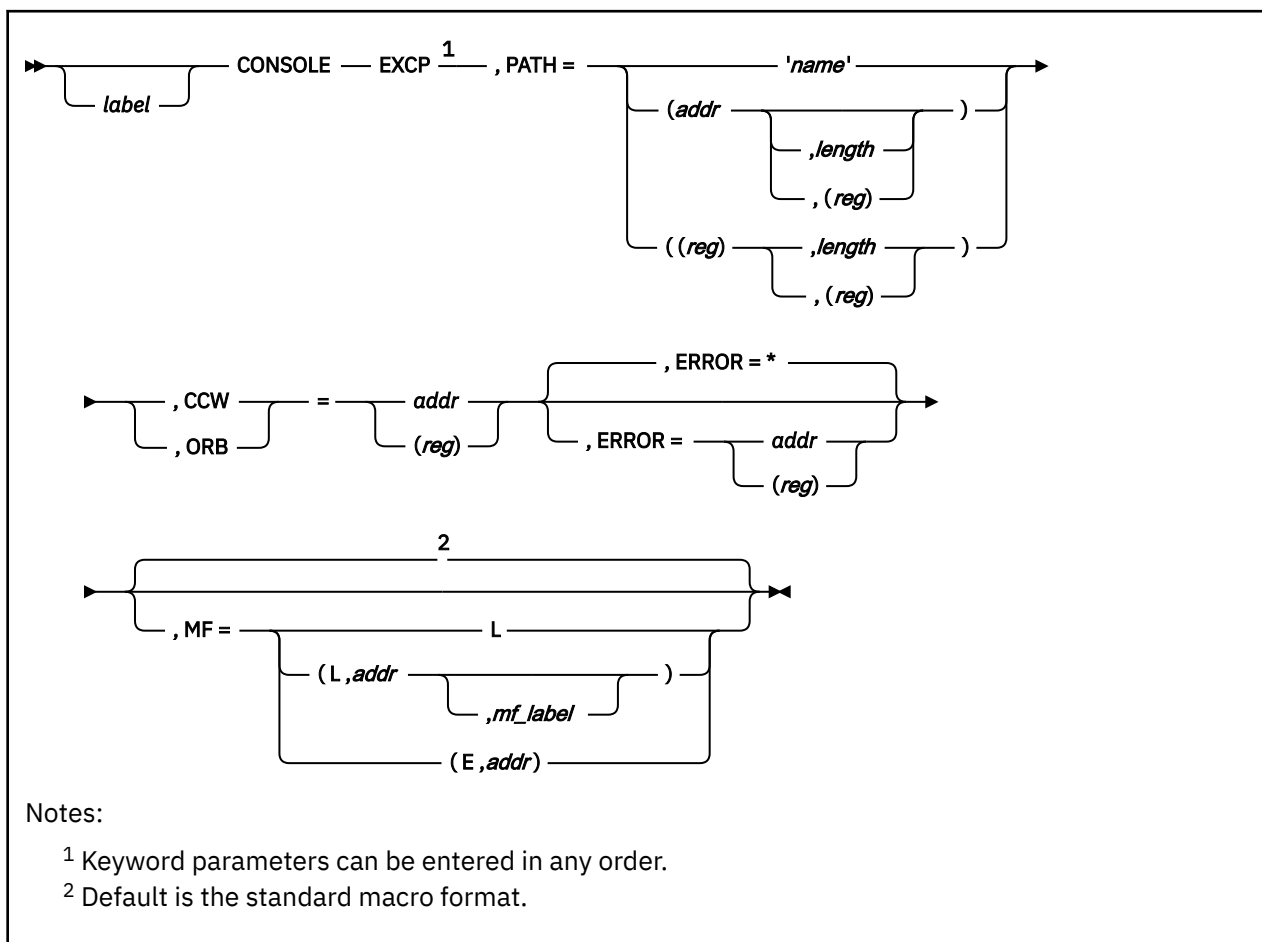
24

The parameter list is invalid; a path was not specified.

28

Path not found.

CONSOLE EXCP



Purpose

Use CONSOLE EXCP when you specify your own channel program to read or write.

Parameters

Required Parameters:

EXCP

lets you specify your own channel program to read or write full-screen I/O.

PATH=

specifies the path name.

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the CONSOLE request is processed.

(addr, length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

CCW=

specifies the address of a channel program, which consists of one or more format-0 CCWs that indicate the operation(s) to be performed. You must specify either CCW or operation request block (ORB), but not both. CCW specifies that the necessary ORB will be built by Console indicating format-0 CCWs. Acceptable values are:

addr

specifies the address of a channel program.

(reg)

specifies a register that contains the address of a channel program. Valid registers are 2-12 enclosed in parentheses.

ORB=

specifies the address of an operation request block (ORB) for a channel program. The ORB can indicate either format-0 or format-1 CCWs. The ORB must contain a field that points to a channel program for the operation(s) to be performed. You must specify either CCW or ORB, but not both. ORB is invalid for paths to the virtual console. Acceptable values are shown in the previous CCW parameter.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If you issue the EXCP request, you are responsible for generating a valid channel program; therefore, you require a knowledge of the virtual machine architecture and the console support implementation. There is no attempt made to validate the channel program or to convert it to a form appropriate to the implementation. **The EXCP parameter is not recommended for use with a virtual console.**
2. Depending on whether CP console communications services or the logical device support facility is being used instead of CP native support, some I/O errors may not be reflected to the application issuing a DIAGNOSE code X'58', SIO instruction, or CONSOLE macro. In this case, the CMS console facility may see a channel end/device end from the I/O and therefore return a return code of 0. If an application that uses the console facility does not get expected results, the data stream or CCW (for EXCP) should be checked for user errors.
3. If you have issued a CONSOLE READ and are waiting for input when CP breaks in and writes a screen (such as a CP warning message), the read is performed and a channel end/device end is returned to CMS. The Console Facility gives your application a return code of 0. Your application should examine the data stream attention identification (AID) to determine whether there are any modified fields to process. An AID byte of X'60' indicates no operation or an unsolicited attention. In this case, the CSW/SCSW will contain X'8E' on the next fullscreen write, causing CONSOLE to give return code 32.
4. All applications whose paths are using the same device are notified of device changes through return code 2 the next time I/O is done. The device characteristics may have changed because the device was disconnected and then reconnected, or another application attempted an OPEN and the device characteristics do not match what is currently in the device table. On the next I/O, regardless of the type of change in device characteristics, CONSOLE will not attempt the I/O. The return code of 2 is returned to the application indicating to query, if necessary, and then reissue the I/O.

Return Codes

Upon completion of the CONSOLE EXCP function, register 15 contains one of the following return codes:

Code

Meaning

0

I/O successful. See Usage Notes [“2” on page 112](#) and [“3” on page 112](#) for additional information.

1

A path has been opened to the virtual device, but no real device is currently connected to that virtual device.

2

You should issue a CONSOLE QUERY for the device before any more I/O is requested. See Usage Notes for additional information.

24

The parameter list is invalid; the function name is unknown, a required parameter is missing, or conflicting options were specified.

28

Path not found. This return code occurs if the path was never opened, or if a device receives an I/O error because it was detached after the path was opened. The console facility closes all paths associated with the device, and indicates that the path no longer exists.

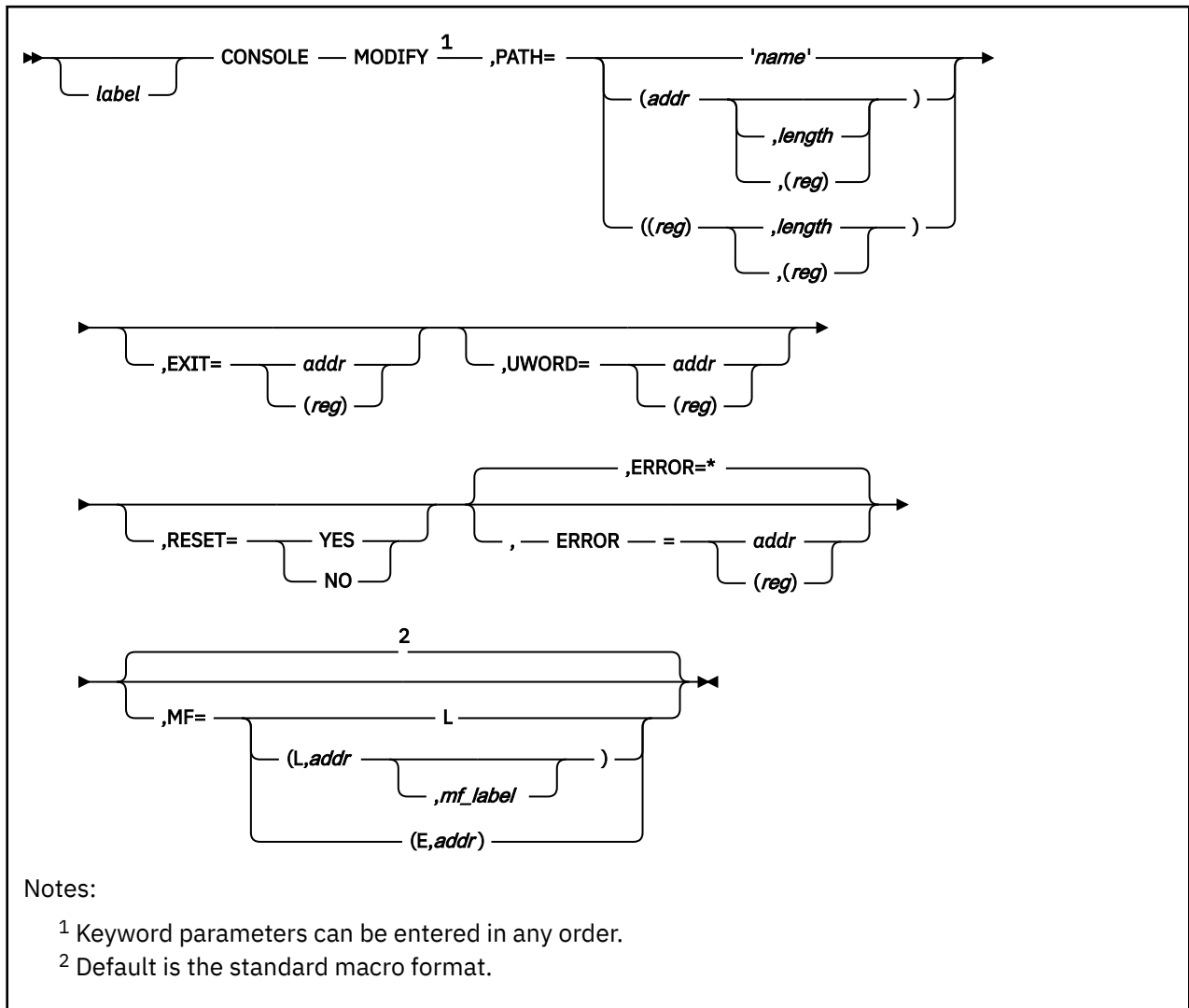
32

A full-screen read or write was requested, but another application wrote to the screen. For a read request, the screen may not belong to your application. An Erase/Write must be issued to reformat the screen and return ownership to the current application.

100

An I/O error has occurred. You can obtain the CSW/SCSW status by issuing a CONSOLE QUERY and specifying a buffer that contains the information.

CONSOLE MODIFY



Purpose

Use **CONSOLE MODIFY** to dynamically set up or change the **EXIT=**, **UWORD=**, or **RESET=** parameter values without closing and reopening the path.

Parameters

Required Parameters:

MODIFY

changes the exit address, user word, or **RESET** setting without closing and reopening the path. If an application did not originally set up one of these parameters when the path was opened, it can also be used to set these parameters up any time after the path has been opened.

PATH=

specifies the path name. Acceptable values are:

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the **CONSOLE** request is processed.

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

EXIT=

specifies the address of a routine to get control in the event of an unsolicited interrupt. The address of the exit routine can be initially set up, changed, or deleted by the MODIFY function. As with CONSOLE OPEN, the exit routine will be established in the same addressing mode (AMODE) as the application issuing the CONSOLE MODIFY call. Acceptable values are:

addr

specifies the address of the exit routine as an assembler expression. CMS passes the address to the exit routine.

(reg)

specifies a register that contains the address of the exit routine. Valid registers are 2-12 enclosed in parentheses. If you specify a register, CMS passes the contents of the register to the exit routine.

For more information on exit routines, see the [“Usage Notes” on page 115](#) in the description of the CONSOLE OPEN function.

UWORD=

specifies an optional fullword parameter you can pass to the exit routine. You can set up a user word or change the contents of the UWORD= parameter specified on a CONSOLE OPEN request. When the exit routine gains control, register 0 contains the UWORD, which can contain any value. If you do not specify a value, UWORD remains unchanged. Acceptable values are:

addr

passes the value of the expression to the routine.

(reg)

passes the contents of the register to the routine. Valid registers are 2-12 enclosed in parentheses.

RESET=

specifies whether a CP RESET command will be issued by the Console Facility when the specified path is the last path to a dedicated device and the path is being deleted. If this parameter is not specified, no default is assumed and the setting remains the same as it was when the path was established by the CONSOLE OPEN request. If the RESET= parameter is specified for the virtual console, it is ignored. Acceptable values are:

YES

specifies that a CP RESET command will be issued when the specified path is the last path to a dedicated device and the path is being deleted.

NO

specifies that a CP RESET command will not be issued by the Console Facility and the application must issue a CP RESET to free up the device.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If a path is opened to the device and the device is subsequently detached (using a CP DETACH command) or redefined at a new address (using a CP DEFINE or REDEFINE command), then all paths to the device are cleared.

Return Codes

Upon completion of the CONSOLE MODIFY function, register 15 contains one of the following return codes:

Code

Meaning

0

The function completed successfully.

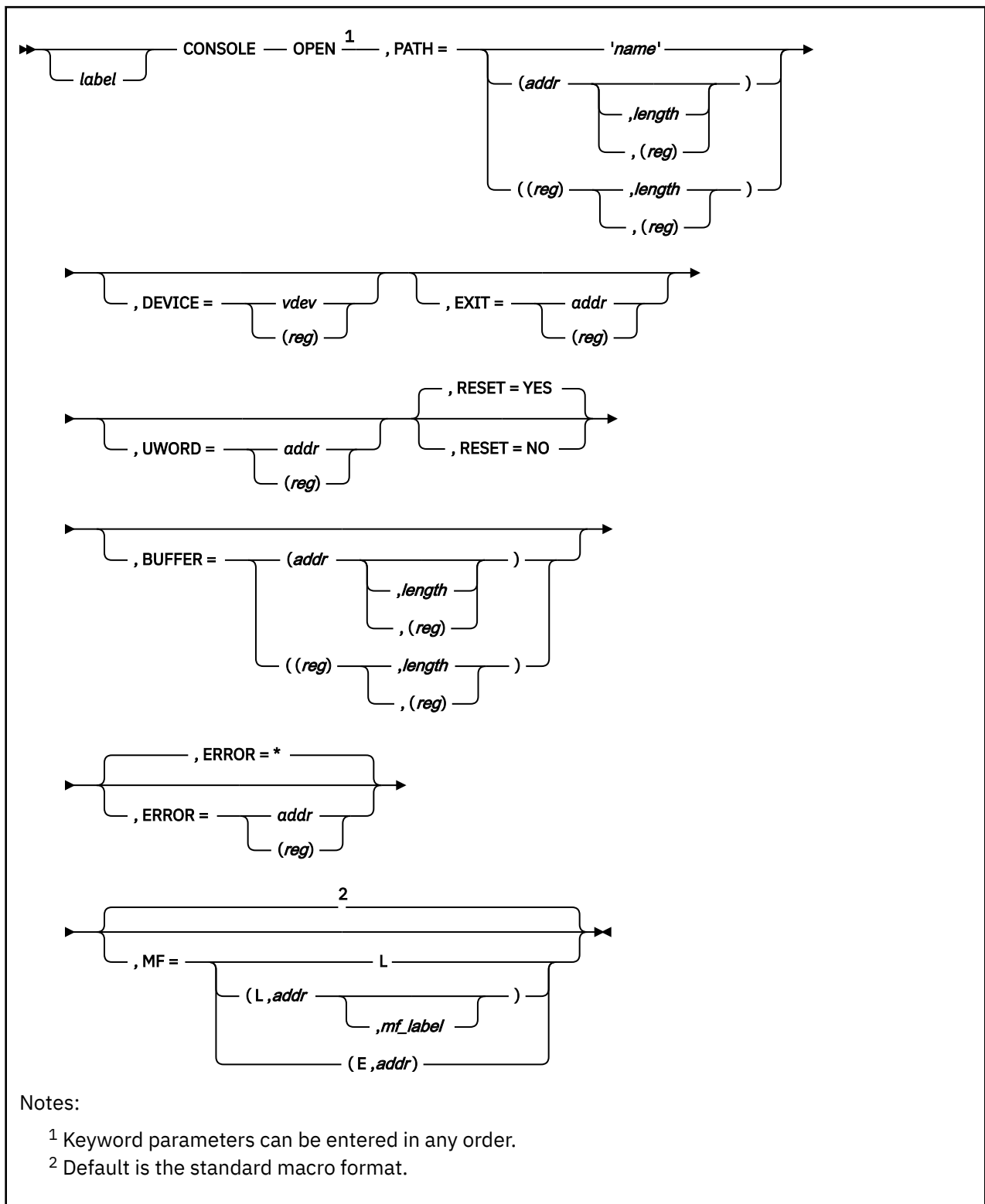
24

The parameter list is invalid; a path was not specified.

28

The path was not found.

CONSOLE OPEN



Purpose

Use **CONSOLE OPEN** to define a path to a device.

Parameters

Required Parameters:

OPEN

opens a specific path to a device.

PATH=

assigns a unique name to the path. Acceptable values are:

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the CONSOLE request is processed.

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

DEVICE=

specifies the virtual device number of the console or dedicated device to be associated with the path. If you omit device on the standard format or list format, the default value is the virtual console. The execute and complex list formats (MF=E and MF=(L,addr)) do not substitute default values. Acceptable values are:

vdev

defines the device number as a hexadecimal constant. A decimal constant of -1 or a hexadecimal constant of 'FFFFFFFF' specifies the virtual console.

(reg)

specifies a register that contains the device number in the low-order 2 bytes and zeros in the rest. Valid registers are 2-12 enclosed in parentheses.

EXIT=

specifies the address of a routine to get control in the event of an unsolicited interrupt. Acceptable values are:

addr

specifies the address of the exit routine as an assembler expression. CMS passes the address to the exit routine.

(reg)

specifies a register that contains the address of the exit routine. Valid registers are 2-12 enclosed in parentheses. If you specify a register, CMS passes the contents of the register to the exit routine.

For more information on exit routines, see [“Usage Notes” on page 119](#).

UWORD=

specifies an optional fullword parameter you can pass to the exit routine. When the exit routine gains control, register 0 contains the UWORD, which can contain any value. If you do not specify a value, a UWORD of F'0' is passed. Acceptable values are:

addr

passes the value of the expression to the routine.

(reg)

passes the contents of the register to the routine. Valid registers are 2-12 enclosed in parentheses.

BUFFER=

specifies the area where the CONSOLE function returns data about the device on the path being opened. Acceptable values are:

(addr,length)

specifies the buffer address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the buffer address and the length as an absolute expression. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

((reg), (reg))

specifies a register that contains the address of the buffer and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

The buffer is mapped by the CQYSECT macro. If the buffer length is less than the length of CQYSECT, the data in the buffer is truncated. Upon completion of a CONSOLE OPEN function, if register 15 contains return codes 0 or 28, register 0 contains the length of the data moved into the buffer. CQYSECT provides length values.

RESET=

specifies whether a CP RESET command will be issued by the Console Facility when the specified path is the last path to a dedicated device and the path is being deleted. This parameter is applicable only for dedicated devices. If the RESET= parameter is specified for the virtual console, it is ignored. Acceptable values are:

YES

specifies that a CP RESET command will be issued when the specified path is the last path to a dedicated device and the path is being deleted. This is the default value.

NO

specifies that a CP RESET command will not be issued by the Console Facility and the application must issue a CP RESET to free up the device.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The addressing mode (AMODE) of the exit routine is the same as the AMODE of the program that issued the CONSOLE OPEN to define the exit routine.

The exit routine should be prepared to handle all interrupts it receives. For more information on Console exits, see the *z/VM: CMS Application Development Guide for Assembler*. You are responsible for establishing proper entry and exit linkage for your routine. When your exit routine receives control, the significant registers contain:

Registers

Contents

R0

UWORD (user word)

R13

Address of 72-byte save area

R14

Return address

R15

Entry point address

Your routine must return control to the address specified in register 14 upon entry.

2. To obtain information about the CSW/SCSW at the time of interrupt, the interrupting device address, or other information about the path and associated device, the exit should issue the CONSOLE QUERY function, specifying PATH and providing a buffer. Use the CQYSECT macro to map the information moved into the buffer.

3. When an unsolicited interrupt occurs, CMS gives control to the exit routine of the path that did the last I/O. If no previous I/O was done, CMS gives control to the exit routine of the path that was last opened.

The exit routine receives control as an extension of CMS I/O interrupt handling; the PSW is set up with a system storage key and is disabled for interrupts. Register 0 contains the user word (see UWORD parameter description).

If the interrupt results from a line mode operation issued by CMS, the interrupt is passed to CMS rather than the exit routine. While in fullscreen mode, virtual console exits receive only attention interrupts, while dedicated device exits can receive all interrupts. For more information on device exits, see the *z/VM: CMS Application Development Guide for Assembler*.

4. To avoid confusion, you should not define an exit routine (using the EXIT parameter) and an HNDINT or HNDIO interrupt handler for the same device.

5. If a path is opened to the device and the device is subsequently detached (using a CP DETACH command) or redefined at a new address (using a CP DEFINE or REDEFINE command), then all paths to the device are cleared.

Return Codes

Upon completion of the CONSOLE OPEN function, register 15 contains one of the following return codes:

Code

Meaning

0

The path is opened. If a buffer is provided, the length of the data stored in the buffer is returned in register 0.

1

A path has been opened to the virtual device, but no real device is currently connected to that virtual device.

24

The parameter list is invalid; a path was not specified.

28

The path is already open. If a buffer is provided, the length of the data stored in the buffer is returned in register 0.

40

The virtual device is invalid or not defined.

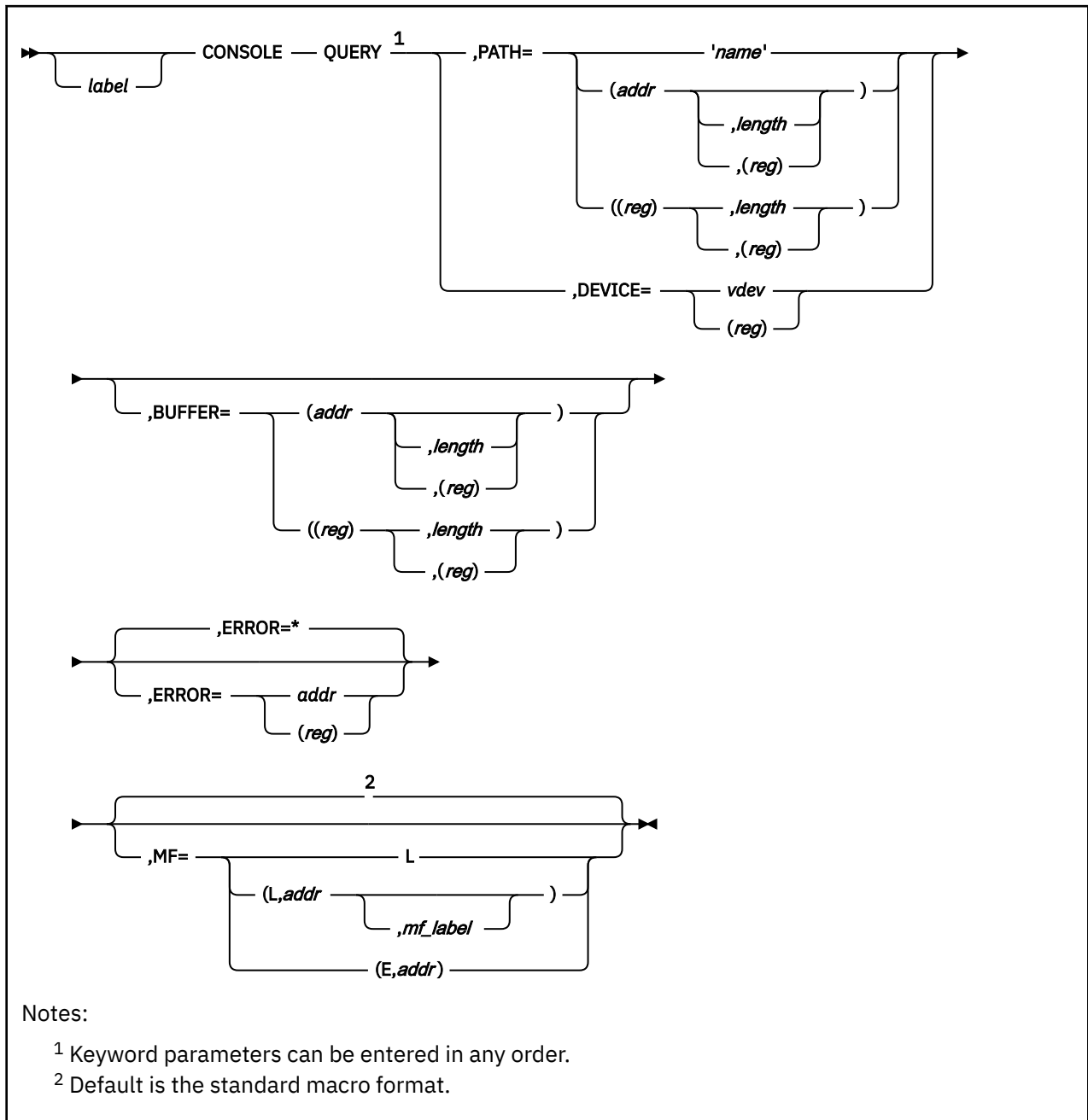
88

The virtual device is not supported by the Console Facility for full-screen I/O. For a typewriter-type device (TTY), a Console path is not opened and device characteristics are not saved in a Console device entry. However, if the application provides a buffer, the DIAGNOSE code X'24' and any DIAGNOSE code X'8C' is returned in the buffer.

104

Unable to obtain storage to process the request.

CONSOLE QUERY



Purpose

Use **CONSOLE QUERY** to get information about a specific device or about a specific path and its corresponding device.

You must specify either the **PATH** or **DEVICE** parameter. If you specify both, **PATH** is ignored.

Parameters

Required Parameters:

QUERY

gets information about the device attributes or about a specific path and its associated device (if the path is open).

PATH=

specifies the path name. Acceptable values are:

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the CONSOLE request is processed.

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

DEVICE=

specifies the virtual device number of the console or dedicated device to be queried. Acceptable values are:

vdev

defines the device address as a hexadecimal constant. A decimal constant of -1 or a hexadecimal constant of X'FFFFFFFF' specifies the virtual console.

(reg)

specifies a register that contains the device address in the low-order 2 bytes and zeros in the rest. Valid registers are 2-12 enclosed in parentheses.

Because this function explicitly queries a given path or device, there is no default value for the DEVICE= parameter. You must specify either the PATH or DEVICE parameter. If you specify both, PATH is ignored.

Optional Parameters:

label

is an optional assembler label for the statement.

BUFFER=

specifies the area where the CONSOLE function returns data about the device or path being queried. Acceptable values are:

(addr,length)

specifies the buffer address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the buffer address and the length as an absolute expression. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

((reg), (reg))

specifies a register that contains the address of the buffer and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

The buffer is mapped by the CQYSECT macro. If the buffer length is less than the length of CQYSECT, the data in the buffer is truncated. Upon completion of a CONSOLE QUERY function, if register 15 contains return code 0, register 0 contains the length of the data moved into the buffer. CQYSECT provides length values.

Also, if you specify PATH and the buffer is large enough, CONSOLE returns device and path information. If you specify DEVICE, CONSOLE returns information about the device only.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. A CONSOLE QUERY PATH extracts whatever information is in the path and device entries. This information reflects the state of the path and associated device either when the path was opened or at the time of the last I/O if any I/O was performed for that path. Therefore, when disconnecting and reconnecting to another device, there is a possibility that the original device information will be returned when a CONSOLE QUERY PATH is done.
2. A CONSOLE QUERY DEVICE updates any device information in the console device entry by issuing a new DIAGNOSE code X'24' and X'8C', and returns the latest device information to your application.

Return Codes

Upon completion of the CONSOLE QUERY function, register 15 contains one of the following return codes:

Code

Meaning

0

If querying a path, the path is open. If querying a device, the device is defined, connected to a real device, and supported by the console facility. If a buffer is provided, the length of data stored in the buffer is returned in register 0.

CONSOLE QUERY

1

The virtual device is defined and supported by the console facility, but it was not connected to a real device when the console facility issued a DIAGNOSE code X'24' to the device.

24

The parameter list is invalid; a path or a device must be specified.

28

Path not found.

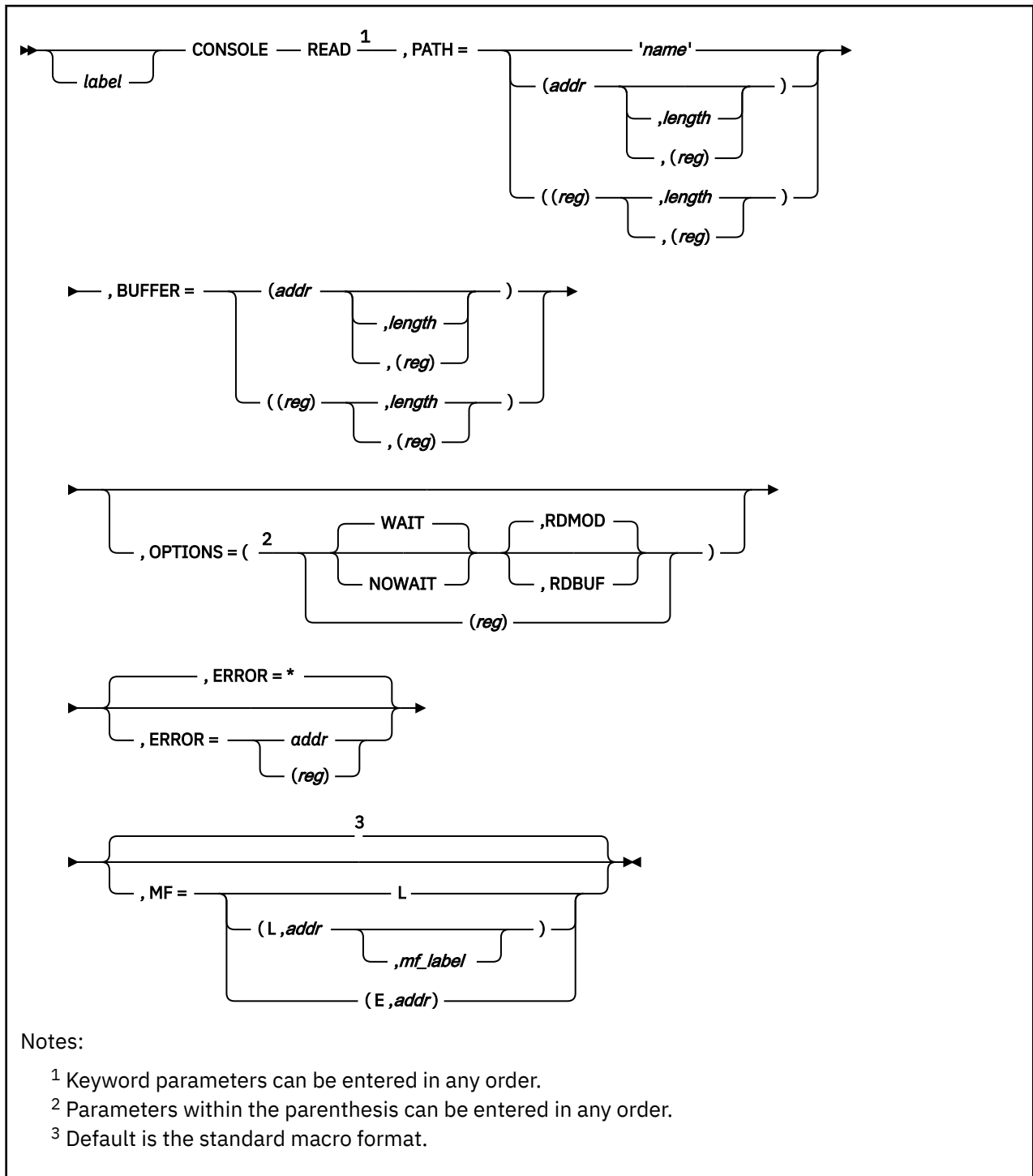
40

The virtual device is invalid or not defined.

88

The virtual device associated with the path is not supported by the console facility for full-screen I/O.

CONSOLE READ



Purpose

Use `CONSOLE READ` to read from a display device.

Parameters

Required Parameters:

READ

reads information from the display device.

PATH=

specifies the path name. Acceptable values are:

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the CONSOLE request is processed.

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

BUFFER=

specifies the address of an area in storage where the data is to be read into. After the read, if register 15 contains return code 0, register 0 contains the length of the data moved into the buffer. The BUFFER parameter is required on CONSOLE READ.

(addr,length)

specifies the buffer address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the buffer address and the length as an absolute expression. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

((reg), (reg))

specifies a register that contains the address of the buffer and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

OPTIONS=

specifies optional processing for this buffer.

Note: Options may be specified in any order. For example, (WAIT,RDMOD) or (RDMOD,WAIT) are both valid. Acceptable values are:

WAIT

specifies that processing of the request is suspended until an I/O interrupt is received from the device after the last write operation is complete. If WAIT or NOWAIT is not specified, then WAIT is the default.

NOWAIT

specifies that the read request is processed immediately.

RDMOD

specifies that the request is processed as Read Modified and transmits only the modified fields from the screen. If RDMOD or RDBUF is not specified, RDMOD is the default.

RDBUF

specifies that the request is processed as Read Buffer and transmits the entire contents of the screen.

(reg)

specifies a register 2-12, whose low-order byte contains the option or options to be used. The hexadecimal value of the byte must be set to one of the following:

```
X'00' = WAIT, RDMOD
X'01' = WAIT, RDBUF
X'80' = NOWAIT, RDMOD
X'81' = NOWAIT, RDBUF
```

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. Depending on whether CP console communications services or the logical device support facility is being used instead of CP native support, some I/O errors may not be reflected to the application issuing a DIAGNOSE code X'58', SIO instruction, or CONSOLE macro. In this case, the CMS console facility may see a channel end/device end from the I/O and therefore return a return code of 0. If an application that uses the console facility does not get expected results, the data stream or CCW (for EXCP) should be checked for user errors.
2. If you have issued a CONSOLE READ and are waiting for input when CP breaks in and writes a screen (such as a CP warning message), the read is performed and a channel end/device end is returned to CMS. The Console Facility gives your application a return code of 0. Your application should examine the data stream attention identification (AID) to determine whether there are any modified fields to process. An AID byte of X'60' indicates no operation or an unsolicited attention. In this case, the CSW/SCSW will contain X'8E' on the next fullscreen write, causing CONSOLE to give return code 32.

3. All applications whose paths are using the same device are notified of device changes through return code 2 the next time I/O is done. The device characteristics may have changed because the device was disconnected and then reconnected, or another application attempted an OPEN and the device characteristics do not match what is currently in the device table. On the next I/O, regardless of the type of change in device characteristics, CONSOLE will not attempt the I/O. The return code of 2 is returned to the application indicating to query, if necessary, and then reissue the I/O.

Examples

The following are some examples of a CONSOLE READ invocation using the register specification of the OPTIONS parameter:

LA	R2,RDBUF	Read full buffer	
CONSOLE	READ,PATH='TEST',BUFFER=(BUF,BUFL),OPTIONS=((R2))		X
*			
LA	R2,NWRDBF	Nowait, Read buffer	
CONSOLE	READ,PATH='TEST',BUFFER=(BUF,BUFL),OPTIONS=((R2))		X
*			
LA	R2,0	Wait, Read Modified	
CONSOLE	READ,PATH='TEST',BUFFER=(BUF,BUFL),OPTIONS=((R2))		X
	.		
RDBUF	EQU X'01'		
NWRDBF	EQU X'81'		
R2	EQU 2		
BUF	DS CL2000		
BUFL	EQU *-BUF		

Return Codes

Upon completion of the CONSOLE READ function, register 15 contains one of the following return codes:

Code

Meaning

0

I/O successful. The length of data stored in the provided buffer is returned in register 0. See Usage Notes [“1” on page 127](#) and [“2” on page 127](#) for additional information.

1

A path has been opened to the virtual device, but no real device is currently connected to that virtual device.

2

You should issue a CONSOLE QUERY for the device before any more I/O is requested. See Usage Notes for additional information.

24

The parameter list is invalid; the function name is unknown, a required parameter is missing, or conflicting or invalid options were specified.

28

Path not found. This return code occurs if the path was never opened, or if a device receives an I/O error because it was detached after the path was opened. The console facility closes all paths associated with the device, and indicates that the path no longer exists.

32

A full-screen read was requested, but another application wrote to the screen. For a read request, the screen may not belong to your application. An Erase/Write must be issued to reformat the screen and return ownership to the current application. See Usage Note [“2” on page 127](#) for more information.

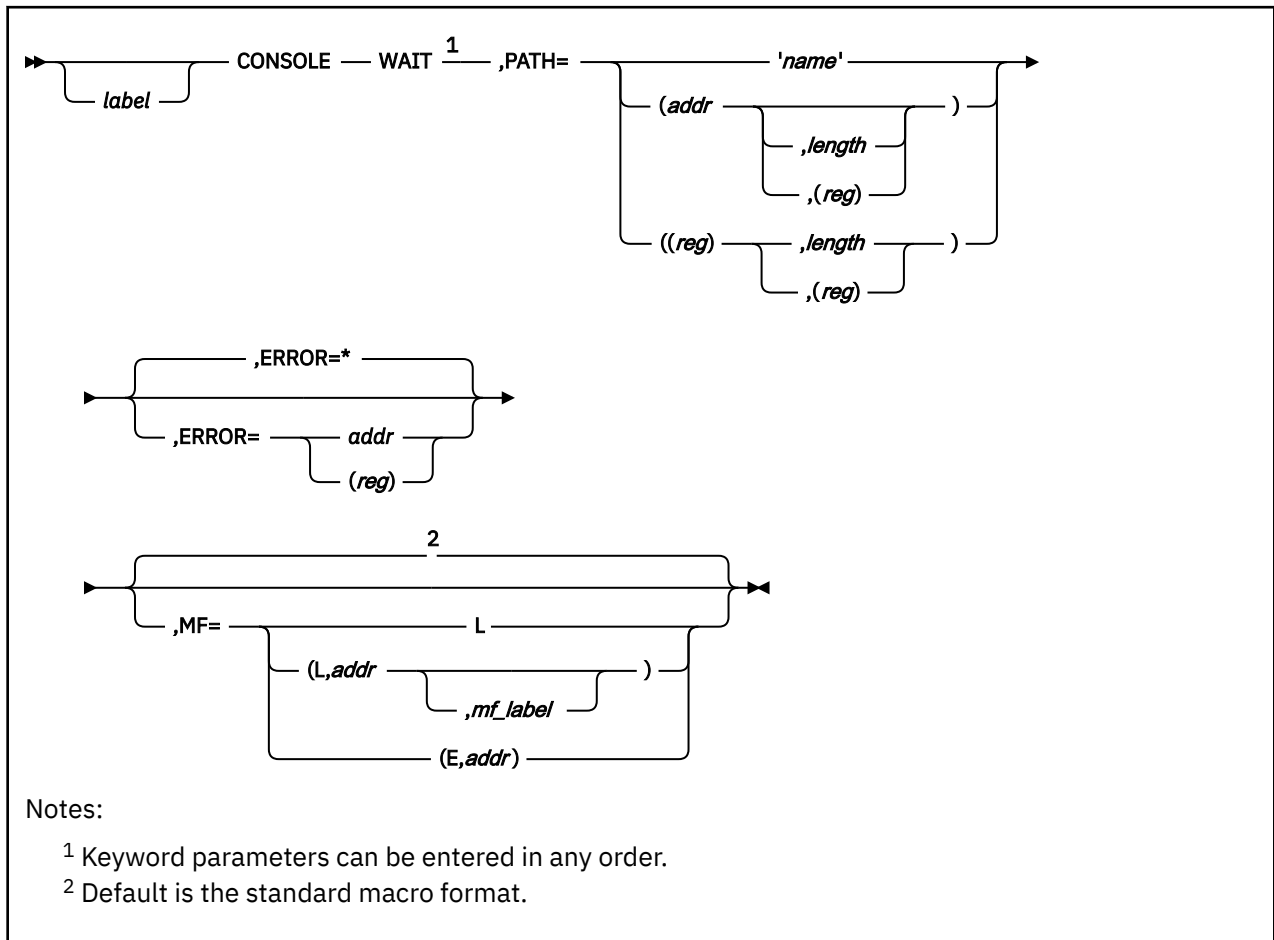
100

An I/O error has occurred. You can obtain the CSW/SCSW status by issuing a CONSOLE QUERY and specifying a buffer that contains the information.

104

Insufficient storage space is available for an IDAL (Indirect Address List) on a CONSOLE READ request.

CONSOLE WAIT



Purpose

Use **CONSOLE WAIT** to wait for an interrupt from a display device.

Parameters

Required Parameters:

WAIT

waits for an interrupt.

PATH=

specifies the path name. Acceptable values are:

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the **CONSOLE** request is processed.

(addr, length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. A CONSOLE WAIT must be followed by a CONSOLE READ after the interrupt is received. Do not use the WAIT option on CONSOLE READ in this case. Issuing the WAIT option on CONSOLE READ would cause another enabled wait before the read is issued, thus resulting in a hung terminal condition.
2. All applications whose paths are using the same device are notified of device changes through return code 2 the next time I/O is done. The device characteristics may have changed because the device was disconnected and then reconnected, or another application attempted an OPEN and the device characteristics do not match what is currently in the device table. On the next I/O, regardless of the type of change in device characteristics, CONSOLE will not attempt the I/O. The return code of 2 is returned to the application indicating to query, if necessary, and then reissue the I/O.

Return Codes

Upon completion of the CONSOLE WAIT function, register 15 contains one of the following return codes:

Code

Meaning

0

The WAIT completed successfully.

2

You should issue a CONSOLE QUERY for the device before any more I/O is requested. See Usage Notes for additional information.

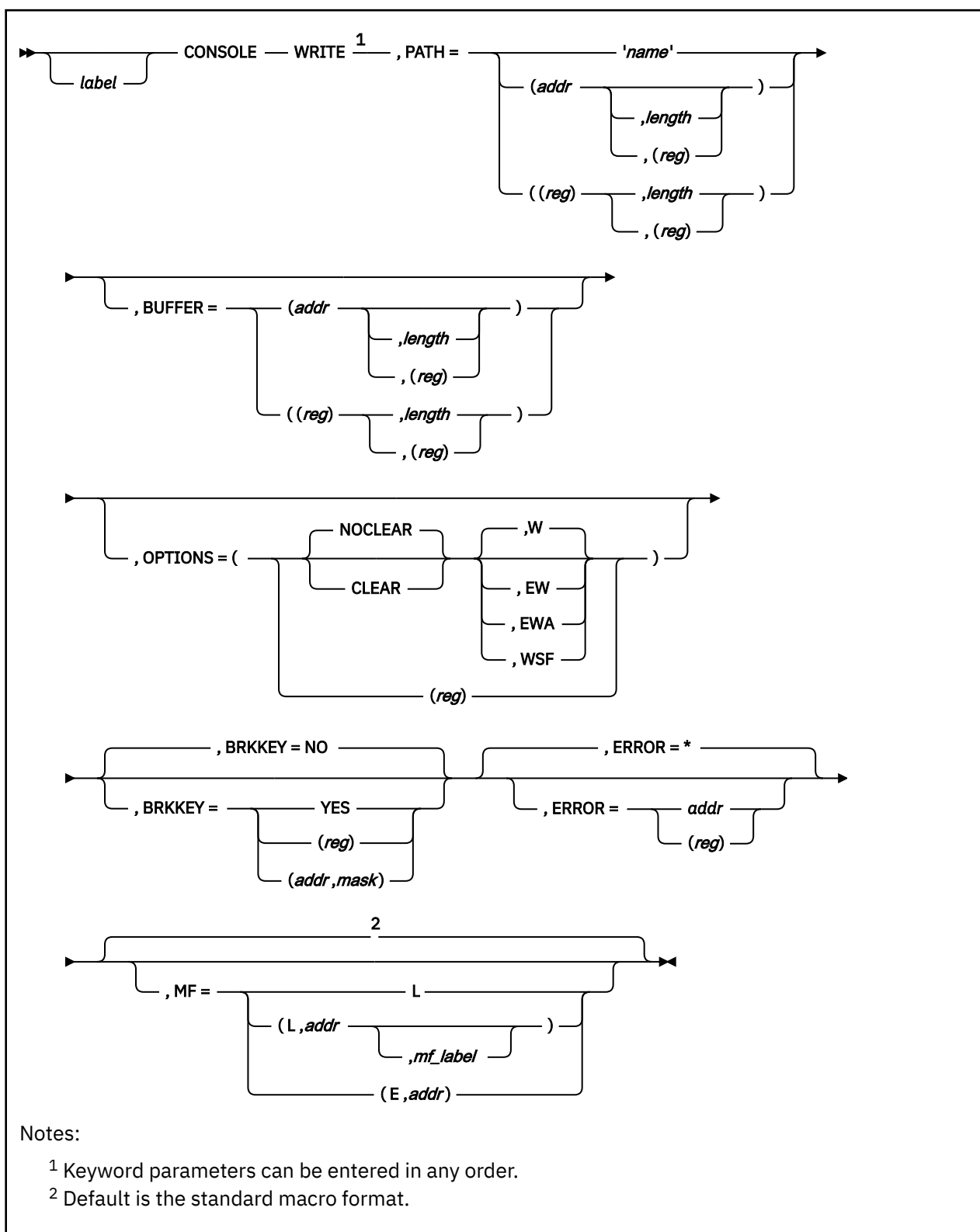
24

The parameter list is invalid; a path was not specified.

28

Path not found.

CONSOLE WRITE



Purpose

Use `CONSOLE WRITE` to write a 3270 data stream.

Parameters

Required Parameters:

WRITE

writes buffers that have 3270 data streams built by the application.

PATH=

specifies the path name. Acceptable values are:

'name'

specifies the path name as a 1- to 16-character literal string enclosed in quotation marks. Anything greater than this will be truncated when the CONSOLE request is processed.

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the path name and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

BUFFER=

specifies the area that contains the 3270 data stream (control characters and data) to be written to the device. Acceptable values are:

(addr,length)

specifies the buffer address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the buffer address and the length as an absolute expression. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

((reg), (reg))

specifies a register that contains the address of the buffer and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

OPTIONS=

specifies optional processing for this buffer. Options may be specified in any order. For example, (NOCLEAR,W) or (W,NOCLEAR) are both valid. Acceptable values are:

NOCLEAR

specifies that the physical screen is not cleared by the CONSOLE macro. The operating system may require you to clear the screen manually before the buffer is written. If you do not specify CLEAR or NOCLEAR, NOCLEAR is assumed.

CLEAR

specifies that the physical screen is cleared before the buffer (if there is one) is written. You can specify this option without the BUFFER parameter to simply clear the screen. If you specify both the BUFFER parameter and the CLEAR option, you should also specify EW, EWA, or WSF. If OPTIONS=clear,W is specified, only a clear is done; the W is ignored because the application must return to full screen mode with EW or EWA before a W is done. CLEAR is a line mode operation that takes the application out of full-screen mode. To return to fullscreen, the application must use EW or EWA.

W

specifies that the buffer is written with an ordinary Write command, overlaying the current contents of the display screen. If W, EW, EWA, or WSF are not specified, W is assumed.

EW

specifies that the buffer is written with the Erase/Write option. This option reformats the screen by causing a complete erasure of the screen before the write operation is started.

EWA

specifies that the buffer is written with the Erase/Write Alternate option to establish the alternate screen mode for the device.

WSF

specifies that the buffer is written with the Write Structured Field option to provide control information to the device.

(reg)

specifies a register 2-12, whose low-order byte contains the option or options to be used. The hexadecimal value of the byte must be set to one of the following:

```
X'00' = NOCLEAR,W
X'01' = NOCLEAR,EW
X'02' = NOCLEAR,EWA
X'04' = NOCLEAR,WSF
X'80' = CLEAR
X'81' = CLEAR,EW
X'82' = CLEAR,EWA
X'84' = CLEAR,WSF
```

BRKKEY=

specifies whether or not the break key interrupt is reflected to the virtual machine application, thus allowing the application full control of the keyboard. The BRKKEY parameter is only valid for the virtual console for EW and EWA operations. If specified for dedicated 3270 devices or options other than EW or EWA, the BRKKEY parameter will be ignored.

NO

specifies that if the break key is pressed, CP posts an attention interrupt to the virtual machine. If the application responds with a READ, or the break key is pressed a second time, the virtual machine is put in line mode and a CP READ is displayed on the screen's status area.

YES

specifies that if the break key is pressed, CP posts an attention interrupt to the virtual machine. If the application responds with a READ, the break key is passed to the application. If the application does not respond with a READ and the break key is pressed a second (or more) time, CP posts another attention interrupt to the virtual machine. In both cases, the passing of the break key interrupt to the virtual machine overrides the CP TERMINAL BRKKEY setting.

(reg)

instructs the macro to check the value of the specified register and, if it is 0, sets BRKKEY to NO. If the register contains a nonzero value, the macro sets BRKKEY to YES.

(addr,mask)

defines a single bit in storage that sets the value of the BRKKEY parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit

is 0, then BRKKEY is set to NO. If the bit is 1, then BRKKEY is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the BRKKEY parameter as

```
BRKKEY=(APPFLAG,X'80')
```

To set the value of the BRKKEY parameter at assembly time, specify BRKKEY=YES or BRKKEY=NO. To set the value at execution time, specify BRKKEY=(*reg*) or BRKKEY=(*addr,mask*)

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

- ***
passes control to the next sequential instruction. This is the default value.
- addr***
passes control to the specified address.
- (*reg*)**
passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

- L**
specifies the list format.
- (L,*addr,mf_label*)**
specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.
- (E,*addr*)**
specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If the console facility receives X'8E' in the channel status word (CSW) or subchannel status word (SCSW) from a W operation, then Console returns to your application with return code 32 to indicate that the screen must be reformatted (returned to full-screen mode) by EW/EWA or an appropriate WSF.
2. If the console facility receives X'8E' in the channel status word or subchannel status word on an EW/EWA operation, the application is in line mode and there is line mode data to be read (CP more . . . status). The console facility issues a line mode read (3215 SIO/SSCH) to clear the status and then reissues the original EW/EWA operation. The information on the more . . . screen is placed on the CMS input queue to be handled by CMS later.
3. Depending on whether CP console communications services or the logical device support facility is being used instead of CP native support, some I/O errors may not be reflected to the application issuing a DIAGNOSE code X'58', SIO instruction, or CONSOLE macro. In this case, the CMS console facility may see a channel end/device end from the I/O and therefore return a return code of 0. If an application that uses the console facility does not get expected results, the data stream or CCW (for EXCP) should be checked for user errors.
4. The BUFFER parameter must contain the address of the 3270 data stream to be written. The console facility does not scan this buffer or verify its contents. Therefore, for WSF, it is the application's responsibility to determine when to reformat the screen with EW/EWA operations when more than one application is writing to the screen. However, a return code of 32 is still returned if the screen is in line mode, such as when CP breaks in and writes to the screen. The field CQYPLIO can be checked with a CONSOLE QUERY path to determine if your application was the last path to write to the screen.

5. The CQYDLIN bit defined in the CQYSECT macro can aid in the determination of knowing whether or not to reformat the screen. CQYDLIN represents the virtual console's state at the last I/O interrupt (1=linemode, 0=fullscreen mode). Along with checking this bit prior to issuing a CONSOLE WRITE, your application should also specifically check for return code 32. This is because CQYDLIN does not reflect the status of the screen properly when the I/O to the virtual console is initiated by CP.
6. All applications whose paths are using the same device are notified of device changes through return code 2 the next time I/O is done. The device characteristics may have changed because the device was disconnected and then reconnected, or another application attempted an OPEN and the device characteristics do not match what is currently in the device table. On the next I/O, regardless of the type of change in device characteristics, CONSOLE will not attempt the I/O. The return code of 2 is returned to the application indicating to query, if necessary, and then reissue the I/O.
7. When BRKKEY=YES is specified for the virtual console, the break key interrupt will be reflected to the virtual machine. This replaces the normal break key function of returning the virtual machine to CP mode, and allows a virtual machine application to have full control of the keyboard. Normal break key function is restored when full screen mode is reset with a subsequent Erase/Write or Erase/Write Alternate.
8. When BRKKEY=YES is specified for the virtual console, CP's passing of the break key interrupt to the virtual machine overrides the BRKKEY setting as defined by the CP TERMINAL BRKKEY command. For more information about the CP TERMINAL BRKKEY command, and the possible break key settings, see the *z/VM: CP Commands and Utilities Reference*.

Examples

The following are some examples of a CONSOLE WRITE invocation using the BRKKEY parameter:

```

        LA      R2,EW      Erase/Write variable
        CONSOLE WRITE,PATH='TEST',          X
              BUFFER=(BUF,BUFL),            X
              OPTIONS=((R2)),                X
              BRKKEY=YES
        LA      R2,CEWA    Clear,Erase/Write Alternate
        CONSOLE WRITE,PATH='TEST',          X
              BUFFER=(BUF,BUFL),            X
              OPTIONS=((R2)),BRKKEY=NO
        CONSOLE WRITE,PATH='TEST',          X
              BUFFER=(BUF,BUFL),            X
              OPTIONS=(EWA),                 X
              BRKKEY=(APPFLAG,X'80')
        LA      R2,0       Do not reflect break key interrupt
        CONSOLE WRITE,PATH='TEST',          X
              BUFFER=(BUF,BUFL),            X
              OPTIONS=(EW),                  X
              BRKKEY=(R2)
        .
        .

R2      EQU X'02'
EW      EQU X'01'
CEWA    EQU X'82'
APPFLAG DC X'FF'
BUF      DC X'C21140401D60'
        DC C'HELLO'
BUFL    DC *-BUF

```

Return Codes

Upon completion of the CONSOLE WRITE function, register 15 contains one of the following return codes:

Code

Meaning

0

I/O successful. See Usage Notes [“2” on page 135](#) and [“3” on page 135](#) for additional information.

- 1**
A path has been opened to the virtual device, but no real device is currently connected to that virtual device.
- 2**
You should issue a CONSOLE QUERY for the device before any more I/O is requested. See Usage Notes for additional information.
- 24**
The parameter list is invalid; the function name is unknown, a required parameter is missing, or conflicting or invalid options were specified.
- 28**
Path not found. This return code occurs if the path was never opened, or if a device receives an I/O error because it was detached after the path was opened. The console facility closes all paths associated with the device, and indicates that the path no longer exists.
- 32**
A full-screen write was requested, but another application wrote to the screen. An Erase/Write must be issued to reformat the screen and return ownership to the current application. See Usage Notes [“1” on page 135](#) and [“2” on page 135](#) for more information.
- 100**
An I/O error has occurred. You can obtain the CSW/SCSW status by issuing a CONSOLE QUERY and specifying a buffer that contains the information.
- 104**
Insufficient storage space is available for an IDAL (Indirect Address List) on a CONSOLE WRITE request.

CQYSECT



Purpose

Use the CQYSECT macro to generate a DSECT for the CQYSECT control block. CQYSECT maps the device-type and path-type information returned by the CONSOLE macroinstruction.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the CQYSECT macro expansion is labeled CQYSECT.

Usage Notes

1. For more information regarding the DIAGNOSE codes X'24' and X'8C', see [z/VM: CP Programming Services](#).
2. The CQYSECT macroinstruction expands the Control Block Format as follows:

CQYSECT	DSECT	QUERY RETURN INFORMATION
CQYHEAD	DS 0D	REPLY BUFFER HEADER
CQYHLEN	DS F	LENGTH OF PATH SECTION
CQYHLEN	DS F	LENGTH OF DEVICE SECTION
CQYHEADL	EQU *-CQYHEAD	REPLY BUFFER HEADER LENGTH
*** Device Section ***		
CQYDEV	DS 0D	DEVICE DATA SECTION
CQYDUSCT	DS F	NO. PATHS OPENED TO THIS DEVICE
CQYDNUMB	DS F	VIRTUAL DEVICE NUMBER
CQYDVIRT	DS 0XL4	DIAGNOSE X'24' INFO
CQYDVCLS	DS X	VIRTUAL DEVICE INFO
CQYDVTYP	DS X	VIRTUAL DEV TYPE CLASS
CQYDVSTT	DS X	VIRTUAL DEVICE TYPE
CQYDVFLG	DS X	VIRTUAL DEVICE STATUS
CQYDREAL	DS 0XL4	VIRTUAL DEVICE FLAGS
CQYDRCLS	DS X	REAL DEVICE INFO
CQYDRTP	DS X	REAL DEVICE TYPE CLASS
CQYDRMDL	DS X	REAL DEVICE TYPE
CQYDRFTR	DS X	REAL MODEL NUMBER
CQYDVCNS	DS 0XL2	REAL FEATURE CODE
CQYDLLEN	DS X	MORE DIAG X'24' INFO
CQYDTMCD	DS X	VIRT. CONSOLE LINE LENGTH
CQYDATTR	DS X	VIRT. CONSOLE TERMINAL CODE
CQYDARMT	EQU X'04'	DEVICE ATTRIBUTE FLAGS
CQYDADSP	EQU X'02'	DEV. IS A REMOTE 3270
CQYDAVCN	EQU X'01'	DEV. IS A DISPLAY
CQYDSTAT	DS X	DEV. IS THE VIRTUAL CONSOLE
CQYDATTN	EQU X'80'	DEVICE STATUS FLAG 1
CQYDDISC	EQU X'40'	ATTENTION PENDING
CQYDLIN	EQU X'20'	DEVICE IS DISCONNECTED
CQYDQOR	EQU *	VIRTUAL CONSOLE STATE
CQYDQORFL	DS X	AT LAST I/O INTERRUPT
CQYDQORREC	EQU X'80'	(1=LINEMODE, 0=FSCR)
CQYDQOR	EQU *	FIRST 6 BYTES DIAG X'8C' INFO
CQYDQORFL	DS X	FLAGS
CQYDQORREC	EQU X'80'	EXTENDED COLOR SUPPORTED

```

CQYDQREH EQU X'40'      EXTENDED HIGHLIGHT SUPPORTED
CQYDQRPS EQU X'20'      PSS SUPPORTED
CQYDQREF EQU X'02'      3270 EMULATION FEATURE
CQYDQR14 EQU X'01'      14-BIT ADDRESSING SUPPORTED
CQYDQRPN DS X           NUMBER OF PARTITIONS
CQYDQRCL DS H           NUMBER OF COLUMNS
CQYDQRRW DS H           NUMBER OF ROWS
*
CQYDQYCD DS X           DEVICE QUERY CODE
          DS X           RESERVED
*
CQYD8CL DS F            LENGTH OF REMAINING DIAG X'8C' INFO
CQYD8CP DS A            PTR TO WSF INFO AFTER THE 1ST 6
*                          BYTES OF DIAG 8C INFO
          DS F            RESERVED
CQYDEVL EQU *-CQYDEV     LENGTH OF DEVICE SECTION
CQYDHL EQU (CQYHEADL+CQYDEVL) LENGTH OF DEV + HDR SECTIONS
*
*** Path Section ***
*
CQYPATH DS 0D           PATH DATA SECTION
CQYPEXIT DS A           USER EXIT ADDRESS
CQYPXWRD DS F           USER WORD
CQYPFLG DS X           PATH FLAG
CQYPLIO EQU X'80'       PATH DID LAST I/O
CQYPNRST EQU X'10'       NO DEVICE RESET REQUESTED
          DS XL3         RESERVED
          DS F           RESERVED
CQYPSCSW DS 0XL12       CHANNEL STATUS WORD
CQYPSLCC DS X           LOGOUT PENDING/COND. CODES
CQYPKSL EQU X'F8'       KEY/SUSPEND/LOG BITS
CQYPLOG EQU X'04'       LOGOUT PENDING
CQYPCC EQU X'03'        DEFERRED CONDITION CODE
CQYPCCTL DS XL3         ORB AND SUBCHANNEL CONTROL BITS
CQYPCCW DS F           LAST CCW EXECUTED
CQYPUST DS X           UNIT STATUS BYTE
CQYPATTN EQU X'80'       ATTENTION
CQYPSTMD EQU X'40'       STATUS MODIFIER
CQYPCUE EQU X'20'       CONTROL UNIT END
CQYPBUSY EQU X'10'       BUSY
CQYPCHEM EQU X'08'       CHANNEL END
CQYPDVEN EQU X'04'       DEVICE END
CQYPUCK EQU X'02'       UNIT CHECK
CQYPUSEX EQU X'01'       UNIT EXCEPTION
CQYPCST DS X           CHANNEL STATUS BYTE
CQYPPCI EQU X'80'       PROGRAM-CONTROLLED
*                          INTERRUPTION
CQYPICL EQU X'40'       INCORRECT LENGTH
CQYPPGCK EQU X'20'       PROGRAM CHECK
CQYPPRCK EQU X'10'       PROTECTION CHECK
CQYPCDCK EQU X'08'       CHANNEL DATA CHECK
CQYPCCK EQU X'04'       CHANNEL CONTROL CHECK
CQYPICCK EQU X'02'       INTERFACE CONTROL CHECK
CQYPCCHK EQU X'01'       CHAINING CHECK
CQYPRCT DS H           RESIDUAL COUNT
*
CQYPSCNT DS F           SENSE COUNT
CQYPSDTA DS XL32        SENSE DATA
CQYPATHL EQU *-CQYPATH   PATH SECTION LENGTH
CQYSIZE EQU (CQYHEADL+CQYDEVL+CQYPATHL) TOTAL
*                          CQYSECT LENGTH
CQYDBSZ EQU ((CQYSIZE+7)/8) SIZE OF ALL SECTIONS
*                          IN DBWRDS

```

CSFCB



Purpose

Use the CSFCB macro to map the data referenced by the fourth word in the extended plist for the CMS subcommand interface when inhibiting implicit recursion of execs.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the CSFCB macro expansion is labeled CSFCB.

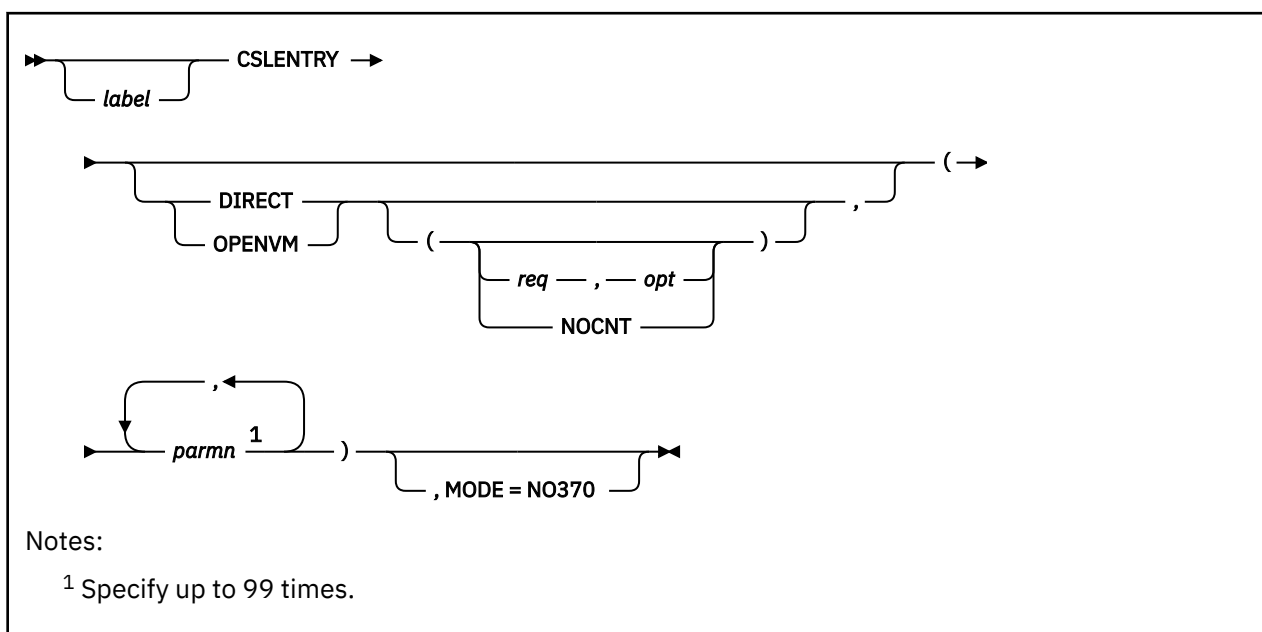
Usage Notes

- 1. The CSFCB macroinstruction expands as follows:

CSFCB	CSFCB		
CSFNMPTR	DSECT		
CSFFLAGS	DS	A	Pointer to file name of current exec
CSFFNBL	DS	F	Reserved for IBM use;
	EQU	*-CSFCB	Length (in bytes)

- 2. Only the CSFNMPTR field is intended as a programming interface.
- 3. An alternate format exec processor uses the name pointed to by the CSFNMPTR field to inhibit implicit recursion of the current exec. This is only effective when the name pointed to is the name of the current exec.
- 4. The CSFFLAGS field is reserved for use by IBM. It should be initialized to zero each time this interface is used.

CSLENTY



Purpose

Use the CSLENTY macro when writing a callable services library (CSL) routine to identify the module entry point, and generate the proper entry code.

You must specify the CSLENTY macro before any executable code or data in a CSL routine.

CSLENTY performs the following:

- Saves the calling program's registers
- Puts the number of parameters passed to the CSL routine in register 0
- Checks parameter list length and issues the proper return codes for improper length plists, if requested (DIRECT option only)
- Generates the USING statement for register 15.

Parameters

Required Parameters:

parm

specifies the name for a parameter expected by the CSL routine. You must define a name for each expected parameter.

Note: You should use the special parameter names RETURN and REASON in your program when referring to the return code and reason code parameters. If you specify the OPENVM option on CSLENTY, you should also use the special parameter name VALUE when referring to the return value parameter. Do not use these special names for any other parameters.

Optional Parameters:

label

is an optional assembler label for the statement.

DIRECT

specifies that the routine will be directly callable.

OPENVM

specifies that the routine will be directly callable and:

- The return code parameter is not required and may appear anywhere in the parameter list.
- The positions of the return code, reason code, and return value in the parameter list are marked using special data types in the CSL template file.

req,opt

supplies the number of required and optional entries in the parameter list. The sum of *req* and *opt* must be between 1 and 99 and *req* must be greater than 0.

NOCNT

indicates that no counting of parameters is to be done. No return codes will be set by the macro generated code. Register 0 will not be updated.

MODE=N0370

specifies that the macro should not create a System/370 code path.

Usage Notes

1. Upon completion of CSLENTY, register 0 contains the number of parameters passed (if requested) and register 1 points to the parameter list. If these two values are to be used later in the CSL routine code, they should be saved in different registers or in storage. Otherwise, various functions, including CSLGETP, may overwrite the contents of registers 0 and 1.
2. The number of parameters passed is treated differently depending on what information you supply to CSLENTY.
 - a. If CSLENTY (*parmn*), CSLENTY DIRECT, (*parmn*), or CSLENTY OPENVM, (*parmn*) is specified, the number of parameters passed is placed in register 0. No return codes for an incorrect parameter list will be returned. If DIRECT is specified, 100 is placed in register 0 when the parameter list size exceeds 99.
 - b. If CSLENTY DIRECT(*req,opt*) is specified, the number of parameters passed is placed in register 0. If *req* is greater than the number of parameters passed, a return code of -11 is returned to the caller. If the sum of *req* and *opt* is less than the number of parameters passed, a return code of -10 is returned to the caller.
 - c. If CSLENTY OPENVM(*req,opt*) is specified, the number of parameters passed is placed in register 0.

How CSLENTY handles a parameter list size error depends on whether a return code or return value is defined in the parameter list:

- If *req* is greater than the number of parameters passed, and a return value or return code or both exists in the parameter list, then a return value of -1 and a return code of -11 are returned to the caller.
 - If the sum of *req* and *opt* is less than the number of parameters passed, and a return value or return code or both exists in the parameter list, then a return value of -1 and a return code of -10 are returned to the caller.
 - In either of these error situations, if no return value or return code is defined in the parameter list, then CSLENTY initiates an ABEND.
 - d. If CSLENTY DIRECT(NOCNT) or CSLENTY OPENVM(NOCNT) is specified, then the number of parameters is ignored. No return codes are returned for an incorrect parameter list size. Register 0 is not updated.
3. You can use the parameter names specified on this macro with the CSLGETP macro to get information about the parameters passed to your CSL routine. For more information on using this macro, see the [z/VM: CMS Application Development Guide for Assembler](#).
 4. Specifying the DIRECT or OPENVM operand does not make a CSL routine a direct call routine. You must specify a path on the ROUTINE line or lines which use this routine's text file. The ROUTINE lines are

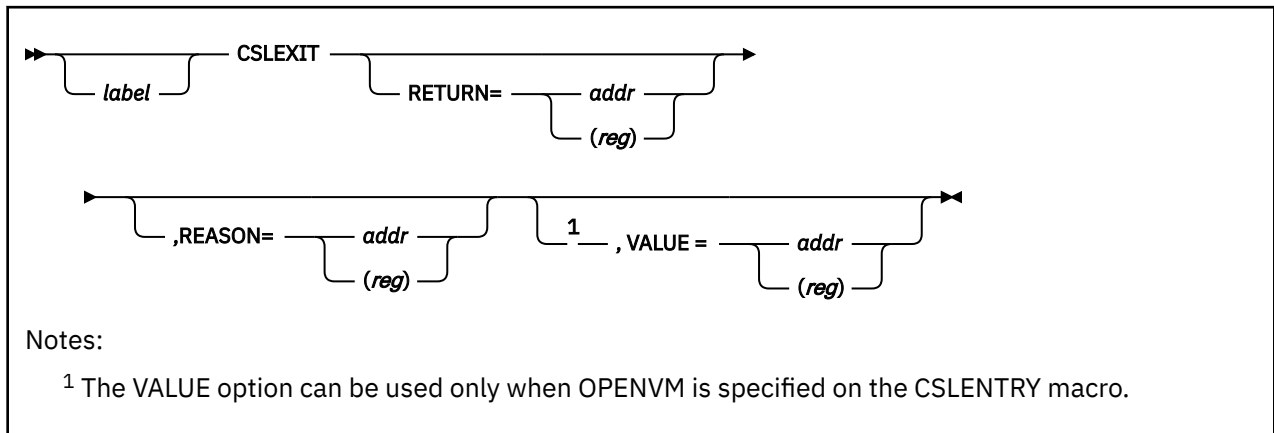
found in the CSLCNTRL file used by CSLGEN. You must also specify DIRECT or OPENVM on the first noncomment line of the template file.

Return Codes

Code	Meaning
------	---------

-10	The sum of <i>req</i> and <i>opt</i> is less than the number of parameters passed.
-11	<i>req</i> is greater than the number of parameters passed.

CSLEXIT



Purpose

Use the CSLEXIT macro when writing a callable services library (CSL) routine to generate the proper exit code.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

RETURN=

specifies a 4-byte numeric return code for the CSL routine to pass back to the calling application. When control is returned to the calling program, the return code value will be in register 15.

addr

is the label on a 4-byte field containing the return code that will be put in register 15.

(reg)

specifies a general register (other than register 1 or 13) which contains the 4-byte numeric return code that will be put in register 15.

In addition, if the name RETURN was used to identify a parameter on the CSLENTY macro, the calling program also gets the return code value in the first four bytes of the parameter that corresponds to RETURN.

REASON=

specifies a 4-byte numeric reason code for the CSL routine to pass back to the calling application. When control is returned to the calling program, the reason code value will be in register 0.

addr

is the label on a 4-byte field containing the reason code that will be put in register 0.

(reg)

specifies the register (other than register 1 or register 13) which contains the reason code that will be put in register 0.

In addition, if the name REASON was used to identify a parameter on the CSLENTY macro, the calling program also gets the reason code value in the first four bytes of the parameter that corresponds to REASON.

VALUE=

specifies a 4-byte numeric return value for the CSL routine to pass back to the calling application. When control is returned to the calling program, the return value is stored in the first four bytes of the parameter that corresponds to VALUE.

addr

is the label on a 4-byte field containing the return value that will be put in the parameter list.

(reg)

specifies the register (other than register 1 or register 13) which contains the return value.

This option can be used only in routines that specify the OPENVM option on the CSLENTY macro.

Usage Notes

1. Before invoking CSLEXIT, a CSL routine must ensure that register 13 contains the value it did at completion of CSLENTY.



Parameters

Required Parameters:

label

When TYPE=AREA or DSECT is specified, the *label* parameter is required and must be a 1- to 4-character label. This label will be used to reference the fast path area. The following labels are generated for the calling program to use:

- *label*CSL1—is equated to the length of the fast path area generated
- *label*CSL2—is equated to the number of parameters
- *label*XCSA—is an area used to save the translation mode and access registers at the time of the CSLFPI TYPE=CALL transfer if the application is executing in access register mode (and the DMSSTATE macro has been called with ASCENV=ARM).

When TYPE=INIT, INITD, SET, or CALL, *label* is an optional assembler label for the statement.

TYPE=

specifies which form of CSLFPI you are using. Acceptable values are:

AREA

means that you are building a fast path area to contain information about a CSL routine and its parameters.

If this fast path area is to be included as part of a larger area mapped by a DSECT, TYPE=AREA should be used within that DSECT; otherwise, TYPE=AREA generates the fast path area in a CSECT.

DSECT

means that you are building a fast path area to contain information about a CSL routine and its parameters. The fast path area will be mapped by a DSECT whose name is specified by *label*.

Note: If you specify parameter information when building a fast path area for mapping (TYPE=DSECT, or TYPE=AREA call within a DSECT), you must still specify parameter information before invoking the CSL routine (using TYPE=INITD or SET).

INIT

means that you are initializing the fast path area that contains information about a CSL routine and its parameters.

INITD

means that you are initializing the fast path area that contains information about a CSL routine and its parameters. Use INITD (rather than INIT) if you previously mapped the fast path area inside a DSECT using CSLFPI with TYPE=AREA or DSECT.

SET

means that you are setting or changing CSL parameter values or lengths. You can use SET to give an initial value or to modify a value.

CALL

means that you are invoking the CSL routine.

Note: Before a CSL routine is invoked, you must specify a name and value for each parameter that the CSL routine expects. See the description of the optional PARMS parameter. You can modify parameter values when invoking CSLFPI with TYPE=CALL and with TYPE=SET, INIT, or INITD.

AREA=*arealab*

specifies the 1- to 4-character label for the fast path area that was specified on a previous CSLFPI with TYPE=AREA or DSECT.

Optional Parameters:

PARMS=

specifies information about CSL routine parameters.

Note:

1. PARMs is required when TYPE=AREA, DSECT, or SET is specified.
2. When using PARMs with TYPE=INIT, INITD, SET, or CALL, you must supply a value or length for at least one parameter.

Acceptable values are:

name

is a 1- to 4-character name that identifies a parameter.

valn

is a label used to reference the associated parameter's value.

(reg)

is a register containing the address of the associated parameter's value.

Note: If you do not specify a parameter value but want to specify a length, you must code the comma as a place holder. notations:

lenn

is the length of the associated parameter.

(reg)

is a register containing the length of the associated parameter.

Note: If the parameter is a character string, you must specify a length, unless the length is implied or the CSL routine defines the length for you.

SERVICE=

specifies the name of the CSL routine to be invoked. SERVICE must be specified when either building or initializing the fast path area. Acceptable values are:

routine

is the name of the CSL routine.

(reg)

is a register containing the address of the name of the CSL routine.

MODE=N0370

specifies that the macro should not create a System/370 code path.

Usage Notes

1. If your application is executing in an XC virtual machine in access register mode, CSL routines must be called such that transfer to the service results in primary space mode entry. You can ensure the correct entry condition by preceding your CSLFPI macro call with a

```
DMSSTATE SET,ASCENV=ARM
```

macro specification. You should also ensure that access registers 12 and 13 contain the value 0 to avoid the inadvertent modification of an address space or the occurrence of a program check. For more information, see the [z/VM: CMS Application Development Guide](#).

2. If you must reinitialize a fastpath area, you should also reinitialize the parameters. The integrity of the fastpath area is not guaranteed after the first reinitialization.
3. Directly callable CSL routines cannot access the parameter length information provided by the indirect CSL interface. Therefore, parameter lengths, other than for parameters specified in the routine template as FCHR 0, will be ignored. These lengths are still saved for routines using the indirect CSL interface.
4. Additional labels, *labelCSL4* - *labelCSLB*, may be generated. These labels are for CSLFPI use only.

Return Codes

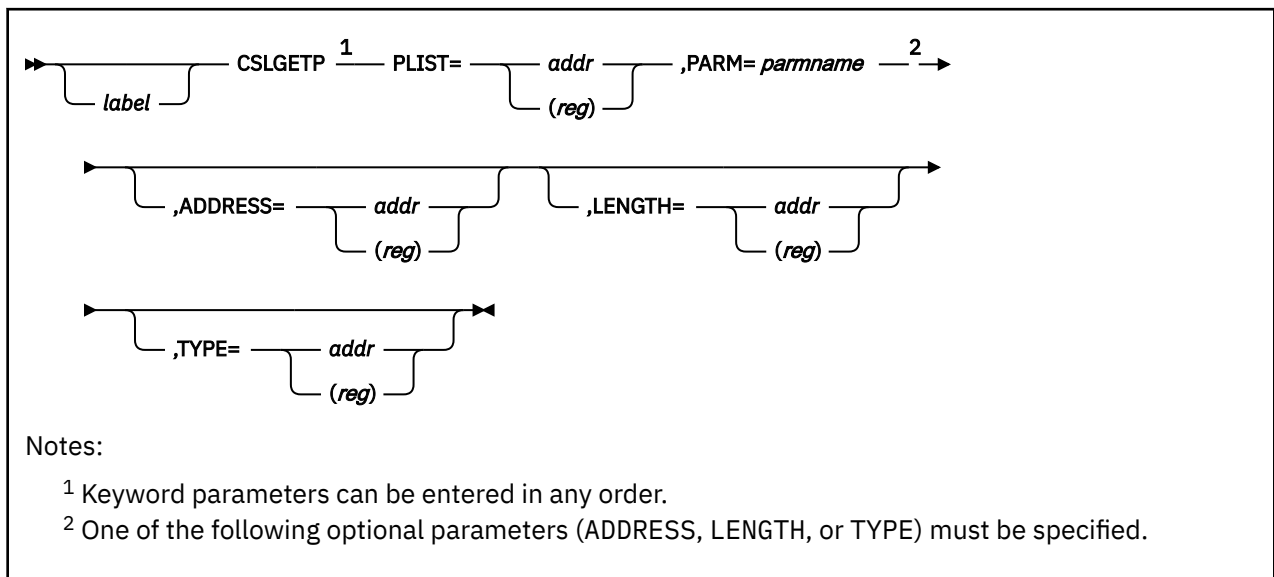
Code

Meaning

- 07**
Routine not loaded. (TYPE=INIT/INITD only)
- 08**
Routine has been dropped. (TYPE=CALL only)
- 09**
Insufficient virtual storage available. (TYPE=INIT/INITD only)
- 10**
Too many parameters specified. (TYPE=INIT/INITD only)
- 11**
Not enough parameters specified. (TYPE=INIT/INITD only)
- 12**
CSL does not exist on the release. (TYPE=CALL only)
- 13**
The CSLFPI fast path area provided cannot be used to call the service specified. (TYPE=INIT/INITD, TYPE=CALL)

The fast path area was created by a CSLFPI macro which cannot provide parameters in the standard plist format required by the currently loaded routine version. The routine was specified by the SERVICE operand. Recompiling all of the programs which use the CSLFPI workarea is required.

CSLGETP



Purpose

Use the CSLGETP macro in the callable services library (CSL) routine you are writing to get information about a parameter passed to the routine. CSLGETP uses the parameter names defined on the CSLENTY macro.

Parameters

Required Parameters:

PLIST=

specifies the address of the parameter list. (When CSLENTY completes, the address of the parameter list is contained in register 1, but do not assume this when using CSLGETP because the contents of register 1 are overwritten by CSLGETP.)

addr

is the name of a 4-byte field containing the address of the parameter list.

(*reg*)

is a register containing the address of the parameter list.

PARM=*parmname*

specifies the name of the parameter you want information about. The parameter name (*parmname*) must have been previously defined on the preceding CSLENTY macro call.

Optional Parameters:

label

is an optional assembler label for the statement.

Note: At least one of the following three optional parameters must be specified.

ADDRESS=

specifies that you want to get the address of *parmname*. Acceptable values are:

addr

is the name of a 4-byte field in which to store the address of the parameter.

(*reg*)

is a register where the address of the parameter will be stored. It may be any register other than register 1.

LENGTH=

specifies that you want to get the length of parameter. This cannot be used when DIRECT is specified on the CSLENTY macro.

addr

is the name of a 4-byte field in which to store the length of the parameter.

(*reg*)

is a register where the length of the parameter will be stored. It may be any register other than register 1.

TYPE=

specifies that you want to get the data type of *parmname*. This cannot be used when DIRECT is specified on the CSLENTY macro. The data type is returned as a 1-byte code, equated to one of the following labels:

Label**Data Type****CSLTSBIN**

Signed binary number

CSLTUBIN

Unsigned binary number

CSLTBIT

Bit string

CSLTLEN

Unsigned binary length parameter for a previous parameter; can also indicate the number of rows for a table

CSLTCHAR

Character string

CSLTFCHR

Fixed-length character string (compatibility)

You can use these labels to check the data type code that is returned. Acceptable values for TYPE are:

addr

is the name of a 1-byte field in which to store the data type code.

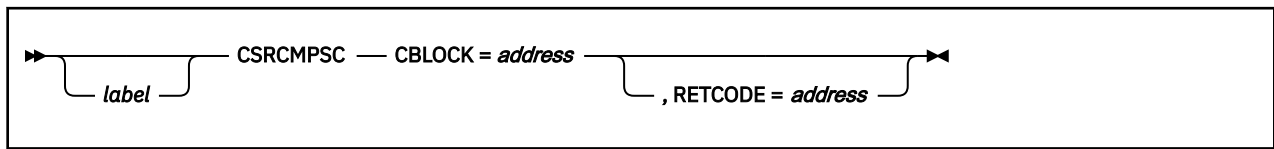
(*reg*)

is a register containing the address of a 1-byte field in which to store the data type code. It may be any register other than register 1.

Usage Notes

1. CSLGETP will return a value for the address, length, and type for optional parameters (defined on CSLENTY) that are not supplied by the calling program. Your program should use the value in register 0 at the completion of CSLENTY to determine how many optional parameters are being passed in.
2. CSLGETP overwrites the contents of register 1.
3. Length and type cannot be specified when DIRECT is coded on the CSLENTY macro.

CSRCMPSC



Purpose

CSRCMPSC is an interface to system Data Compression Services. It:

- Compresses data
- Expands previously compressed data

This interface uses the S/390® hardware compression instruction CMPSC. If your hardware does not support the CMPSC compression instruction, the system software simulation of the instruction will be used to perform the service.

Parameters

Required Parameters:

CBLOCK=

is the parameter list for compression services mapped by the CSRYCMPS macro. This parameter list is 36 bytes and contains:

- Bit fields for compression service control
- Addresses of the compression and expansion dictionaries
- Source and target area addresses and associated lengths
- Source and target ALET definitions, if Access Register (AR) mode is being used
- Address of a compression services work area.

Optional Parameters:

label

is an optional assembler label for the statement.

RETCODE=

Is the output return code from the compression service.

address

Is an RS-type address or a register specification. CBLOCK can be used with registers (2)-(12).

RETCODE can be used with registers (2)-(12).

Note: The register save area pointed to by register 13 must be 144 bytes.

Usage Notes

1. The compression and expansion dictionaries specified in the CBLOCK must both be defined on page boundaries and must be contiguous in storage.
2. Printing of the macro expansion is controlled by the ZPRINT global macro variable. Any value other than 'NO' will result in the macro expansion being printed. The default is 'YES'. The following examples illustrate how the ZPRINT variable should be set.

```
GBLC &ZPRINT
&ZPRINT SETC 'YES'
&ZPRINT SETC 'NO'
```

Examples

1. The following is an example of compression.

```

        LA    13,SAVEAREA          Get address of save area
        LA    2,MYCBLOCK          Get address of parm
        USING CMPSC,2
        XC    CMPSC(CMPSC_LEN),CMPSC  Clear block
        OI    CMPSC_FLAGS_BYTE2,CMPSC_SYMSIZE_5 Set size
                Symbol size is 5+8. Dictionary has
                2**(5+8) entries
        L      3,DICTADDR
        ST    3,CMPSC_DICTADDR      Set dictionary address
        L      3,COMPADDR
        ST    3,CMPSC_TARGETADDR    Set compression area
        L      3,COMPLEN
        ST    3,CMPSC_TARGETLEN     Set compression length
        L      3,EXPADDR
        ST    3,CMPSC_SOURCEADDR    Set expansion area
        L      3,EXPLEN
        ST    3,CMPSC_SOURCELEN     Set expansion length
        LA    3,WORKAREA
        ST    3,CMPSC_WORKAREAADDR  Set workarea address
        CSRCMPSC CBLOCK=CMPS
        DROP 2

        .
        .
        DS    0F                  Align parameter on word boundary
        MYCBLOCK DS (CMPSC_LEN)CL1  CBLOCK parameter
        COMPADDR DS A              Output "To" (compression) area
        COMPLEN  DS F              Length of "To" area
        EXPADDR DS A              Input "From" (expansion) area
        EXPLEN  DS F              Length of "From" area
        DICTADDR DS A              Address of compression dictionary
        DS      0D                Doubleword align workarea
        WORKAREA DS CL192         Work area
        SAVEAREA DS CL144         Register save area
        CSRYCMPS ,

```

The expansion dictionary must immediately follow the compression dictionary and both must be aligned on page boundaries.

2. The following is an example of expansion.

```

        LA    13,SAVEAREA          Get address of save area
        LA    2,MYCBLOCK          Get address of parm
        USING CMPSC,2
        XC    CMPSC(CMPSC_LEN),CMPSC  Clear block
        OI    CMPSC_FLAGS_BYTE2,CMPSC_SYMSIZE_5 Set size
                Symbol size is 5+8. Dictionary has
                2**(5+8) entries
        OI    CMPSC_FLAGS_BYTE2,CMPSC_EXPAND Do expansion
        L      3,EDICTADDR
        ST    3,CMPSC_DICTADDR      Set dictionary address
        L      3,EXPADDR
        ST    3,CMPSC_TARGETADDR    Set expansion area
        L      3,EXPLEN
        ST    3,CMPSC_TARGETLEN     Set expansion length
        L      3,COMPADDR
        ST    3,CMPSC_SOURCEADDR    Set compression area
        L      3,COMPLEN
        ST    3,CMPSC_SOURCELEN     Set compression length
        LA    3,WORKAREA
        ST    3,CMPSC_WORKAREAADDR  Set workarea address
        CSRCMPSC CBLOCK=CMPS
        DROP 2

        .
        .
        DS    0F                  Align parameter on word boundary
        MYCBLOCK DS (CMPSC_LEN)CL1  CBLOCK Parameter
        EXPADDR DS A              Output "To" (expansion) area
        EXPLEN  DS F              Length of "To" area
        COMPADDR DS A              Input "From" (compression) area
        COMPLEN  DS F              Length of "From" area
        EDICTADDR DS A              Address of expansion dictionary
        DS      0D                Doubleword align workarea
        WORKAREA DS CL192         Work area
        SAVEAREA DS CL144         Register save area
        CSRYCMPS ,

```

Note: The expansion dictionary must be aligned on a page boundary.

Return Codes

When this macro completes processing, it passes a return code in register 15 to the caller.

Code

Meaning

0

No errors detected.

4

Target operand exhausted before source.

16

An operand is missing.

20

Value in CMPSC_SYMSIZE is not supported. Must be 1-5.

24

No work to do. The compression area length (the target for compression, the source for expansion) is not large enough to hold even one compression symbol.

28

Compression dictionary processing exceeded the limit of 260 for the length of a compressed symbol.

32

A dictionary entry exceeded the limit of 260 total children.

36

A dictionary entry exceeded the limit of a child count of 6.

40

A dictionary entry exceeded the limit of 4 extension characters when there were 0 or 1 children.

44

A sibling descriptor dictionary entry has a count of 0.

48

Extension of a symbol used more than 127 dictionary entries.

ABEND Code

Meaning

0C4

The user may get this completion code if a user-provided data area is not accessible.

0C6

The user may get this completion code if the symbol size value within the CBLOCK area is not 1-5.

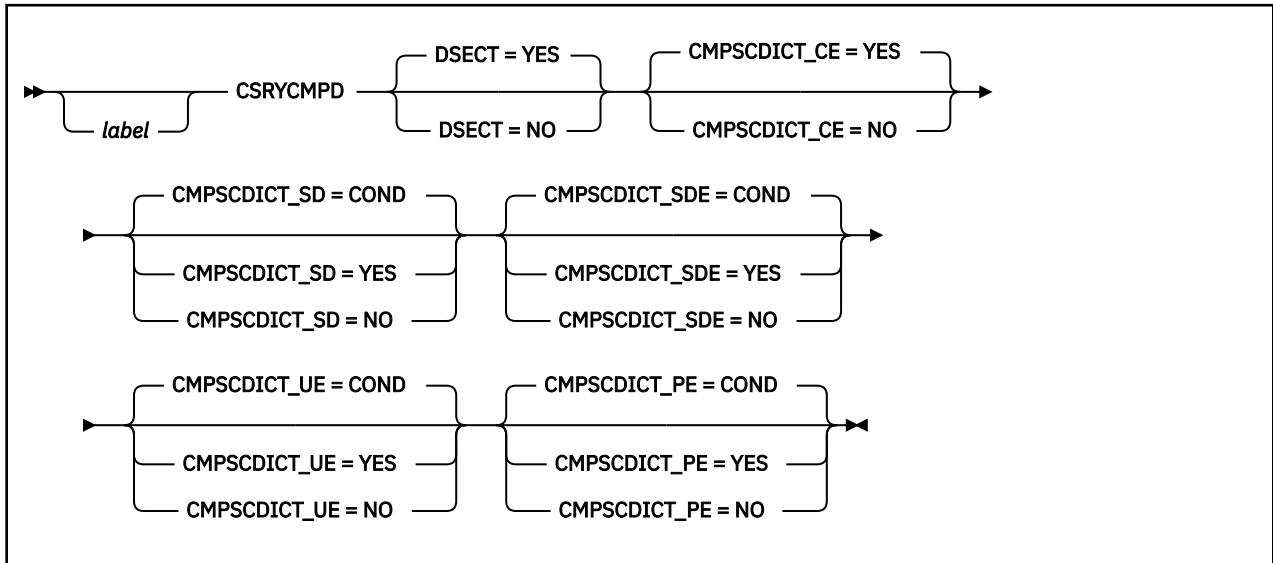
0C7

The user may get this completion code in the following circumstances when the dictionary is built incorrectly:

- If the length of a string to be represented by a single compression symbol, encountered during a compression operation, exceeds 260 characters.
- If a dictionary entry has more than 260 total children.
- If the "child count" in a dictionary entry indicates more than 6 children.
- If the number of extension characters for a dictionary entry with 0 or 1 children exceeds 4.
- If a sibling descriptor dictionary entry has a sibling count of 0.
- If expansion of a compression symbol uses more than 260 characters.
- If expansion of a compression symbol uses more than 127 dictionary entries.

In all these cases, the user must fix the dictionary.

CSRYCMPD



Purpose

Use the CSRYCMPD macro to map the compression and expansion dictionaries.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

DSECT=

indicates that you are about to specify whether the template produced will be a DSECT (dummy control section).

YES

indicates that the template will be created as a DSECT. If you omit the DSECT parameter altogether, then the template is produced as a DSECT. This is the default.

NO

indicates that the DSECT statement should not be generated.

CMPSCDICT_CE=

indicates a request for mapping of the child entries in the compression dictionary.

YES

indicates that the mapping will be created for the child entries in the compression dictionary. This is the default.

NO

indicates that the mapping will not be created for the child entries in the compression dictionary.

CMPSCDICT_SD=

indicates a request for mapping of the sibling descriptors in the compression dictionary.

COND

indicates that the mapping is included if DSECT=YES, but not if DSECT=NO. This is the default.

YES

indicates that the mapping will be created for the sibling descriptors in the compression dictionary.

NO

indicates that the mapping will not be created for the sibling descriptors in the compression dictionary.

CMPSCDICT_SDE=

indicates a request for mapping of the sibling descriptor extensions in the compression dictionary.

COND

indicates that the mapping is included if DSECT=YES, but not if DSECT=NO. This is the default.

YES

indicates that the mapping will be created for the sibling descriptor extensions in the compression dictionary. Note that they are physically located within the expansion dictionary.

NO

indicates that the mapping will not be created for the sibling descriptor extensions in the compression dictionary.

CMPSCDICT_UE=

indicates a request for mapping of the unpreceded entries in the expansion dictionary.

COND

indicates that the mapping is included if DSECT=YES, but not if DSECT=NO. This is the default.

YES

indicates that the mapping will be created for the unpreceded entries in the expansion dictionary.

NO

indicates that the mapping will not be created for the unpreceded entries in the expansion dictionary.

CMPSCDICT_PE=

indicates a request for mapping of the preceded entries in the expansion dictionary.

COND

indicates that the mapping is included if DSECT=YES, but not if DSECT=NO. This is the default.

YES

indicates that the mapping will be created for the preceded entries in the expansion dictionary.

NO

indicates that the mapping will not be created for the preceded entries in the expansion dictionary.

Usage Notes

1. The compression and expansion dictionaries must both begin on page boundaries. When compressing, the expansion dictionary must immediately follow (be contiguous to) the compression dictionary.
2. Each dictionary consists of 512, 1024, 2048, 4096, or 8192 8-byte entries. These are indicated by a value of 1, 2, 3, 4, or 5 in the CMPSC_SYMSIZE field which is part of the parameter information passed to the CSRCMPSC service.

The compression dictionary consists of child entries (DSECT CMPSCDICT_CE), sibling descriptors (DSECT CMPSCDICT_SD), sibling descriptor extensions (DSECT CMPSCDICT_SDE). Note that the latter are physically resident within the expansion dictionary.

The expansion dictionary consists of unpreceded entries (DSECT CMPSCDICT_UE) and preceded entries (DSECT CMPSCDICT_PE).

3. The following fields are generated by the CSRYCMPD macro:

Table 14. CSRYCMPD Macro

Offsets		Type/Value	Len	Name (Dim)	Description
Hex	Dec				
(0)	0	BITSTRING	1	CMPSCDICT_CE_H1	First byte of header
		111.		CMPSCDICT_CE_CHILDCT	X'E0' Child character count
		...1 1111		CMPSCDICT_CE_EXCHILD	X'1F' Examine child bits
(1)	1	BITSTRING	2	CMPSCDICT_CE_H23	Second/third bytes of header
		111.		CMPSCDICT_CE_AECCT	X'E0' Additional extension count
		11..		CMPSCDICT_CE_EXSIB	X'C0' Examine sibling bits
		..1.		CMPSCDICT_CE_ADDEXTCHAR	X'20' If on, add ext character
(1)	1	BITSTRING	1	CMPSCDICT_CE_FIRSTCHILDINDEX_REPLACED (0)	
		BITSTRING		CMPSCDICT_CE_FIRSTCHILDINDEX	X'1FFF' This mask can be used to isolate the 13-bits of field CMPSCDICT_CE_H23 that represent the index of the first child
(3)	3	CHARACTER	5	CMPSCDICT_CE_CHILDCHAR	Child character entries
		..1.		CMPSCDICT_CE_CHILDCT_1	"B'00100000" Value of 1 for CMPSCDICT_CE_CHILDCT within field CMPSCDICT_CE_H1
		..1.		CMPSCDICT_CE_AECCT_1	"B'00100000" Value of 1 for CMPSCDICT_CE_AECCT within field CMPSCDICT_CE_H23
		X'8'		CMPSCDICT_CE_LEN	"*-CMPSCDICT_CE"

Table 15. CMPSCDICT_SD DSECT

Offsets	Type/Value	Len	Name (Dim)	Description
He x	Dec			
(0)	0	BITSTRING	2	CMPSCDICT_SD_HD Header
		1111		CMPSCDICT_SD_SIBCT X'F0' Sibling count
(0)	0	BITSTRING	1	CMPSCDICT_SD_EXSIB_REPLACED (0)
		BITSTRING		CMPSCDICT_SD_EXSIB X'0FFF' This represents a 12-bit subfield of CMPSCDICT_SD_HD. Each bit indicates to examine the corresponding sibling.
(2)	2	CHARACTER	6	CMPSCDICT_SD_CHILDCHAR Sibling character entries
		. . . 1		CMPSCDICT_SD_SIBCT_1 "B'00010000" Value of 1 for CMPSCDICT_SD_SIBCT within field CMPSCDICT_SD_HD
		X'8'		CMPSCDICT_SD_LEN "*-CMPSCDICT_SD"

Table 16. CMPSCDICT_SDE DSECT

Offsets	Type/Value	Len	Name (Dim)	Description
He x	Dec			
(0)	0	CHARACTER	8	CMPSCDICT_SDE_CHILDCHAR Sibling character entries
		X'8'		CMPSCDICT_SDE_LEN "*-CMPSCDICT_SDE"

Table 17. CMPSCDICT_UE DSECT

Offsets	Type/Value	Len	Name (Dim)	Description
He x	Dec			
(0)	0	BITSTRING	1	CMPSCDICT_UE_HD Header
		111.		CMPSCDICT_UE_PARTSYMLEN X'E0' Partial symbol length = 0

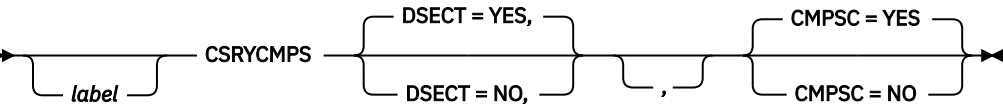
Table 17. CMPSCDICT_UE DSECT (continued)

Offsets	Type/Value	Len	Name (Dim)	Description
He x	Dec			
			CMPSCDICT_UE_COMPSYMLEN	
				X'07' Completed symbol length
(1)	1	CHARACTER	7	CMPSCDICT_UE_CHARS
				Extension characters
			CMPSCDICT_UE_COMPSYMLEN_1	
				"B'00000001" Value of 1 for CMPSCDICT_UE_COMPSYMLEN within field CMPSCDICT_UE_HD
			CMPSCDICT_UE_LEN	
				X'8'
				"*-CMPSCDICT_UE"

Table 18. CMPSCDICT_PE DSECT

Offsets	Type/Value	Len	Name (Dim)	Description
He x	Dec			
(0)	0	CHARACTER	2	CMPSCDICT_PE_HD
				Header
			CMPSCDICT_PE_PARTSYMLEN	
				X'E0' Partial symbol length →= 0
(0)	0	BITSTRING	1	CMPSCDICT_PE_PRECENTINDEX_REPLACED (0)
		BITSTRING		CMPSCDICT_PE_PRECENTINDEX
				X'1FFF' This mask can be used to isolate the 13- bits of field CMPSCDICT_PE_HD that represent the index of the preceding entry
(2)	2	CHARACTER	5	CMPSCDICT_PE_CHARS
				Extension characters
(7)	7	SIGNED	1	CMPSCDICT_PE_OFFSET
				Offset where first character in CMPSCDICT_PE_CHARS belongs
			CMPSCDICT_PE_PARTSYMLEN_1	
				"B'00100000" Value of 1 for CMPSCDICT_PE_PARTSYMLEN within field CMPSCDICT_PE_HD
			CMPSCDICT_PE_LEN	
				X'8'
				"*-CMPSCDICT_PE"

CSRYCMPS



Purpose

Use the CSRYCMPS macro to map the compression services parameter list which is required to call Data Compression Services.

The fields in the CBLOCK parameter list are filled in by the user of Data Compression Services. The address of the CBLOCK is passed to Data Compression Services through the CBLOCK parameter of the CSRCMPSC macro. The CSRYCMPS macro also provides equates used to map error codes returned by Data Compression Services.

Parameters

Optional Parameters:

label
is an optional assembler label for the statement.

DSECT=
indicates that you are about to specify whether the template produced will be a DSECT (dummy control section).

YES
indicates that the template will be created as a DSECT. If you omit the DSECT parameter altogether, then the template is produced as a DSECT. This is the default.

NO
indicates that the DSECT statement should not be generated.

CMPSC=
indicates that you are about to specify whether the template produced will be a DSECT (dummy control section).

YES
indicates that a mapping of CMPSC has been requested. This is the default.

NO
indicates that a mapping of CMPSC has not been requested.

Usage Notes

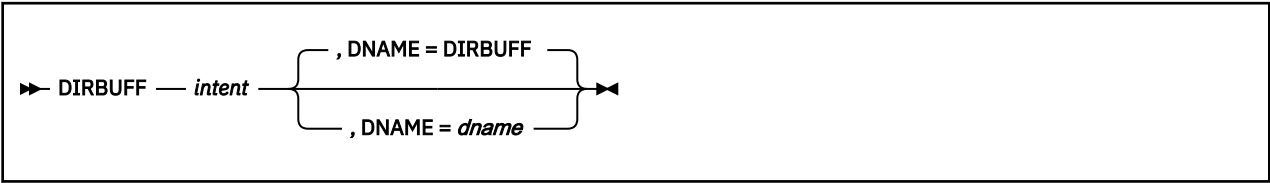
1. All the CMPSC_FLAGS bits should be zero, except for the CMPSC_SYMSIZE and CMPSC_EXPAND bits in the CMPSC_FLAG2 byte. If other bits are found on, unpredictable results may occur.
2. The following fields are generated by the CSRYCMPS macro:

Offsets	Type/Value	Len	Name (Dim)	Description
Hex				
(0) 0	STRUCTURE		CMPSC	, Compression parameter block
(0) 0	BITSTRING	4	CMPSC_FLAGS	

Offsets Hex	Dec	Type/Value	Len	Name (Dim)	Description
					Flag bits within which only the SymSize and Expand fields should be set. All other fields must be 0.
(0)	0	BITSTRING	1	CMPSC_FLAGS_BYTE0	Byte 0 of flags
(1)	1	BITSTRING	1	CMPSC_FLAGS_BYTE1	Byte 1 of flags
(2)	2	BITSTRING	1	CMPSC_FLAGS_BYTE2	Byte 2 of flags
		1111		CMPSC_SYMSIZE	
					X'F0' When 8 is added, indicates size in bits of a compressed entry. Must be 1-5. You can use the assembler CMPSC_SYMSIZE equate to define a value that you can use to clear the field prior to use. You can use the assembler equates CMPSC_SYMSIZE_n to set the field
	1		CMPSC_EXPAND	
					X'01' If on, do an expand operation. Otherwise compress
(3)	3	BITSTRING	1	CMPSC_FLAGS_BYTE3	Byte 3 of flags
(4)	4	ADDRESS	4	CMPSC_DICTADDR	Address of the dictionary for the compress/expand function on a page boundary. Low order 12 bits of the field are treated as 0s when forming the address. Low order 3 bits contain a bit number.
(4)	4	BITSTRING	3		
(7)	7	BITSTRING	1	CMPSC_DICTADDR_BYTE3	
	111		CMPSC_BITNUM	
					X'07' If compressing, place the first compression symbol at this bit in the leftmost byte. If expanding, expand beginning with the compression symbol that begins with this bit in the left-most byte. Normally, this bit should be set to 0 for the start of compression. For expansion, it should be set to the same value used for the start of compression. Upon completion of the operation, the value is set to the bit number of the bit following the last bit of compressed data.

Offsets Hex	Type/Value Dec	Len	Name (Dim)	Description
(8)	8	ADDRESS	4	CMPSC_TARGETADDR
				Address of area to which compression/expansion is to be done. Upon completion of the request, this address has been increased by the number of bytes processed.
(C)	12	SIGNED	4	CMPSC_TARGETLEN
				Length of area to which compression/expansion is to be done. Upon completion of the request, this length has been decreased by the number of bytes processed.
(10)	16	ADDRESS	4	CMPSC_SOURCEADDR
				Address of area from which compression/expansion is to be done. Upon completion of the request, this address has been increased by the number of bytes processed.
(14)	20	SIGNED	4	CMPSC_SOURCELEN
				Length of area from which compression/expansion is to be done. For expansion, the length should be the difference between the TargetLen at completion of compression and the TargetLen at start of compression, incremented by 1 if field CMPSC_BITNUM was non-zero upon completion of compression. Upon completion of the request, this length has been decreased by the number of bytes processed.
(18)	24	SIGNED	4	CMPSC_TARGETALET
				The ALET of the space in which the target area resides. Should be 0 for primary ASC mode callers.
(1C)	28	SIGNED	4	CMPSC_SOURCEALET
				The ALET of the space in which the source area resides. Also the ALET of the space in which the dictionary resides. Should be 0 for primary addressing mode callers.
(20)	32	ADDRESS	4	CMPSC_WORKAREAADDR
				Address of a 192-byte work area for use by the compression service. This area does not need to be provided if you have verified, by checking that bit CVTCMPSH is on, that the hardware CMPSC instruction is present. This work area should begin on a doubleword boundary.

DIRBUFF



Purpose

Use the DIRBUFF macroinstruction to map the records returned by a Get Directory request.

Parameters

Required Parameters:

intent

specifies the type of directory record to be mapped. Legal types are:

- FILE
- FILEEXT
- SEARCHAUTH
- SEARCHALL
- ALIAS
- AUTH
- LOCK
- DIR
- ALL.

If ALL is specified, all of the record mappings are generated. This is used when one program uses Get Directory for more than one type of record.

Optional Parameters:

DNAME=dname

specifies an optional DSECT label. If none is provided, the default label is DIRBUFF.

Usage Notes

1. Mapping of these buffers is also possible using the Get Directory (DMSGETDI) callable services library (CSL) routine and its companion routines for deblocking the various buffers. For more information on Get Directory and the output mapped by DIRBUFF, see the [z/VM: CMS Callable Services Reference](#).
2. The DIRBUFF mapping macro expands the Directory Record Types as follows:

&DNAME	DSECT		
DIRTYPE	DS	CL1	Type of record
DIRFSTYP	DS	CL1	Filesystemtype (used only for
*			FILEEXT, and LOCK
DIRRECL	DS	H	Length of record
*			
*		Constants	
*			
DIRTFIL	EQU	C'1'	DIRTYPE of FILE
DIRTSRHL	EQU	C'2'	DIRTYPE of SEARCHALL
DIRTSRHU	EQU	C'3'	DIRTYPE of SEARCHAUTH (same as SEARCHALL)
DIRTALIA	EQU	C'4'	DIRTYPE of ALIAS
DIRTAUTH	EQU	C'5'	DIRTYPE of AUTH
DIRTLOCK	EQU	C'6'	DIRTYPE of LOCK

```

DIRTDIR EQU C'7'      DIRTYPE of DIR
DIRTEXT EQU C'8'      DIRTYPE of FILEEXT
*
DIRCBASE EQU C'1'      status = BASE
DIRCALIA EQU C'2'      status = ALIAS
DIRCERAS EQU C'3'      status = ERASED
DIRCREVO EQU C'4'      status = REVOKED
DIRCDIR EQU C'5'      status = DIR
DIRCEXT EQU C'6'      status = EXTRNL
*
DIRCMD EQU C'7'      status = Minidisk
DIRCFIX EQU C'F'      rec. format = fixed
DIRCVAR EQU C'V'      rec. format = variable
DIRCDIRF EQU C'D'      rec. format = dir
DIRCERSF EQU C'-'      rec. format = erased
*
DIRCSHAR EQU C'1'      lock type = shared
DIRCEXCL EQU C'2'      lock type = exclusive
DIRCUPDT EQU C'3'      lock type = update
DIRCSESS EQU C'1'      lock length = session
DIRCLAST EQU C'2'      lock length = lasting
*
DIROSFS EQU C'0'      Filesystemtype = SFS file space
DIROBFS EQU C'1'      Filesystemtype = BFS file space
*
DIRDATA DS 0F          Start of records
*
*      Get Directory record for FILE
*
*
      AIF ('&INTENT' EQ 'ALL').FILEMAP
      AIF ('&INTENT' NE 'FILE').FILEEXT
.FILEMAP ANOP
      ORG DIRDATA
DIRFILE DS 0F          Get Directory for FILE
DIRFFN DS CL8          Filename
DIRFFT DS CL8          Filetype
      ORG DIRFFN
DIRFSUBD DS CL16       Subdirectory name(16 characters)
DIRFFMN DS CL1         Filemode number
DIRFRECF DS CL1        Record Format
      DS CL1           Reserved
DIRFFM DS CL1          File Mode (blank if Dir)
DIRFRECL DS F          Record length
DIRFBLKS DS F          Number of Blocks
DIRFRECS DS F          Number of Records
DIRFDATD DS XL3        Date (decimal yymmdd)
DIRFATTR DS CL1        Directory attribute
DIRFTIMD DS XL3        Time (decimal hhmmss)
DIRFMIGR DS CL1        Migrated file
DIRFDATC DS CL8        Date (character yy/mm/dd)
DIRFTIMC DS CL8        Time (character hh:mm:ss)
DIRFUSER DS CL8        Userid
DIRFSTAT DS CL1        Status
DIRFRATH DS CL1        Read Authority
DIRFWATH DS CL1        Write Authority
DIRFPROT DS CL1        External protection indicator
DIRFUNQD DS CL16       Unique Id
DIRFDAXD DS XL4        Date (decimal yyyyymmdd)
DIRFDAXC DS CL10       Date (character yyyy/mm/dd)
DIRFDAXI DS CL10       Date (character yyyy-mm-dd)
DIRFCEND DS 0F
DIRFLEN EQU *-&DNAME   Length of FILE record
DIRFR1L EQU DIRFLEN-(DIRFCEND-DIRFUNQD)      R1 Length
DIRFLV13 EQU DIRFLEN-(DIRFCEND-DIRFDAXD)      cmslevel 13 len
*
*      Get Directory record for FILEEXT
*
*
      AIF ('&INTENT' EQ 'ALL').EXTMAP
      AIF ('&INTENT' NE 'FILEEXT').SRCHA
.FILEEXT AIF
.EXTMAP ANOP
      ORG DIRDATA
DIREXT DS 0F          Get Directory for FILEEXT
DIREFN DS CL8          Filename
DIREFT DS CL8          Filetype
      ORG DIREFN
DIRESUBD DS CL16       Subdirectory name (16 chars)
DIREFMN DS CL1         Filemode number
DIRERECF DS CL1        Record Format
DIRERECV DS CL1        Recoverability

```



```

DIREQVWR DS CL1 Over Write
DIRERECL DS F Record length
DIREBLKS DS F Number of Blocks
DIRERECS DS F Number of Records
DIREDATD DS XL3 Date (decimal yymmdd)
DIREATTR DS CL1 Directory attribute
DIRETIMD DS XL3 Time (decimal hhmmss)
DIREMIGR DS CL1 Migrated file
DIREDATC DS CL8 Date (character yy/mm/dd)
DIRETIMC DS CL8 Time (character hh:mm:ss)
DIREUSER DS CL8 Userid
DIRESTAT DS CL1 Status
DIRERATH DS CL1 Read Authority
DIREWATH DS CL1 Write Authority
DIREPROT DS CL1 External protection indicator
DIREDLRD DS XL3 Date of Last Ref(decimal yymmdd)
DS XL1 Reserved
DIREDLRC DS CL8 Date of Last Ref (char yy/mm/dd)
DIRECDTD DS XL3 Creation Date (decimal yymmdd)
DS XL1 Reserved
DIRECTMD DS XL3 Creation Time (decimal hhmmss)
DS XL1 Reserved
DIRECDTC DS CL8 Creation Date (char yy/mm/dd)
DIRECTMC DS CL8 Creation Time (char hh:mm:ss)
DIREMAXB DS F Maximum Blocks
DIREDATB DS F Data Blocks
DIRESYSB DS F System Blocks
DS XL1 Reserved
DIREDRA1 DS CL8 DRA field 1
DIREDRA2 DS CL8 DRA field 2
DIREDRA3 DS CL8 DRA field 3
ORG DIREDRA1
DIREDRAS DS CL24 DRA fields
DIREUNQD DS CL16 Unique Id
ORG DIREMAXB
DIRETDFM DS CL53
DIREDLCD DS XL3 Date Of Last Change (yymmdd)
DS XL1
DIRETLCD DS XL3 Time Of Last Change (hhmmss)
DS XL1
DIREDLCC DS CL8 Date Of Last Change (yy/mm/dd)
DIRETLCC DS CL8 Time Of Last Change (hh:mm:ss)
ORG DIREDLCD
DIREDFDS DS CL24DIREPAD DS CL50
DIREEND DS 0F
ORG DIREPAD
DIREAXD DS XL4 Date (decimal yyyymmdd)
DIREAXC DS CL10 Date (character yyyy/mm/dd)
DIREAXI DS CL10 Date (character yyyy-mm-dd)
DIREDRXD DS XL4 Date of Last Ref (dec yyyymmdd)
DIREDRXC DS CL10 Date of Last Ref(char yyyy/mm/dd)
DIREDRXI DS CL10 Date of Last Ref(char yyyy-mm-dd)
DIRECDXD DS XL4 Creation Date (decimal yyyymmdd)
DIRECDXC DS CL10 Creation Date (char yyyy/mm/dd)
DIRECDXI DS CL10 Creation Date (char yyyy-mm-dd)
DIREDCXD DS XL4 Date of Last Change(dec yyyymmdd)
DIREDCXC DS CL10 Date of Last Chg(char yyyy/mm/dd)
DIREDCXI DS CL10 Date of Last Chg(char yyyy-mm-dd)
DS CL3 Reserved
DIRECEND DS 0F
DIRELEN EQU *-&DNAME Length of FILEEXT record
DIRER21L EQU DIRELEN-(DIRECEND-DIREDFDS-11) Rel 2.1 Length
DIRLV13 EQU DIRELEN-(DIRECEND-DIREEND+1) cmslv1 13 length
*
*
* Get Directory record for SEARCHALL and SEARCHAUTH
*
*
.SRCHA AIF ('&INTENT' EQ 'ALL').SRCHMAP
AIF ('&INTENT' EQ 'SEARCHALL').SRCHMAP
AIF ('&INTENT' NE 'SEARCHAUTH').ALIAS
.SRCHMAP ANOP
ORG DIRDATA
DIRSRCH DS 0F Get Directory for SEARCHALL/SEARCHAUTH
DIRSFN DS CL8 Filename
DIRSFT DS CL8 Filetype
ORG DIRSFN
DIRSSUBD DS CL16 Subdirectory name(16 characters)
DIRSFMN DS CL1 Filemode number
DIRSRECF DS CL1 Record Format
DS CL2 Reserved
DIRSRECL DS F Record length

```

```

DIRSBLKS DS F Number of Blocks
DIRSRECS DS F Number of Records
DIRSDATD DS XL3 Date (decimal yymmdd)
DIRSATTR DS CL1 Directory attribute
DIRSTIMD DS XL3 Time (decimal hhmmss)
DIRSMIGR DS CL1 Migrated file
DIRSDATC DS CL8 Date (character yy/mm/dd)
DIRSTIMC DS CL8 Time (character hh:mm:ss)
DIRSUSER DS CL8 Userid
DIRSSTAT DS CL1 Status
DIRSRATH DS CL1 Read Authority
DIRSWATH DS CL1 Write Authority
DIRSPROT DS CL1 External protection indicator
DIRSDIR DS 0F Directory id
DIRSNLEN DS X Directory name length
DIRSNAME DS CL153 Directory name
DS CL2 Reserved
DIRSDAXD DS XL4 Date (decimal yyyymmdd)
DIRSDAXC DS CL10 Date (character yyyy/mm/dd)
DIRSDAXI DS CL10 Date (character yyyy-mm-dd)
DIRSCEND DS 0F
DIRSLEN EQU *-&DNAME Length of SEARCH record
DIRSLV13 EQU DIRSLEN-(DIRSCEND-DIRSDAXD+2) cmslvl 13 length
*
*
* Get Directory record for ALIAS
*
*
*
      AIF ('&INTENT' EQ 'ALL').ALSMAP
.ALIAS AIF ('&INTENT' NE 'ALIAS').AUTH
.ALSMAP ANOP
      ORG DIRDATA
DIRALIAS DS 0F Get Directory for ALIAS
DIRAINFN DS CL8 Input Filename
DIRAINFT DS CL8 Input Filetype
DIRAINMN DS CL1 Input Filemode number
DIRASTAT DS CL1 Status
DIRAMIGR DS CL1 Migrated file
DS CL1 Reserved
DIRAALNM DS F Number of aliases
DIRADIR DS 0F Directory id
DIRANLEN DS X Directory name length
DIRANAME DS CL153 Directory name
DIRAOTFN DS CL8 Output Filename
DIRAOTFT DS CL8 Output Filetype
DIRAOTMN DS CL1 Output Filemode number
DIRAUSER DS CL8 Owner userid
DIRALEN EQU *-&DNAME Length of ALIAS record
*
*
* Get Directory record for AUTH
*
*
*
      AIF ('&INTENT' EQ 'ALL').AUTHMAP
.AUTH AIF ('&INTENT' NE 'AUTH').LOCK
.AUTHMAP ANOP
      ORG DIRDATA
DIRAUTH DS 0F Get Directory for AUTH
DIRUFN DS CL8 Filename
DIRUFT DS CL8 Filetype
      ORG DIRUFN
DIRUSUBD DS CL16 Subdirectory name(16 characters)
DIRUFMN DS CL1 Filemode number
DIRUSTAT DS CL1 Status
DIRURATH DS CL1 Read Authority
DIRUWATH DS CL1 Write Authority
DIRUPROT DS CL1 External protection
DIRUOWNR DS CL8 Owner userid
DIRUDRAT DS CL1 Directory read authority
DIRUDWAT DS CL1 Directory write authority
DIRUNRAT DS CL1 NEWREAD authority
DIRUNWAT DS CL1 NEWWRITE authority
DIRUATTR DS CL1 Directory attribute
DIRUMIGR DS CL1 Migrated file
DIRUCEND DS 0F
DIRULEN EQU *-&DNAME Length of AUTH record
DIRUR7L EQU DIRULEN-(DIRUCEND-DIRUMIGR) Rel 1 Length
DIRUR6L EQU DIRULEN-(DIRUCEND-DIRUATTR) Rel 6 Length
*
*
* Get Directory record for LOCK

```

```

*
*
      AIF      ('&INTENT' EQ 'ALL').LOCKMAP
.LOCK      AIF      ('&INTENT' NE 'LOCK').DIR
.LOCKMAP   ANOP
      ORG      DIRDATA
DIRLOCK    DS      0F              Get Directory for LOCK
DIRLFN     DS      CL8             Filename
DIRLFT     DS      CL8             Filetype
      ORG      DIRLFN
DIRLSUBD   DS      CL16            Subdirectory name(16 characters)
DIRLFMN    DS      CL1             Filemode number
DIRLSTAT   DS      CL1             Status
DIRLTYPE   DS      CL1             Lock Type
DIRLLNTH   DS      CL1             Lock Length
DIRLUSER   DS      CL8             Lock holder userid
DIRLATTR   DS      CL1             Directory attr.
DIRLMIGR   DS      CL1             Migrated file
DIRLCEND   DS      0F
DIRLLEN    EQU     *-&DNAME         Length of LOCK record
DIRLR1L    EQU     DIRLLEN-(DIRLCEND-DIRLMIGR)    R1 Length
DIRLR6L    EQU     DIRLLEN-(DIRLCEND-DIRLATTR)    SP6 length
*
*
*      Get Directory record for DIR
*
*
      AIF      ('&INTENT' EQ 'ALL').DIRMAP
.DIR      AIF      ('&INTENT' NE 'DIR').DIREND
.DIRMAP    ANOP
      ORG      DIRDATA
DIRDIR     DS      0F              Get Directory for DIR
DIRDDIRD   DS      0F              Directory id
DIRDNLEN   DS      X              Directory name length
DIRDNAME   DS      CL153           Directory name
DIRDATTR   DS      CL1             Directory attribute
DIRDCEND   DS      0F
DIRDLEN    EQU     *-&DNAME         Length of DIR record
DIRDR6L    EQU     DIRDLEN-(DIRDCEND-DIRDATTR)    SP6 length

```

DMSABEXP

➤ DMSABEXP ➤

Purpose

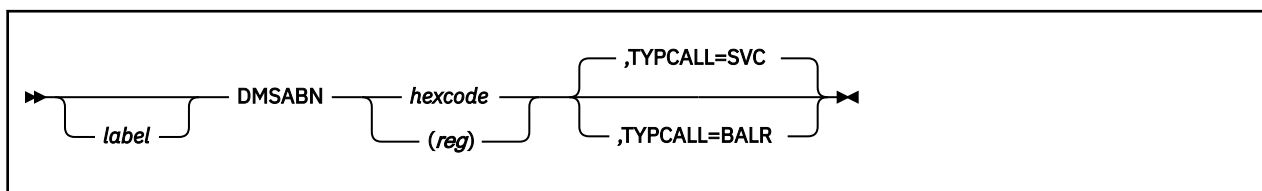
Use the DMSABEXP macroinstruction with the DCB abend exit to map the parameter list. For an example of how to use this macro, see the [z/VM: CMS Application Development Guide for Assembler](#).

Usage Notes

- 1. The DMSABEXP mapping macro expands as follows:

ABENDEXP	DSECT		DCB abend exit parameter list
ABEXPARM	DS	0F	Parameters:
ABEXSCC	DS	H	System completion code
ABEXRC	DS	X	Return code
ABEXOPT	DS	X	Options mask
ABEIGNOR	EQU	X'04'	Indicates that the error can be ignored
ABEXDCB	DS	A	Address of the DCB with the error
ABEXWAA	DS	A	Work area address, unsupported
ABEXRSV	DS	X	Reserved
ABEXRWA	DS	3X	Recovery work area address, not supported
ABEDWLGT	EQU	((*-ABENDEXP)+7)/8	Length of the parameter list in doublewords
*			
ABEYLG	EQU	*-ABENDEXP	Length of the parameter list in bytes

DMSABN



Purpose

Use the DMSABN macro to abnormally end (abend) a program. The first three hexadecimal digits of the system abend code appear in the CMS abend message, DMSABE148T.

Parameters

Required Parameters:

hexcode

is the abnormal termination code (0 through FFF) that appears in the DMSABE148T system termination message.

(reg)

is a register containing the abnormal termination code.

Optional Parameters:

label

is an optional assembler label for the statement.

TYPCALL=

specifies how control is passed to the CMS abnormal termination routine. Acceptable values are:

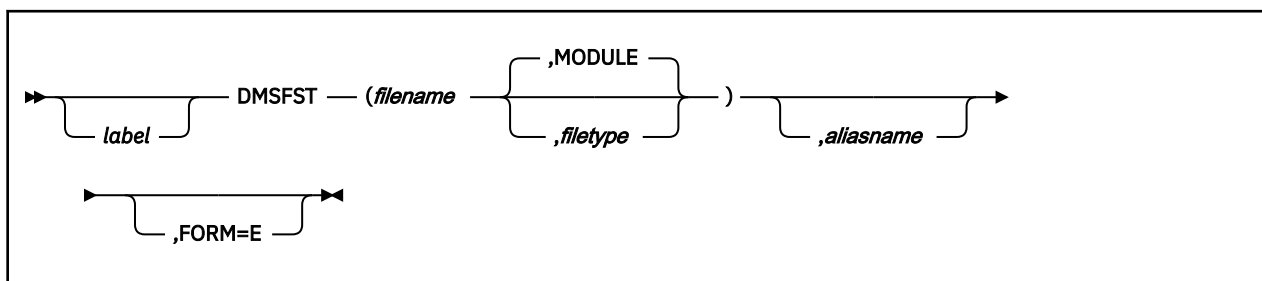
SVC

generates CMSCALL linkage to the CMS abnormal termination routine. Routines that do not reside in the nucleus should use TYPCALL=SVC. This is the default value.

BALR

generates a direct branch to the CMS abnormal termination routine. Nucleus-resident routines should specify TYPCALL=BALR.

DMSFST



Purpose

Use the DMSFST macro to set up a file status table for a specific file when building an auxiliary directory.

Parameters

Required Parameters:

filename

is the name of the module whose file status table (FST) information is to be copied.

Optional Parameters:

label

is an optional assembler label for the statement.

filetype

is the module type whose file status table (FST) information is to be copied. The default file type is MODULE.

aliasname

is another name for the module.

FORM=E

specifies that 64-byte FST entries are to be generated rather than 40-byte entries. Either length FST entry operates correctly on CMS; however, the 40-byte form does not contain such information as date/time after initialization by GENDIRT.

DMSJNEPL

►► DMSJNEPL ◄◄

Purpose

Use the DMSJNEPL macro to map the parameter list used by the DMSJNE exit routine.

Usage Notes

1. For more information on the DMSJNEPL macro, see [z/VM: CMS File Pool Planning, Administration, and Operation](#).
2. The DMSJNEPL macro expands as follows:

```

MACRO
DMSJNEPL
DSECT
JNEPL
*-----*
*      Plist used when calling DMSJNE      *
*-----*
*
JNEMOD  DS    CL8      Identifies Namelist user exit
*                      (always 'DMSJNE ')
JNEFUNCT DS    CL8      Function call
*                      'USERNODE' - Return a localid when
*                      passed a user ID and node
JNEUSER  DS    CL8      User ID
JNENODE  DS    CL8      Node
JNEFPID  DS    CL8      Filepool id (without ':')
JNELOCID DS    CL8      User supplied localid
JNEPLSIZ EQU  *-JNEPL   Length of JNEPL

```

DMSQEFL

➤ DMSQEFL ➤

Purpose

Use the DMSQEFL macroinstruction to determine the level of CMS as defined by the DMSQEFL CSL routine.

The CMS level value is returned in Register 15.

For mapping returned values to CMS levels, see DMSQEFL CSL in the *z/VM: CMS Callable Services Reference*.

DMSSDWA

► DMSSDWA ◄

Purpose

Use the DMSSDWA macroinstruction to generate a DSECT for the CMS-provided System Diagnostic Work Area, which maps the area pointed to by register 1 upon entry to an ABNEXIT routine. This area contains information concerning the original error that initiated the abend process.

Usage Notes

1. The DMSSDWA macro expands as follows:

```

MACRO
DMSSDWA
CMSSDWA DSECT ,
*****
* CMS provided System Diagnostic Work Area
* The following fields applies to all CMS virtual machine
* modes and is considered the CMS SDWA base section.
*****
SDWREGS DS XL64 GP regs at time of ABEND
SDWPSW DS XL8 PSW at time of ABEND
SDWSA DS XL72 Savearea pointed to by R13
SDWSALNT EQU *-SDWSA Length of save area
SDWFP RS DS XL32 FP regs at time of ABEND
DS 0D
SDWOPSW DS 0XL40 Old PSW fields
SDWEOPSW DS XL8 External old PSW
SDWSOPSW DS XL8 Supervisor-call old PSW
SDWPOPSW DS XL8 Program old PSW
SDWMOPSW DS XL8 Machine-check PSW
SDWIOPSW DS XL8 I/O old PSW
SDWVSTR DS XL8 Vector Status Register
SDWTXID DS XL1 Contains the ID of the access
* reg involved with the program
* chk that occurred on an AR
X reference. On non XC-mode
* virtual machines, contains the
* ID of the general reg involved
* with the program chk that
* occurred on an AR reference
SDWFLAG1 DS XL1 The SDWFLAG1 field may be used
* by the ABNEXIT rtn to further
* determine abend cause.
SDWMCKAB EQU X'80' When on abend caused by MCH Ck
SDWPCKAB EQU X'40' When on abend caused by PGM Ck
SDWSVCAB EQU X'10' When on abend initiated by SVC
* DMSABN (SVC) or ABEND macro
* initiated. If flags SDWPCKAB,
* SDWMCKAB, SDWAVCAB are off, the
* abend was initiated via DMSABN
* (BALR) or direct branch to
* CMS native abend processing.
SDWFLAG2 DS XL1 The SDWFLAG2 field is used to
* convey information between CMS and
* the ABNEXIT rtn. Recovery action
* may need to be performed that is
* related to certain abend codes.
* This action may have been
* performed by CMS prior to entry
* to the ABNEXIT rtn. In some
* cases CMS may not be able to
* perform the recovery action, in
* this case the ABNEXIT should
* attempt the action. If successful,
* the corresponding flag should be
* be set by the exit, telling CMS

```

```

* the recovery action has been
* completed.
SDWFSRPL EQU X'80' Indicates that the failing
* storage page within the Data
* Space identified by the SDWASIT
* field has been released. This
* may be set prior to invoking the
* ABNEXIT rtn, indicating CMS has
* already done the release. If CMS
* couldn't do the release, the
* exit should do it and set the
* flag. This flag is applicable to
* X'1F4' abends.
*
* DS XL1 Reserved for future IBM use
* DS 1F Reserved for future IBM use
SDWMCIC DS XL8 Mach chk int code,
* valid when mach chk initiated
* abend, else will be 0's
SDWIINFO DS 0XL4 Following 4 bytes relate to XA
* PSW interrupt information
* DS XL1 Reserved for IBM use
SDWPILC DS XL1 ILC associated with XA PSW at
* abend. For 370-mode machine
* ILC part of PSW.
SDWINTCD DS XL2 Int code, related with XA PSW
* - Pgm chk abend, = to int code
* - SVC initiated abend, = to
* svc int code.
* - Branch entered, will be 0
SDWUWORD DS XL4 User word specified via ABNEXIT
SDWABNCD DS XL4 Abend code on entry to exit rtn
SDWABNRC DS XL4 Reason code on entry to exit rtn
SDWEXPTR DC A(SDWEPTRS) Pntr to CMS SDWA extention pntrs
SDWENDBS DS 0D End of CMS's base SDWA section
*****
* The following section applies to XC-mode virtual machines,
* for non XC-mode virtual machines the fields are present
* but the content has no meaning.
*****
SDWXCS DSECT ,
SDWREGAR DS XL64 AR regs at time of ABEND
SDWALET DS XL4 ALET value related to prog chk
* that occured during an AR ref
SDWASIT DS XL8 ASIT value related to Mach chk
* that occured during an AR ref
* that gets reported as a stg ck
* used to identify the data space.
SDWFSA DS XL4 For a stg chk in a data space,
* this field contains the failing
* storage address. It can be used
* to determine the page address
* in which the error occurred.
SDWENDXC DS 0D End of CMS's SDWA XC-mode ext
*****
* The following section applies to Z-mode virtual machines,
* for non Z-mode virtual machines the fields are present
* but the content has no meaning.
*****
SDWZMS DSECT ,
ZDWREGSG DS XL128 64-bit regs at time of ABEND
ZDWPSW DS XL16 PSW at abend time
ZDWOPSW DS 0XL80 Old PSW fields
ZDWEOPSW DS XL16 External old PSW
ZDWSOPSW DS XL16 Supervisor-call old PSW
ZDWPOPSW DS XL16 Program old PSW
ZDWMOPSW DS XL16 Machine-check PSW
ZDWIOPSW DS XL16 I/O old PSW
SDWENDZM DS 0D End of CMS's SDWA Z-mode ext
*****
* The following section contains the pointers to other CMS
* SDWA extentions.
* This section must be at least a double word in length,
* any other SDWA section extentions referred to by the
* pointer addresses must be at least a double word in
* length.
* - this section is pointed to by the SDWEXPTR field in
* the CMSSDWA DSECT.
*****
SDWEPTRS DSECT , Pointed to by SDWEXPTR
SDWXCSP DS A Pointer to CMS's XC-mode sect
SDWZMSP DS A Pointer to CMS's Z/mode sect
SDWENDPS DS 0D End of CMS's SDWA pointers sect

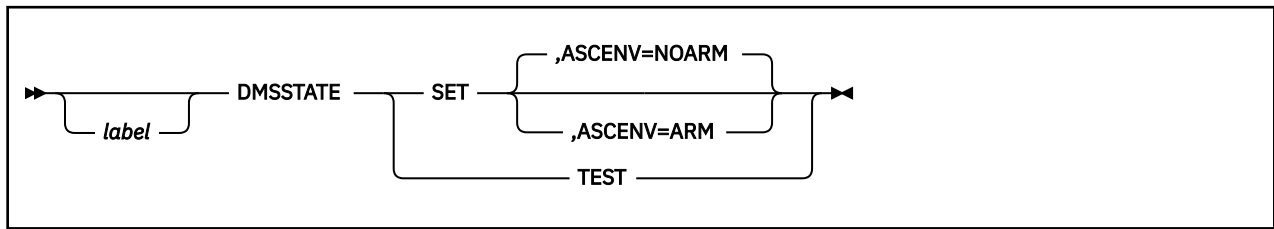
```

```

*****
*   The following contains the length calculations of the
*   individual CMS SDWA sections as well as the overall
*   length of the CMS SDWA with all its section extensions.
*****
SDWPTLEN EQU    SDWENDPS-SDWEPTRS    Length of Pointer section
SDWBSLEN EQU    SDWENDBS-CMSSDWA      Length of Base section
SDWXCLLEN EQU    SDWENDXC-SDWXCS      Length of XC section
SDWZMLLEN EQU    SDWENDZM-SDWZMS      Length of ZM section
SDWLNTH  EQU    SDWBSLEN+SDWXCLLEN+SDWZMLLEN+SDWPTLEN Real length

```

DMSSTATE



Purpose

Use the DMSSTATE macro in programs that call CMS preferred group macros while executing in access register (AR) mode on an XC virtual machine. By coding a DMSSTATE SET,ASCENV=ARM as the first macro in your program, you will ensure that subsequent preferred group macros will have the proper macro expansion for operating in AR mode. This allows programs to remain in AR mode even when calling CMS services.

Note: If DMSSTATE SET,ASCENV=ARM is coded at the beginning of an existing program that has a number of CMS preferred group macros and is near its base register addressing limits, addressability errors may occur during re-assembly due to the increase of code because of the new macro expansions. In this case, selective use of DMSSTATE should be considered. See Usage Note “3” on page 177 for more information.

Parameters

Required Parameters:

SET

specifies that you want to set the assembler global variable according to the specification of the ASCENV= parameter. If the ASCENV= parameter is not specified, the global variable is set to the default mode. The default mode that is provided by DMSSTATE is NOARM.

TEST

specifies that you want to set the assembler global variable to the default mode if it is not currently set.

Optional Parameters:

label

is an optional assembler label for the statement.

ASCENV=

specifies the translation mode, either AR mode or primary-space mode, for subsequent CMS preferred group macros within the same assembly as this DMSSTATE macro. An assembler global variable is set accordingly to cause the proper macro expansion.

NOARM

specifies that CMS preferred group macros should be expanded for operation in primary-space mode. This bypasses the additional code expansion required for the AR mode environment. This is the default value.

ARM

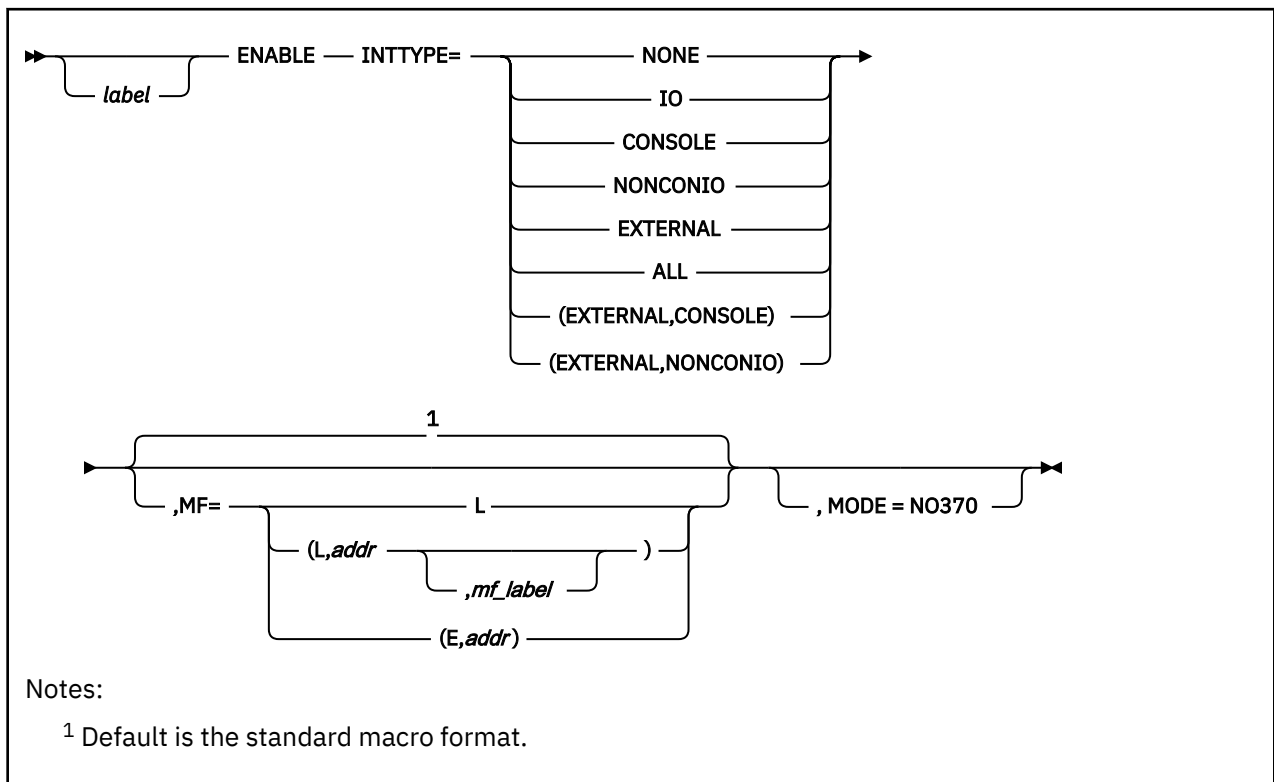
specifies that CMS preferred group macros should be expanded for operation in AR mode on an XC virtual machine. The macro expansion also provides for bypassing this code when execution is not on an XC virtual machine.

Usage Notes

1. The DMSSTATE macro does not affect the macro expansions of CMS preferred group macros in prior CMS releases.

2. The ASCENV=ARM parameter sets an assembler global variable that causes applicable CMS preferred macros to expand with code appropriate for execution in an XC virtual machine. This code is executed when in AR mode and is bypassed when execution is not in an XC virtual machine.
3. The DMSSTATE macro can be used selectively within a program to cause correct expansion for a specific section of code. You can code a DMSSTATE SET,ASCENV=ARM just before the CMS preferred group macro that will be operating in AR mode and then code a DMSSTATE SET,ASCENV=NOARM after the last macro that will be executing in AR mode. This will cause only the group of macros between the DMSSTATE calls to have the AR mode expansion.
4. As far as the application is concerned, DMSSTATE has no executable code associated with it. It works more like a compiler directive than a CMS service macro.

ENABLE



Purpose

Use the **ENABLE** macro to manipulate the PSW interrupt mask.

Parameters

Required Parameters:

INTTYPE=

indicates the types of interrupts to be enabled. Any interrupt types not specified are disabled. [Table 19 on page 179](#) summarizes what types of interrupts are enabled for each option. Acceptable values are:

NONE

disables all interrupts.

IO

enables all I/O interrupts.

CONSOLE

enables only for I/O interrupts from the virtual machine console. The interrupt subclass (ISC) for the console is enabled.

NONCONIO

enables for only nonconsole I/O interrupts. All ISCs except for the console ISC are enabled.

EXTERNAL

enables for external interrupts.

ALL

enables for all interrupts.

(EXTERNAL,CONSOLE)

enables for external interrupts and for I/O interrupts from the virtual machine console. The interrupt subclass (ISC) for the console is enabled.

(EXTERNAL,NONCONIO)

enables for external interrupts and nonconsole I/O interrupts. All ISCs except for the console ISC are enabled.

Optional Parameters:

label

is an optional assembler label for the statement.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

MODE=NO370

specifies that the macro should not create a System/370 code path.

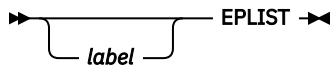
Usage Notes

1. The following table summarizes the types of interrupts enabled for each option:

Table 19. Summary of Interrupt Types Affected by ENABLE INTTYPE Options

Options	External	Console I/O	Other I/O
NONE	Disabled	Disabled	Disabled
EXTERNAL	Enabled	Disabled	Disabled
CONSOLE	Disabled	Enabled	Disabled
NONCONIO	Disabled	Disabled	Enabled
EXTERNAL,CONSOLE	Enabled	Enabled	Disabled
EXTERNAL,NONCONIO	Enabled	Disabled	Enabled
IO	Disabled	Enabled	Enabled
ALL	Enabled	Enabled	Enabled

EPLIST



Purpose

Use the EPLIST macro to generate a DSECT for the extended parameter list.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the EPLIST macro expansion is labeled EPLIST.

Usage Notes

1. For more information on the EPLIST macro, see [z/VM: CMS Application Development Guide for Assembler](#) and [z/VM: REXX/VM Reference](#).
2. The EPLIST macroinstruction expands as follows:

```

EPLIST
*
***      EPLIST - EXTENDED PLIST DSECT
*
EPLIST   DSECT
EPLCMD   DS      A           ADDRESS OF COMMAND TOKEN.
EPLARGBG DS      A           ADDR OF BEGINNING OF ARGUMENTS.
EPLARGND DS      A           ADDR OF END OF ARGUMENTS.
EPLUWORD DS      A           USER WORD
*
EPL4LNBY EQU  *-EPLIST      4 WORD HEADER LENGTH IN BYTES
EPL4LNDW EQU  (*-EPLIST+7)/8 4 WORD HEADER LENGTH IN DWORDS
EPARGLST DS      A           ADDRESS OF FUNCTION ARGUMENT LIST.
EPFUNRET DS      A           ADDRESS FOR RETURN OF FUNCTION
*                               DATA.
EPL6LNBY EQU  *-EPLIST      6 WORD HEADER LENGTH IN BYTES
EPL6LNDW EQU  (*-EPLIST+7)/8 6 WORD HEADER LENGTH IN DWORDS
*
          DS      2A           PADDING (FOR USE WITH SCAN MACRO)
EPLSCANT DS      0CL8         BEGINNING OF TOKENIZED PLIST
*                               BUILT BY SCAN MACRO.
*
EPLRSRVD EQU  EPLUWORD,4,C'A' (OLD NAME, FOR COMPATIBILITY)
* * * * *
*
*      THE EXTENDED PLIST FLAGS INDICATE THE PRESENCE
*      OF AN EXTENDED PLIST IN REGISTER 0. THE HIGH
*      ORDER BYTE OF REGISTER 1 WILL CONTAIN EITHER
*      EPLCMDFL OR EPLFNCFL TO INDICATE THE EXTENDED
*      PLIST IS AVAILABLE. ONLY THE FIRST 4 WORDS OF
*      OF THE EXTENDED PLIST ARE AVAILABLE WITH THESE
*      CODES.
*
*      IF THE HIGH ORDER BYTE OF REGISTER 1 CONTAINS
*      EPFUNSUB, THEN THE INVOCATION IS AN EXTERNAL
*      FUNCTION/SUBROUTINE CALL FROM REXX. WITH THIS
*      PLIST, ALL 6 WORDS OF THE PLIST ARE AVAILABLE.
*      WORD 5 POINTS TO A LIST OF DOUBLEWORD ADLENS
*      (ADDRESS-LENGTH PAIRS) WHICH DESCRIBE THE
*      ARGUMENTS TO THE ROUTINE (EPARGLST). WORD 6
*      (EPFUNRET) IS THE LOCATION FOR THE CALLED
*      ROUTINE TO STORE THE ADDRESS OF AN EVALBLOK
*      TO RETURN DATA TO THE CALLING PROGRAM.
*

```

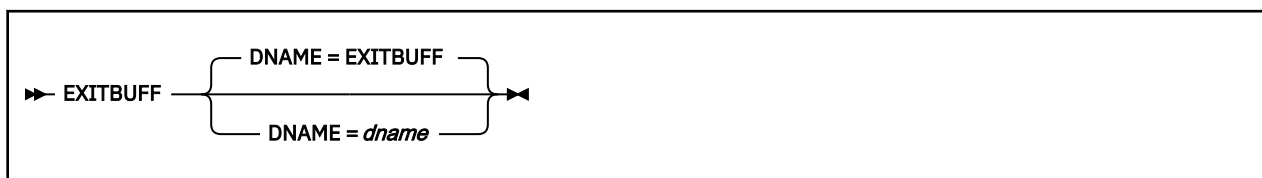


```

*          **** NOTE ****          *
*      IF THE CALLED PROGRAM IS AN AMODE 24          *
*      PROGRAM, THE HIGH ORDER BYTE OF REGISTER 1    *
*      CONTAINS THIS CALL TYPE INFORMATION.          *
*      IF THE CALLED PROGRAM IS AMODE 31 OR AMODE    *
*      ANY, THE HIGH ORDER BYTE OF REGISTER 1 IS     *
*      PART OF THE ADDRESS.                          *
*      THE USECTYP FIELD IN THE USER SAVE AREA ALSO *
*      CONTAINS CALL TYPE INFORMATION REGARDLESS     *
*      OF THE CALLING PROGRAMS AMODE.              *
*
* * * * *
EPLCMDL EQU  X'0B'      EXTENDED PLIST AVAILABLE FLAG.
EPLFNCFL EQU  X'01'      EXTENDED PLIST AVAILABLE FLAG.
EPFUNSUB EQU  X'05'      EXTERNAL FUNCTION PLIST AVAILABLE
*
* FLAG DEFINITIONS.  EXCEPT AS NOTED, ONLY THE FIRST FOUR
* WORDS OF THE EXTENDED PLIST ARE AVAILABLE.
*
*      EPLIST
* FLAG      VALUE      AVAIL? MEANING
EPLFPROG EQU  X'00'      N PROGRAM
EPLFCMND EQU  X'01'      Y ADDRESS COMMAND
EPLFSBCM EQU  X'02'      Y SUBCOM
EPLFNNUC EQU  X'03'      Y NO NUCEXT, EXTENDED
EPLFNNUT EQU  X'04'      N NO NUCEXT, TOKENIZED
EPLFRXFN EQU  X'05'      Y REXX EXTERNAL FUNCTION,
*                        6 WORD EXTENDED PLIST PRESENT
EPLFIMMD EQU  X'06'      Y IMMEDIATE COMMAND
EPLFSRCH EQU  X'0B'      Y COMMAND SEARCH
EPLFEXEC EQU  X'10'      N INVOKED BY BPX1EXC
EPLFENDC EQU  X'FE'      N END OF COMMAND
EPLFABEN EQU  X'FF'      N ABEND OR NUCXDROP

```

EXITBUFF



Purpose

Use the EXITBUFF macro to generate a DSECT for the general data buffer that SFS provides for the File Space Usage and User Storage Group Full exits.

Parameters

Optional Parameters:

DNAME=*dname*

specifies an optional DSECT label. The default is EXITBUFF.

Usage Notes

1. For more information on the EXITBUFF macro, see [z/VM: CMS File Pool Planning, Administration, and Operation](#).
2. The EXITBUFF macro expands as follows:

```

*          EXITBUFF  &DNAME=EXITBUFF
*
*          *****
*          Constants
EXICSLN  DC      C'DMSSFSEX'          Other CSL name
EXIEYECA DC      C'EXITBUFF'          Eye catcher
EXIEYECB DC      C'EXITFSRW'          Eye catcher
EXIEYECC DC      C'EXILIST '          Eye catcher
&DNAME   DSECT
*          *****
*          EXITBUFF MAPPING MACRO
*          *****
EXIEYEC  DS      CL8                  Eye catcher - "EXITBUFF"
EXIFUNC  DS      0CL4                  Function Code
EXITID   DS      X                     Exit ID
          DS      CL3                  Reserved
EXIMACID DS      CL8                  SFS Machine ID
EXIFPID  DS      CL8                  Filepoolid
EXIREQID DS      CL8                  Requester ID
EXIRSRVD DS      CL24                 Reserved
EXIRPTR  DS      A                     Remainder pointer
EXIBSIZE EQU    *-&DNAME               Size of EXITBUFF
*
EXITFSRW DSECT
EXIEYE1  DS      CL8                  Eye catcher - "EXITFSRW"
EXIPTR1  DS      A                     Pointer to first entry in
*          following list
EXILCNT  DS      F                     List count
EXISG    DS      H                     Storage group
EXISGTHR DS      H                     Storage group threshold
EXISGSIZ DS      F                     Storage group size
EXISGAVA DS      F                     SG blocks available
EXICSIZE EQU    *-EXITFSRW            Size of EXITFSRW
*
EXILIST  DSECT
EXIEYE2  DS      CL8                  Eye catcher - "EXILIST"
EXINPTR  DS      A                     Pointer to next list entry
EXICONID DS      CL8                  Owner of connection
EXIALTID DS      CL8                  Alternate user ID

```

EXINNEWID	DS	CL8	New connect user ID
EXILUWGP	DS	F	LUW grouping
EXIFSID	DS	CL8	Filespace ID
EXIFSTHR	DS	F	Filespace threshold
EXIFSSIZ	DS	F	Filespace size
EXIFSCOM	DS	F	FS blocks committed
EXIWUFUN	DS	F	FS blocks uncommitted
EXIINFL	DS	XL1	In flags
EXIINCOM	EQU	X'80'	In process of committing
EXIINTRI	EQU	X'40'	This connection triggered the exit
*			
EXIOUTFL	DS	XL1	Out flags
EXIOUTRO	EQU	X'80'	Rollback this connection
	DS	CL2	Reserved
EXILSIZE	EQU	**EXILIST	Size of EXILIST
*			
*	Constants		
*	Other Exit ID definitions		
*			
EXIFSU	EQU	X'2'	2-Filespace Usage
EXIRAW	EQU	X'3'	3-User Storage Group Full
*			
*	CSL Routine return codes		
*			
EXISUCC	EQU	X'0'	Successful
EXINSCON	EQU	X'4'	Function not supported, continue to call.
*			
EXINSSUP	EQU	X'5'	Function not supported, suppress further calls.
*			
EXIOTHER	EQU	2	2-Other exit type

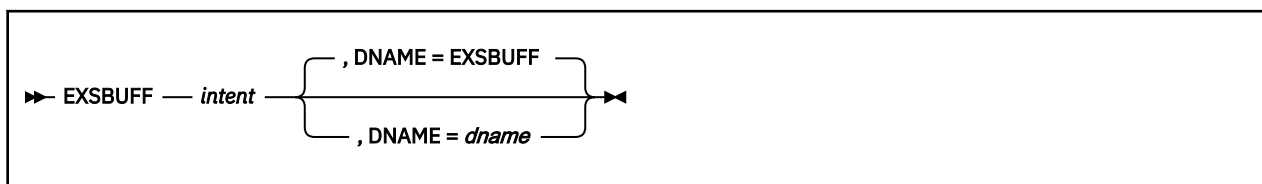
3. The fields in the general data buffer are defined as follows:

Table 20. General Data Buffer Fields		
Field	Data	Meaning
Beginning of General Data Buffer:		
EYE CATCHER	CHAR(8)	the eye catcher "EXITBUFF".
FUNCTION CODE	CHAR(4)	is a 4-byte field where: a. Byte 1: is the EXIT ID. It has a value of X'2' (File Space Usage) or X'3' (User Storage Group Full) b. Bytes 2, 3, 4: are reserved and have a value of binary 0.
SFS MACHINE ID	CHAR(8)	is the virtual machine identification (VMID) of the SFS server machine.
FILEPOOLID	CHAR(8)	is the file pool ID of the SFS server. It may be, but is not necessarily, the same as the SFS MACHINE ID.
REQUESTER ID	CHAR(8)	is the SFS-known ID of the user triggering the exit.
RESERVED	CHAR(24)	is an area reserved for IBM use. SFS sets this area to binary 0.
REMAINDER POINTER	PTR(31)	is a pointer to the remaining part of the buffer (EXITFSRW).
Part of General Data Buffer Common to Both Exits:		
EYE CATCHER	CHAR(8)	is the eye catcher "EXITFSRW".
POINTER TO TOP	PTR(31)	is a pointer to the first list entry. There is only one list entry for the File Space Usage exit. There are one or more entries for the User Storage Group Full exit, where there is a corresponding entry for each work unit and connection to a file space in the storage group.
NUMBER OF ENTRIES	FIXED(32)	is the number of list entries which follows.
STORAGE GROUP	FIXED(16)	is the storage group number.
GROUP THRESH	FIXED(16)	is the storage group threshold number.
BLOCKS IN SG	FIXED(32)	is the number of 4K blocks in the storage group.
BLOCKS AVAILABLE	FIXED(32)	is the number of 4K blocks available in the storage group.
List Entries:		
EYE CATCHER	CHAR(8)	is the eye catcher "EXILIST".

Table 20. General Data Buffer Fields (continued)

Field	Data	Meaning
NEXT POINTER	PTR(31)	is a pointer to the next list entry. The pointer is zero for the last list entry.
CONNECTION ID	CHAR(8)	is the real VM user ID of the owner of the connection. The same user ID appears in multiple entries in the list if the user has multiple work units or is connected to multiple file spaces. If the user connected to SFS after issuing DIAGNOSE code X'D4' for specifying an alternate user ID, this field still contains the user's real VM user ID.
ALTERNATE ID	CHAR(8)	is the alternate user ID specified by the DIAGNOSE code X'D4', or is the user ID of the owner of the connection. The same user ID appears in multiple entries in the list if the user has multiple work units or is connected to multiple file spaces.
NEW CONNECT ID	CHAR(8)	If the user connected to SFS changed the user ID by specifying the optional <i>userid</i> parameter of DMSGETWU, this field contains the specified 'new user connect' user ID. Otherwise it contains the alternate user ID.
LUW GROUPING	FIXED(32)	is an identifier that groups together all list entries associated with the same logical unit of work.
FILESPACE ID	CHAR(8)	is the user ID of the owner of the file space.
FILESPACE THRESH	FIXED(32)	is the file space threshold value.
SIZE OF FILESPACE	FIXED(32)	is the number of the blocks in the file space.
COMMITTED IN FS	FIXED(32)	is the number of file space blocks committed in the file space. A rollback of this work unit will not affect this number.
UNCOMMITTED IN FS	FIXED(32)	is the number of uncommitted file space blocks in the file space allocated in this work unit by this user. This number only includes new allocations. A rollback of this work unit will make this number of blocks available.
IN FLAGS	BIT(8)	are flags set by SFS; a flag of '1'B means the condition is met. Flags are defined for: <ul style="list-style-type: none"> • Bit 0: This connection has done a Prepare to Commit or is doing a commit and therefore is not eligible to be rolled back. The data associated with this connection is included for informational purposes only. • Bit 1: This connection triggered the exit. • Bits 2-7: Reserved.
OUT FLAGS	BIT(8)	are flags set by the exit and used by SFS upon return from the exit, when the return code is 0; a flag of '1'B means the condition is met. Flags are defined for: <ul style="list-style-type: none"> • Bit 0: Roll back this connection. The exit can select any or all of the active users for whom there is a list entry (except those not eligible in the IN FLAGS above) for rollback. If the exit doesn't pick one, the server implements its current policy for the exit type: <ul style="list-style-type: none"> – File Space Usage: Continue normal processing. – User Storage Group Full: Roll back the logical unit of work that triggered the storage group full condition. • Bits 1-7: Reserved.

EXSBUFF



Purpose

Use the EXSBUFF macroinstruction to map the records returned by an Exist request for a file or a directory.

Parameters

Required Parameters:

intent

specifies the type of directory record to be mapped. Legal types are FILE, DIR, and ALL. If ALL is specified, both record mappings are generated.

Optional Parameters:

DNAME=dname

specifies an optional DSECT label. If none is provided, the default label is EXSBUFF.

Usage Notes

- Mapping of these buffers is also possible using the callable services library (CSL) routine Exist (DMSEXIST). These buffers can be mapped into variables by using the CSL routines Exist - Directory (DMSEXIDI) and Exist - File (DMSEXIFI). These routines are discussed in [z/VM: CMS Callable Services Reference](#).
- The EXSBUFF mapping macro expands as follows:

```

EXSBUFF
EXSTYPE DS CL1      Type of record
EXSTFILE EQU C'1'   EXSTYPE of FILE
EXSTDIR EQU C'2'    EXSTYPE of DIR
EXSFSTYP DS CL1     Filesystemtype
EXSOSFS EQU C'0'    Filesystemtype = SFS file space
EXSOBFS EQU C'1'    Filesystemtype = BFS file space
EXSOMD EQU C'2'     Filesystemtype = minidisk
EXSLEN DS H         Length of actual data passed back*
EXSDATA DS 0F       Start of records
*
*      Exist record for FILE
*
*
      AIF ('&INTENT' EQ 'ALL').FILEMAP
      AIF ('&INTENT' NE 'FILE').DIRMAP
.FILEMAP ANOP
      ORG EXSDATA
EXSFILE DS 0F       Exist for FILE
EXSFFN DS CL8       Filename
EXSFFT DS CL8       Filetype
      ORG EXSFFN
EXFSUBD DS CL16     Subdirectory name(16 characters)
EXSFFMN DS CL1      Filemode number
EXSFRECF DS CL1     Record Format
EXSFRECV DS CL1     Recoverability
EXSFOVWR DS CL1     Overwrite
EXSFRECL DS F       Record length
EXSBLKS DS F        Number of Blocks
EXSFRECS DS F       Number of Records

```

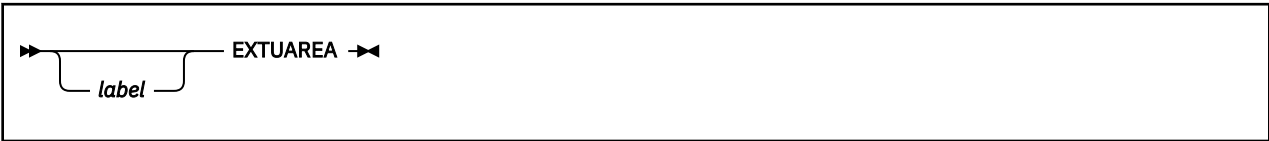
EXSFDATD	DS	XL3	Date (decimal yymmdd)
EXSFFM	DS	CL1	File mode
EXSFTIMD	DS	XL3	Time (decimal hhmmss)
EXSFRFM	DS	CL1	Real File Mode
EXSFDATE	DS	CL8	Date (character yy/mm/dd)
EXSFTIMC	DS	CL8	Time (character hh:mm:ss)
EXSFUSER	DS	CL8	Userid
EXSFSTAT	DS	CL1	Status
EXSFSTB	EQU	C'1'	Base
EXSFSTA	EQU	C'2'	Alias
EXSFSTE	EQU	C'3'	Erased
EXSFSTR	EQU	C'4'	Revoked
EXSFSTO	EQU	C'6'	External Object
EXSFSTM	EQU	C'7'	Minidisk
EXSFSTD	EQU	C'8'	OS or DOS formatted minidisk
EXSFRATH	DS	CL1	Read Authority
EXSFRYES	EQU	C'1'	Read Authority exists
EXSFRNO	EQU	C'0'	No Read Authority
EXSFWATH	DS	CL1	Write Authority
EXSFWYES	EQU	C'1'	Write Authority exists
EXSFWNO	EQU	C'0'	No Write Authority
EXSFPROT	DS	CL1	External protection indicator
EXSFPYES	EQU	C'1'	External protection exists
EXSFPNO	EQU	C'0'	No External protection
EXSFDLRD	DS	XL3	DoLR (decimal yymmdd)
	DS	XL1	Reserved
EXSFDLRC	DS	CL8	DoLR (character yy/mm/dd)
	ORG	EXSFDLRD	DoLR extended attributes
EXSFDOLR	DS	CL12	
EXSFCDTD	DS	XL3	Creation Date (decimal yymmdd)
	DS	XL1	Reserved
EXSFCTMD	DS	XL3	Creation Time (decimal hhmmss)
	DS	XL1	Reserved
EXSFCDDC	DS	CL8	Creation Date (char yy/mm/dd)
EXSFCTMC	DS	CL8	Creation Time (char hh:mm:ss)
	ORG	EXSFCDTD	DTOC extended attributes
EXSFDTOC	DS	CL24	
EXSFMAXB	DS	F	Maximum Blocks
EXSFDATE	DS	F	Data Blocks
EXSFSYSB	DS	F	System Blocks
EXSFMIGR	DS	CL1	Migrated file
EXSFDRA1	DS	CL8	DRA field 1
EXSFDRA2	DS	CL8	DRA field 2
EXSFDRA3	DS	CL8	DRA field 3
	ORG	EXSFDRA1	
EXSFDRA5	DS	CL24	DRA values
EXSFUNQD	DS	CL16	Unique id
	ORG	EXSFMAXB	
EXSFTDFM	DS	CL53	
EXSFDIRL	DS	X	Length of Directory ID
EXSFDIRD	DS	CL153	Directory ID
	ORG	EXSFDIRL	
EXSFCONV	DS	CL154	Dirname and length values
EXSFDLCD	DS	XL3	DOLC (decimal yymmdd)
	DS	XL1	Reserved
EXSFTLCD	DS	XL3	TOLC (decimal hhmmss)
	DS	XL1	Reserved
EXSFDLCC	DS	CL8	DOLC (char yy/mm/dd)
EXSFTLCC	DS	CL8	TOLC (char hh:mm:ss)
	ORG	EXSFDLCD	DTOLC extended attributes
EXSFDTLCD	DS	CL24	
EXSFDAXD	DS	XL4	Date (decimal yyyymmdd)
EXSFDAXC	DS	CL10	Date (character yyyy/mm/dd)
EXSFDAXI	DS	CL10	Date (character yyyy-mm-dd)
EXSFDXRD	DS	XL4	DoLR (decimal yyyymmdd)
EXSFDXRC	DS	CL10	DoLR (character yyyy/mm/dd)
EXSFDXRI	DS	CL10	DoLR (character yyyy-mm-dd)
EXSFCDDX	DS	XL4	Creation Date (decimal yyyymmdd)
EXSFCDDC	DS	CL10	Creation Date (char yyyy/mm/dd)
EXSFCDDI	DS	CL10	Creation Date (char yyyy-mm-dd)
EXSFDCCX	DS	XL4	DOLC (decimal yyyymmdd)
EXSFDCCX	DS	CL10	DOLC (character yyyy/mm/dd)
EXSFDCCI	DS	CL10	DOLC (character yyyy-mm-dd)
	ORG	EXSFDAXD	Date Extensions, added cmslvl 13
EXSF2000	DS	CL96	these are for year 2000
EXSFRES	DS	CL1	Reserved for future
EXSFCEND	DS	0F	
EXSFLEN	EQU	*-&NAME	Length of FILE record
EXSFLV13	EQU	EXSFLEN-(EXSFCEND-EXSFDAXD-76)	cmslevel 13 length
EXSFCNVL	EQU	EXSFLEN-(EXSFCEND-EXSFDLCD)	
EXSFTDFL	EQU	EXSFLEN-(EXSFCEND-EXSFDIRL)	
EXSFCDTL	EQU	EXSFLEN-(EXSFCEND-EXSFMAXB)	

```

EXSFDLRL EQU   EXSFLEN-(EXSFCEND-EXSFCDTD)
EXSFLEN6 EQU   EXSFLEN-(EXSFCEND-EXSFDLRD)
*
*
*       Exist record for DIR
*
*
*       AIF   ('&INTENT' EQ 'ALL').DIRMAP
*       AGO   .DIREND
.DIRMAP ANOP
        ORG   EXSDATA
EXSDIR   DS   0F           Exist for DIR
EXSDDIRL DS   X           Length of Directory ID
EXSDDIRD DS   CL153       Directory ID
EXSDRATH DS   CL1         Read Authority
EXSDRYES EQU   C'1'       Read Authority exists
EXSDRNO  EQU   C'0'       No Read Authority
EXSDWATH DS   CL1         Write Authority
EXSDWYES EQU   C'1'       Write Authority exists
EXSDWNO  EQU   C'0'       No Write Authority
EXSDPROT DS   CL1         External protection
EXSDPYES EQU   C'1'       External protection exists
EXSDPNO  EQU   C'0'       No External protection
EXSDDRAT DS   CL1         Directory Read Authority
EXSDDWAT DS   CL1         Directory Write Authority
EXSDATTR DS   CL1         Directory Attribute
EXSDNRAT DS   CL1         Directory NEWREAD Authority
EXSDNWAT DS   CL1         Directory NEWWRITE Authority
        ORG   EXSDDRAT
EXSDMCDS DS   CL5
EXSDDRA1 DS   CL8         DRA field 1
EXSDDRA2 DS   CL8         DRA field 2
EXSDDRA3 DS   CL8         DRA field 3
        ORG   EXSDDRA1
EXSDDRAS DS   CL24       DRA values
EXSDUNQD DS   CL16       Unique Id
EXSDDLCD DS   XL3         DOLC           (decimal yymmdd)
        DS   XL1         Reserved
EXSDTLCD DS   XL3         TOLC           (decimal hhmmss)
        DS   XL1         Reserved
EXSDDLCC DS   CL8         DOLC           (char yy/mm/dd)
EXSDTLCC DS   CL8         TOLC           (char hh:mm:ss)
        ORG   EXSDDLCD
EXSDDTLC DS   CL24       DTOLC extended attributes
EXSDCDTD DS   XL3         Creation Date (decimal yymmdd)
        DS   XL1         Reserved
EXSDCTMD DS   XL3         Creation Time (decimal hhmmss)
        DS   XL1         Reserved
EXSDCDTC DS   CL8         Creation Date (char yy/mm/dd)
EXSDCTMC DS   CL8         Creation Time (char hh:mm:ss)
        ORG   EXSDCTMD
EXSDDTOC DS   CL24       DTOLC extended attributes
EXSDDCXD DS   XL4         DOLC (decimal yyyymmdd)
EXSDDCXC DS   CL10        DOLC (character yyyy/mm/dd)
EXSDDCXI DS   CL10        DOLC (character yyyy-mm-dd)
EXSDCDXD DS   XL4         Creation Date (decimal yyyymmdd)
EXSDCDXC DS   CL10        Creation Date (char yyyy/mm/dd)
EXSDCDXI DS   CL10        Creation Date (char yyyy-mm-dd)
        ORG   EXSDDCXD
EXSD2000 DS   CL48       Date Extensions, added cmslvl 13
EXSDRES  DS   CL6         These are for Year 2000
EXSDEND  DS   0F         Reserved for future
EXSDLEN  EQU   *-&DNAME   Length of DIR record
EXSDLV13 EQU   EXSDLEN-(EXSDDEND-EXSDDCXD-52) cmslevel 13 length
EXSDCNVL EQU   EXSDLEN-(EXSDDEND-EXSDDLCD)
EXSDTDFL EQU   EXSDLEN-(EXSDDEND-EXSDUNQD)
EXSDMCDL EQU   EXSDLEN-(EXSDDEND-EXSDDRA1)
EXSDLEN6 EQU   EXSDLEN-(EXSDDEND-EXSDDRAT)

```

EXTUAREA



Purpose

Use the EXTUAREA macro to generate a DSECT for the EXTUAREA control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the EXTUAREA macro expansion is labeled EXTUAREA.

Usage Notes

1. The EXTUAREA macroinstruction expands as follows:

EXTUAREA			
EXTUAREA	DSECT		
EXTUGPRS	DS	16F	General registers at interrupt time
EXTUFRS	DS	4D	Floating point registers at interrupt time
EXTUPSW	DS	1D	External Old PSW at interrupt time
EXTUSAVE	DS	20F	Save area for handler routine; pointed to
*			by R13 when control passed to handler rtn.
	DS	0F	Need fullword boundary
EXTUINT	DS	0XL8	Length of 8
EXTUCPID	DS	H	2 bytes = CPU ID bytes
EXTUCODE	DS	H	2 bytes = external interrupt code
EXTUPARM	DS	F	4 bytes = external interrupt parm, for
*			service signal
EXTUPREV	DS	F	Pointer to previous user area
EXTUARS	DS	0F	Access Registers
EXTUAR0	DS	F	Access Register 0
EXTUAR1	DS	F	Access Register 1
EXTUAR2	DS	F	Access Register 2
EXTUAR3	DS	F	Access Register 3
EXTUAR4	DS	F	Access Register 4
EXTUAR5	DS	F	Access Register 5
EXTUAR6	DS	F	Access Register 6
EXTUAR7	DS	F	Access Register 7
EXTUAR8	DS	F	Access Register 8
EXTUAR9	DS	F	Access Register 9
EXTUAR10	DS	F	Access Register 10
EXTUAR11	DS	F	Access Register 11
EXTUAR12	DS	F	Access Register 12
EXTUAR13	DS	F	Access Register 13
EXTUAR14	DS	F	Access Register 14
EXTUAR15	DS	F	Access Register 15
	DS	F	Reserved for IBM use
EXTUGPRH	DS	16F	64-bit saved entry high regs
	DS	16F	Reserved for IBM use
EXTUSIZE	EQU	(*-EXTUAREA)	Size of EXTUAREA in bytes

EXTXCTL



Purpose

Use the EXTXCTL macroinstruction to resume execution of code that was suspended by a X'2603' external interrupt (page fault initiation) after the X'2603' external interrupt (page fault completion) has occurred.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

Usage Notes

1. The routine using this macro must have a DSECT for NUCON.
2. The calling routine must make sure it calls EXTXCTL with interrupts disabled.
3. Register 1 must point to an area that is mapped by the EXTUAREA macro when EXTXCTL is called. The address in register 1 is treated as a 31-bit address.
4. The general registers, floating-point registers, and access registers are loaded from the EXTUGPRS, EXTUFRS, and EXTUARS fields respectively in the area mapped by the EXTUAREA macro. Upon successful completion, control is transferred by way of the PSW in the EXTUPSW field of the area mapped by EXTUAREA. No error checking is performed on the PSW.
5. A second-level interrupt handler (SLIH) should not use this macro because the first-level interrupt handler (FLIH) will not regain control.

Return Codes

If an error occurs, register 15 contains the following return code:

Code

Meaning

8

The macro was not issued from an XC virtual machine.

FPERROR

►► FPERROR ◄◄

Purpose

Use the FPERROR macro to map the file pool extended error information returned in the *wuerror* parameter of callable services library (CSL) routines.

Usage Notes

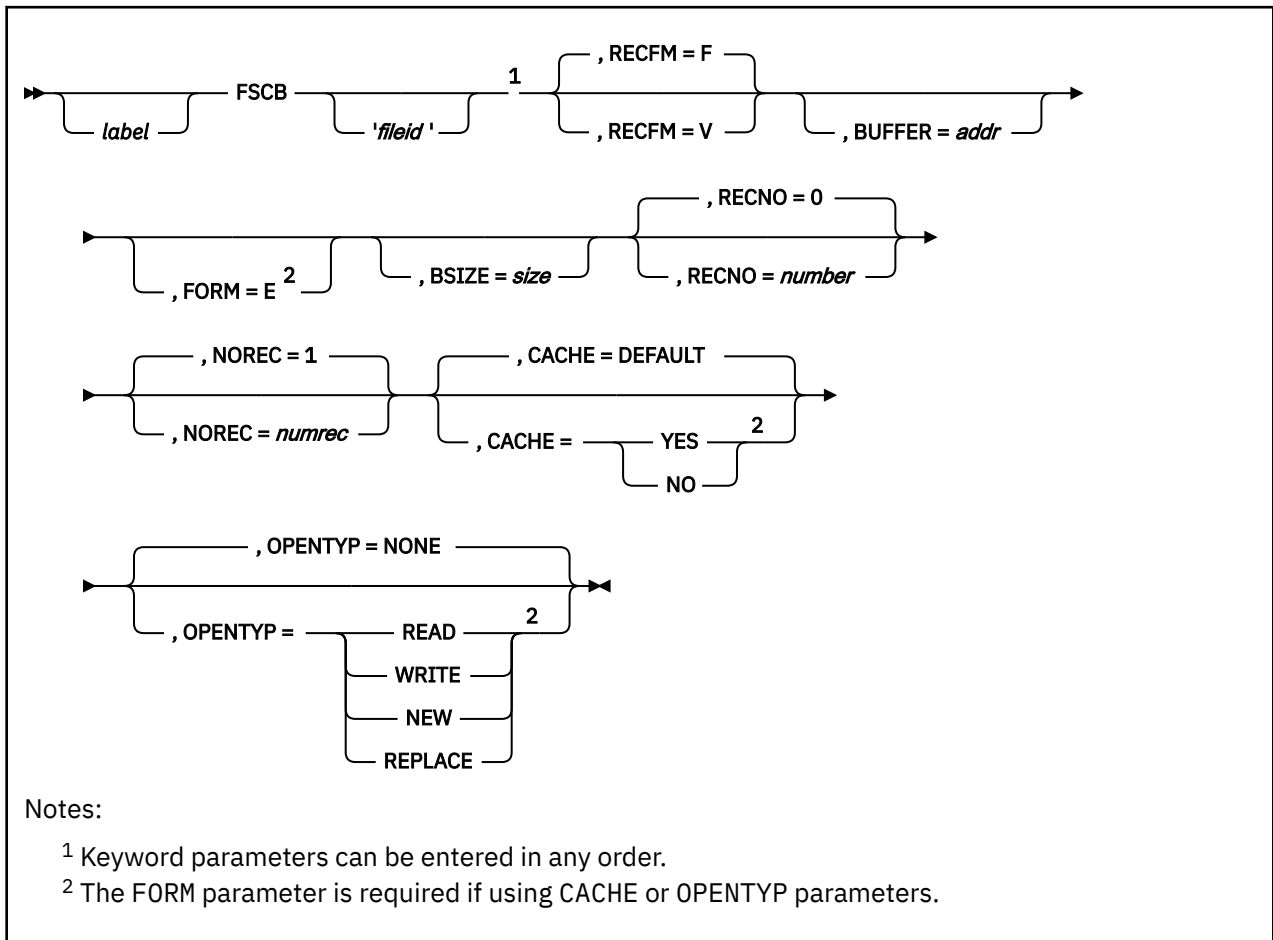
1. The information in the WUERROR and FPERROR buffers can also be accessed as individual variables by using the CSL routine DMSWUERR (Work Unit Error Data Deblocker). This routine is described in [z/VM: CMS Callable Services Reference](#).
2. This macro must be used in conjunction with the WUERROR macro.
3. If the FPEREAS field (error reason code) contains reason code 71800, the FPEAUGMT field will contain the recovery token of the conflicting Coordinated Resource Recovery (CRR) resynchronization activity. Any display of this value should be hexadecimal. If your application is unable to access the required resource it should inform the user that the resource is unavailable and provide the following information for the user to pass on to the file pool administrator:
 - The recovery token (contents of FPEAUGMT field)
 - The file pool ID (contents of the FPEFPOOL field).

For information on CRR, see [z/VM: CMS File Pool Planning, Administration, and Operation](#).

4. If the FPEREAS field (error reason code) contains reason code 50500, the FPEAUGMT field will contain the specific reason for the failure. (50500 means that your request succeeded, but the work unit could not be committed.)
5. If the error was generated by an SFS error during an implicit recall of a file in migrated status (residing in the DFSMS/VM storage repository), the FPEAUGMT field will contain the CSL reason code of the specific request that generated the error, and the FPEDETFP field will contain the file pool ID of the failing resource.
6. File pool extended error information can also be returned for byte file system (BFS) files when CSL routines are used to access them.
7. There can be one or more instances of FPERROR data within the WUERROR data area returned. The FPERROR macroinstruction expands as follows:

FPEFPOOL	DS	CL8	File pool ID
	DS	CL8	Reserved
FPEUWORK	DS	F	Work unit ID
FPEREAS	DS	F	Error reason code
	DS	F	Reserved
	DS	F	Reserved
FPRETC	DS	F	Return code
FPEWRN	DS	16F	Warning reason codes
FPEUSERI	DS	F	User ID index
FPELEVEL	DS	0FL8	
FPERELLV	DS	CL4	FPELEVEL subfield 1
FPECOMLV	DS	CL4	FPELEVEL subfield 2
FPEAUGMT	DS	F	Reason code augmentation field
FPEDETFP	DS	CL8	File pool ID of failing resource
	DS	CL12	Reserved
FPENLEN	EQU	*-FPERROR	Length of FPERROR
FPEDBSZ	EQU	((FPENLEN+7)/8)	Size of FPERROR in doublewords

FSCB



Purpose

Use the FSCB macroinstruction to create a file system control block (FSCB) for a CMS input or output disk file.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement

'fileid'

specifies the CMS file identifier, which must be enclosed in single quotation marks and separated by blanks (*'filename filetype filemode'*). An asterisk (*) is allowed in place of the file name, file type, or file mode. If file mode is omitted, A1 is assumed. See the individual file system (FS) macros to determine the validity of an asterisk in a given position. Before using such an FSCB in another FS macro (such as FSOPEN or FSREAD), you must supply the omitted fields in the file ID at execution time.

RECFM=

specifies the format of the records in the file. Acceptable values are:

F

specifies the fixed-length format (RECFM=F). This is the default value.

V

specifies variable length format (RECFM=V).

BUFFER=addr

specifies the address of the I/O buffer for reading or writing records. This address must be specified as a relocatable expression.

Note: The buffer address is interpreted as a 31-bit field and the high order bit is ignored.

FORM=E

generates an extended format FSCB. An extended format FSCB lets you specify a value up to $(2^{31} - 1)$ for RECNO and NOREC. If you do not specify FORM=E, the RECNO and NOREC values cannot exceed 65535. Specifying FORM=E also results in more efficient code being generated, whether or not you need the larger values for RECNO and NOREC.

The specification of the FORM parameter on FSCB should agree with the specification of the FORM parameter on any FSOPEN, FSREAD, FSWRITE, FSPOINT, or FSSTATE macros which reference this FSCB.

BSIZE=size

specifies the number of bytes to be read or written for each read or write request. The value for *size* must be specified as an absolute expression.

RECNO=number

specifies the record number of the next record to be accessed, relative to the beginning of the file, record 1. The value for *number* must be specified as an absolute expression. The default is 0, which indicates that records are accessed sequentially.

NOREC=numrec

specifies the number of records to be read in the next read operation. The value for *numrec* must be specified as an absolute expression. The default is 1.

CACHE=

indicates whether caching of multiple data blocks is to be performed for this file. This option applies only to SFS files and EDF minidisk files.

The CACHE parameter is pertinent only for explicit opens of the file and cannot be changed on later FSREADs or FSWRITEs by using an FSCB with a different CACHE value specified. The value for CACHE that is in the FSCB when the file is explicitly opened will be used. The file would be explicitly opened by an FSOPEN with OPENTYP of READ, WRITE, NEW, or REPLACE in effect.

To use the CACHE parameter, FORM=E must also be specified. If FORM=E is not specified, or the file is not explicitly opened, then CACHE=DEFAULT is assumed when the file is opened by way of the first FSREAD, FSWRITE, or FSPOINT.

Acceptable values are:

DEFAULT

indicates that the file system should determine whether to cache multiple data blocks, based on the file's characteristics and the actual or anticipated accesses to the file. This is the default value. In most cases, this will be equivalent to CACHE=YES. If your application requires a specific value for the CACHE option, you should specify that value for CACHE rather than rely on the default.

YES

indicates the file system should cache multiple data blocks for the file. When specified, the file system will employ a 'read-ahead' and 'write-behind' method of I/O to the file. This will generally reduce the number of separate I/O operations performed on the file.

NO

indicates that the file system should not cache multiple data blocks.

For more information on the CACHE option, see [z/VM: CMS Application Development Guide for Assembler](#).

OPENTYP=

is the type of open to be performed on the file. OPENTYP is an FSOPEN macro parameter, provided on the FSCB macro for convenience. The OPENTYP parameter on the FSCB will be ignored if no FSOPEN is issued.

Also, this operand cannot be used unless the code containing the FSOPEN macro is reassembled at the VM/SP Release 6 level or above, regardless of whether it was specified on the FSOPEN macro or the FSCB macro. To use the OPENTYP parameter, FORM=E must also be specified. Acceptable values are:

NONE

indicates that the file is not actually opened. The file is implicitly opened when the first FSREAD, FSWRITE, or FSPOINT is issued to the file. This is the default value.

READ

indicates that the file exists and will only be read.

WRITE

indicates that the file may be read or modified. All changed and added records are written. Other records remain unchanged. If the file does not exist, it is created.

NEW

indicates that the file does not exist and is created. It may then be written to or read from. If the file already exists, it is an error and the file is not opened.

REPLACE

indicates that the file is replaced with only the subsequently written records. If the file does not exist, it is created.

Usage Notes

1. To access fields within the FSCB, use the FSCBD macro. Refer to the FSCBD macro description for the layout of the file system control block.
2. IBM recommends that you do not use the same FSCB to reference several different files. If you must, you can override the *fileid* and any of the other options on the FSOPEN, FSWRITE, or FSREAD macroinstructions when you reference a file by way of its FSCB. If, however, you use the FSOPEN macro to open an existing file, CMS resets the BSIZE and RECFM fields in the FSCB to reflect actual file characteristics, not necessarily the characteristics you specify on FSOPEN.

When you use the same FSCB for multiple files, care must be taken to specify the appropriate FSCB options on any other macros that reference the FSCB, particularly when the options differ from file to file. Each time these options are specified on another macro (FSOPEN, FSREAD, FSWRITE) the FSCB is modified. This may lead to an error if a subsequent operation for a different file is issued which allows an option to default to the value present in the FSCB.

For example:

```
FSWRITE 'NEW FILE A1',FSCB=OUTFSCB,RECFM=F,FORM=E
FSWRITE 'OLD FILE A1',FSCB=OUTFSCB,FORM=E
      . . .
OUTFSCB  FSCB  RECFM=V,BUFFER=RECCAREA,BSIZE=80,FORM=E
```

Even though OUTFSCB has RECFM=V specified, the FSWRITE to 'NEW FILE A1' with RECFM=F will change OUTFSCB to contain RECFM=F. The second FSWRITE to 'OLD FILE A1' will assume RECFM=F (not V) because that is the value that is now in the FSCB. Thus, if the file 'OLD FILE A1' on disk is actually RECFM=V, an error will occur on the second FSWRITE, even though the FSCB had specified RECFM=V. To avoid this problem, the preferable approach is to code a separate FSCB for each file which is being used. Otherwise, you must specify the option (in this example, RECFM) on **each** FSWRITE, FSREAD, or FSOPEN which references the same FSCB. For further information, see [z/VM: CMS Application Development Guide for Assembler](#).

3. You can use multiple FSCBs to reference the same file, for example, if you want one FSCB for writing and a different FSCB for reading the file. Remember, the file characteristics are inherent to the file and not to the FSCB. If you establish a read or write pointer using the RECNO option in one FSCB, that pointer remains unchanged unless you specify the RECNO option again on the same or any other FSCB for that file.

FSCBD



Purpose

Use the FSCBD macroinstruction to generate a DSECT for the file system control block (FSCB).

Parameters

Optional Parameter:

label

is an optional assembler label for the statement. The first statement in the FSCBD macro expansion is labeled FSCBD.

Usage Notes

1. The FSCBD macroinstruction expands as follows:

```

FSCBD      FSCBD
DSECT
FSCBCOMM  DS      CL8      File system command (e.g. ERASE)
FSCBFILE  DS      CL18     File ID (name, type, and mode)
          ORG      FSCBFILE
FSCBFNFT  DS      CL16     File name and file type
          ORG      FSCBFNFT
FSCBFN    DS      CL8      File name
FSCBFT    DS      CL8      File type
FSCBFM    DS      CL2      File mode (letter and number)
          ORG      FSCBFM
FSCBFML   DS      CL1      File mode letter
FSCBFMN   DS      CL1      File mode number
FSCBITNO  DS      H        Relative record number to be accessed on
                          FSREAD and FSWRITE (applies only to the
                          non-extended FSCB)
FSCBBUFF  DS      A        Address of the input/output buffer for
                          FSREAD and FSWRITE (also used on calls
                          to STATE routines for the FST address)
FSCBSIZE  DS      F        Length (in bytes) of the input/output
                          buffer (also used to return the record
                          length on FSOPEN)
FSCBFV    DS      CL2      Record format and first flag byte
          ORG      FSCBFV
FSCBRECF  DS      CL1      Record format - F or V
FSCBFLG   DS      XL1      First flag byte
*
*          'FSCBFLG' flag byte definition
*
FSCBTHEX  EQU      X'80'    Space threshold exceeded (SFS only)
FSCBITAV  EQU      X'40'    Item available (no longer used)
FSCBEPL   EQU      X'20'    Extended PLIST (FORM=E)
FSCBMSG   EQU      X'10'    MSG=YES on FSSTATE or FSOPEN
FSCBSTW   EQU      X'08'    STATEW specified on FSSTATE
FSCBCACY  EQU      X'04'    CACHE=YES specified
FSCBCACN  EQU      X'02'    CACHE=NO specified
FSCBRCAY  EQU      X'01'    Previous record null (no longer used)
FSCBNOIT  DS      H        Number of records to be accessed on
                          FSREAD and FSWRITE (applies only to the
                          non-extended FSCB)
          ORG      FSCBNOIT Extended format fields defined
                          over non-extended FSCBNOIT
FSCBFLG2  DS      XL1      Second flag byte
*
*          'FSCBFLG2' flag byte definition (FORM=E only)

```

```

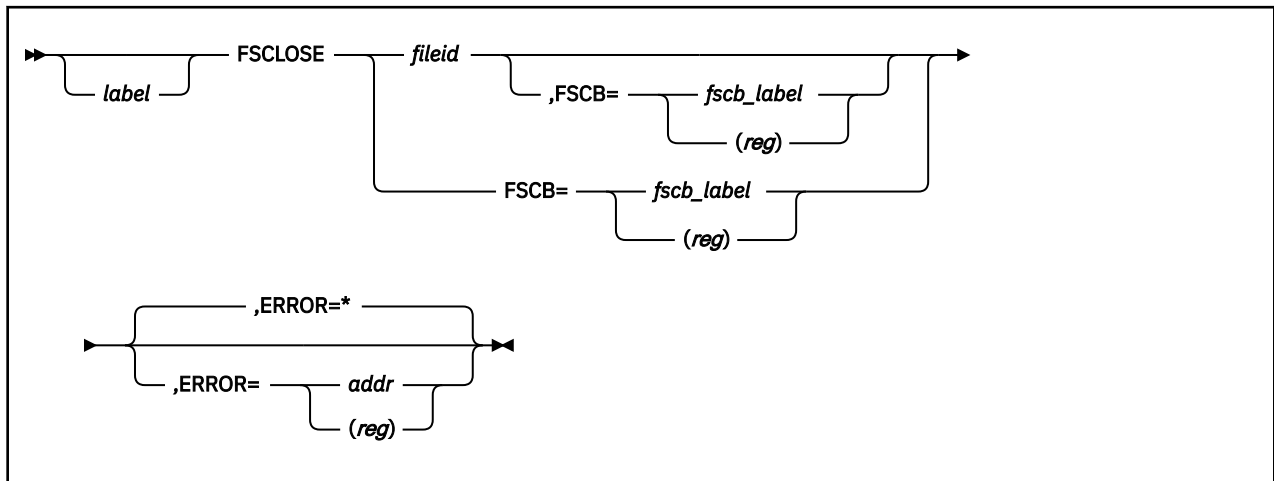
*
FSCBNMAC EQU   X'80'      NOMSG=ACTIVE specified on FSOPEN
FSCBNMNF EQU   X'40'      NOMSG=NOTFOUND specified on FSOPEN
FSCBNMOS EQU   X'20'      NOMSG=OSDOS specified on FSOPEN
FSCBOTYP DS     CL1        OPENTYP value
*
*       'FSCBOTYP' Values (FORM=E only)
*
FSCBTNON EQU   X'00'      OPENTYP=NONE specified
FSCBTRD EQU    C'R'       OPENTYP=READ specified
FSCBTWR EQU    C'W'       OPENTYP=WRITE specified
FSCBTNEW EQU   C'N'       OPENTYP=NEW specified
FSCBTREP EQU   C'X'       OPENTYP=REPLACE specified
FSCBNORD DS     F         Number of bytes actually read on
                          FSREAD
                ORG      FSCBNORD
*
* 'FSCBFST' is returned on FSOPEN. Its value is based on
* the OPENTYP specified and whether the file exists.
* Note that a non-extended format FSCB (FORM=E not specified)
* implies OPENTYP=NONE. The values are as follows:
*
* File doesn't exist .... FSCBFST=A(0)
*
* File exists:
* Not FORM=E ..... FSCBFST=A(Copy of 40 byte FST)
* OPENTYP=NONE ..... FSCBFST=A(Copy of 64 byte FST)
* OPENTYP=READ ..... FSCBFST=A(Copy of 64 byte FST)
* OPENTYP=WRITE ..... FSCBFST=A(Copy of 64 byte FST)
* OPENTYP=REPLACE ..... FSCBFST=A(-1)
* OPENTYP=NEW ..... Error, FSCBFST is unchanged
*
FSCBFST DS     A          Address of a copy of the FST
                          returned on FSOPEN
*
* The following fields apply only to the extended form FSCB
* (for example, FORM=E was specified).
*
FSCBAITN DS     F          Relative record number to be accessed on
                          FSREAD and FSWRITE (also referred to as
                          the "alternate item number")
FSCBANIT DS     F          Number of records to be accessed on
                          FSREAD and FSWRITE (also referred to as
                          the "alternate number of items")
FSCBWPTR DS     F          Extended write pointer (input on FSPOINT
                          FORM=E, output on FSOPEN)
FSCBRPTR DS     F          Extended read pointer (input on FSPOINT
                          FORM=E, output on FSOPEN)
FSCBLNBY EQU    *-FSCBD    Length (in bytes) of the extended FSCB

```

2. You can use the labels in the FSCBD DSECT to access the fields in an FSCB for a particular file. An FSCB is created explicitly by the FSCB macroinstruction, and implicitly by the FSREAD, FSWRITE, and FSOPEN macroinstructions when the FSCB parameter is not specified. Also note that the fields within the FSCB are modified by the macros FSREAD, FSWRITE, FSOPEN, FSCLOSE, FSPOINT, FSSTATE, and FSERASE.
3. When you specify FORM=E on the FSCB macroinstruction (or on FSREAD, FSWRITE, FSPOINT, FSOPEN, or FSSTATE):
 - The fields FSCBAITN and FSCBANIT are used for the RECNO and NOREC macro options instead of FSCBITNO and FSCBNOIT, which are reserved for other purposes.
 - The fields FSCBFLG2, FSCBOTYP, FSCBWPTR, and FSCBRPTR are also used.
 - The FSCBEPL flag is turned on (X'20' in FSCBFLG).

You must use FORM=E FSCBs to manipulate files larger than 65,535 items.

FSCLOSE



Purpose

Use the FSCLOSE macroinstruction to close an open file. FSCLOSE can only be used to close files opened by other file system macros. You cannot use FSCLOSE to close a file opened by the DMSOPEN routine.

Parameters

Required Parameters:

fileid

specifies the CMS file identifier. It may be:

'fn ft fm'

file ID enclosed in single quotation marks and separated by blanks. An asterisk (*) may be specified for *fn*, *ft*, or *fm*, or any combination, indicating all file names, file types or file modes respectively. If *fm* is omitted, A1 is assumed.

(reg)

a register, other than 0 or 1, containing the address of the file ID (18 characters). When you specify *(reg)*, the file ID must be exactly 18 characters in length; 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be padded with blanks. If the file mode is left blank, it is treated the same as an asterisk (*), meaning all file mode occurrences of the specified file name and file type.

FSCB=

specifies the address of an FSCB. Acceptable values are:

fscb_label

specifies the label on the FSCB macroinstruction.

(reg)

specifies a register containing the address of an FSCB.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

You can specify any general register other than 0, 1, or 15.

Usage Notes

1. Within your application, you should close any files that you open, whether the files were opened explicitly by FSOPEN or implicitly by FSREAD, FSWRITE, or FSPOINT. If you do not close files that you have opened, CMS will close them at end-of-command (Ready ;). However, if your application is called from another application program (for example, from an EXEC), failure to close your files can lead to problems, such as:
 - Delayed commit of data. FSCLOSE cannot commit updates to the shared file system on a work unit which has files open, nor can it commit updates to a minidisk if there are files open for output on the given minidisk.
 - Incorrect records being read or written. The read and write pointers for a file are initialized when a file is opened. Each time you read or write a file, the corresponding pointer gets adjusted. Thus, if the application which calls your program uses sequential processing and references the same files as your program, failure to close your files can lead to unpredictable results because of the read and write pointers not being re-initialized. This could occur when the calling application follows your program with an FSREAD, FSWRITE, or EXECIO using sequential processing, expecting the operation to implicitly open the file. The read and write pointers, however, would remain where your program left them, rather than being set to the beginning and end of the file, respectively, by an open.
2. If you code *fileid* and the FSCB parameter, CMS uses the *fileid* to fill in the FSCB.
3. If you want reentrant code, you must specify the FSCB parameter.
4. Even though an FSCLOSE macro is issued for a file on a CMS minidisk, the directory cannot be updated on disk as long as other files are open for output on that disk.
5. When using FSCLOSE to close a file residing in the Shared File System, changes are committed based on the default work unit that was in effect when the file was opened. However, the commit will not be performed unless you are closing one of the following:
 - The last file open for output on a work unit
 - The last file open on a work unit.

This applies only to files that have been opened through macros or non-SFS statements (for example: CMS FS macros, EXECIO command). The commit performed by FSCLOSE is a coordinated commit, meaning that all changes to protected resources on the work unit are committed in unison (or rolled back if any of the resources cannot commit). If the program using FSCLOSE to close an SFS file has a protected conversation with another application that has open SFS files on the work unit associated with the conversation, the other application's open files do not prevent the commit. Changes to those files are also committed. This applies regardless of how the other application opened its SFS files.
6. When an open minidisk file cannot be successfully closed and that file is in an inconsistent state, a message is issued and CMS is terminated. For SFS files, when the close is unsuccessful:
 - If an implicit rollback occurred because of the nature of the failure, a message is issued to that effect.
 - If an implicit rollback did not occur, FSCLOSE initiates a rollback for the work unit on which the file was opened. A message is issued reflecting the error.
 - If FSCLOSE initiates a rollback but the rollback fails, a message is issued and CMS is terminated.

7. For a file open for output, the FSCBTHEX (X'80') indicator bit of the FSCBFLG byte indicates when you have reached your SFS file space threshold. (For more information on the SFS file space threshold, see the SET THRESHOLD command in *z/VM: CMS Commands and Utilities Reference*.) Because the CMS portion of the file system does buffering, you will only see the indicator when it is necessary to write the buffers to the file pool. This can occur during a read, write, or close. For small files, the indicator might not be returned until the close.

Return Codes

Register 15 contains the following return codes:

Code

Meaning

0

One or more files closed successfully and/or one or more files opened using the DMSOPEN or DMSOPDBK interfaces were committed to disk.

6

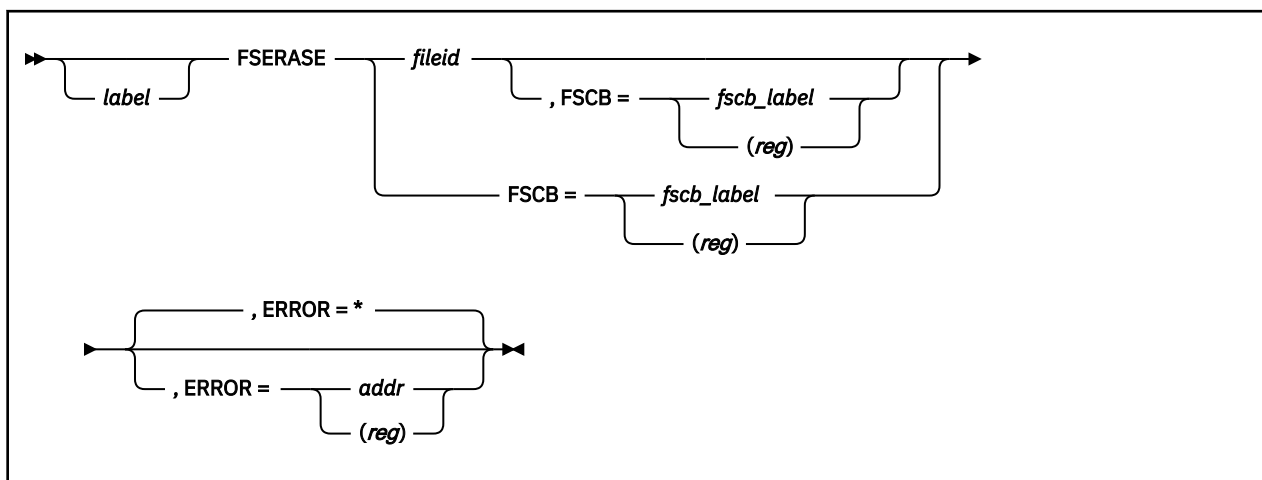
No open files matching the input file ID were found or invalid file ID (fn ft fm) specified.

31

Close failed for one or more SFS files. Rollback performed on each affected work unit.

An application error, system error, or lack of required resource can be the cause of this return code. If the error persists, refer to the *z/VM: Diagnosis Guide* for more information about diagnosing the problem.

FSERASE



Purpose

Use the FSERASE macroinstruction to delete a CMS file from a minidisk or SFS directory. FSERASE cannot be used to erase directories.

To erase a file in another user's directory, you must have write authority to both the directory and the file and you must have the directory accessed in read/write status. (Use the FORCERW option of the ACCESS command to access another user's directory in read/write status.)

Parameters

Required Parameters:

fileid

specifies the CMS file identifier. Acceptable values are:

'fn ft fm'

file ID enclosed in single quotation marks and separated by blanks. An asterisk (*) may be specified for *fn*, *ft*, or *fm* (*fn ft* cannot both be *, unless the file mode letter **and** file mode number are both specified for file mode), indicating all file names, file types, or file modes respectively. If *fm* is omitted, A1 is assumed.

(reg)

a register, other than 0 or 1, containing the address of the file ID (18 characters). When you specify *(reg)*, the file ID must be exactly 18 characters in length: 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be filled with blanks. If the file mode is left blank, only the A disk is searched.

FSCB=

specifies the address of an FSCB. Acceptable values are:

fscb_label

specifies the label of an FSCB macroinstruction.

(reg)

specifies a register containing the address of an FSCB.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

You can specify any general register other than 0, 1, or 15.

Usage Notes

1. On return from the FSERASE macro, register 1 points to a parameter list. The second doubleword contains the file name; the third doubleword contains the file type; and the next halfword contains the file mode of the file.
2. If you code both *fileid* and the FSCB parameter, CMS uses the *fileid* to fill in the FSCB.
3. When *fileid* refers to an SFS alias, only the alias is erased. The base file remains intact. When *fileid* refers to an SFS base file, all authorities and aliases to that file are dropped. If the file is later recreated, none of the previous authorities or aliases will apply to the new file.

Return Codes

Register 15 contains one of the following return codes:

Code**Meaning**

20

Invalid character in file name or file type.

24

Invalid file mode.

25

Insufficient storage available.

28

File not found.

31

Erase failed. Rollback Performed.

36

Disk or directory is not accessed or is accessed read only.

40

One of the following errors occurred:

- A required CSL routine was dropped.
- A required CSL routine was not loaded.
- There was an error in a user exit routine.
- There was an error calling the user accounting exit routine (DMS2AB).

55

APPC/VM error.

70

SFS file sharing conflict or minidisk file is already open by DMSOPEN or DMSOPDBK

99

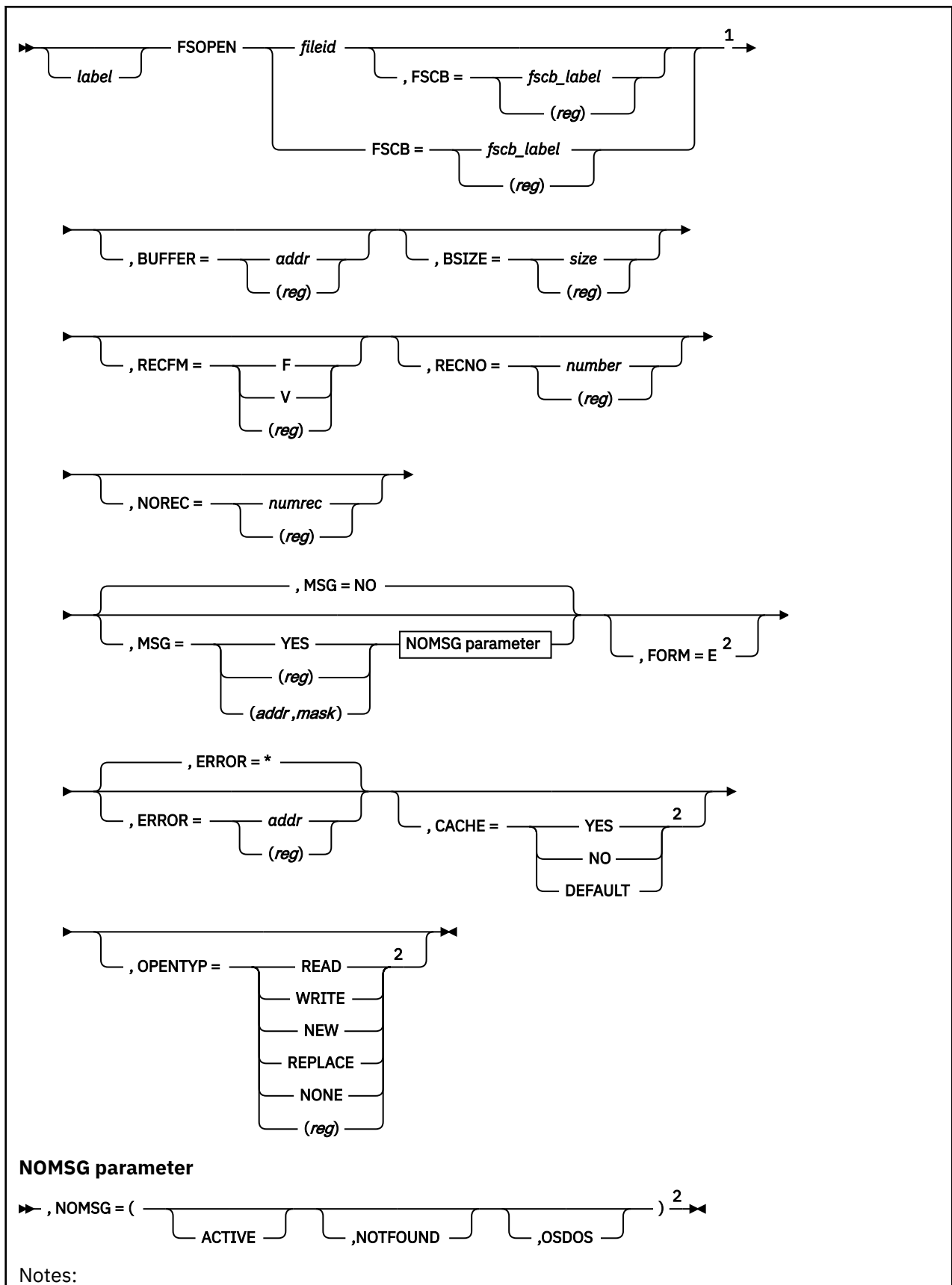
A required system resource is unavailable for one of the following reasons:

- There is insufficient virtual storage for the file pool.
- The file pool server is unavailable.

104

Supervisor or file pool supervisor error.

FSOPEN



¹ Keyword parameters can be entered in any order.

² The FORM parameter is required if using the NOMSG,CACHE, or the OPENTYP parameters.

Purpose

Use the FSOPEN macro to open a file for input or output.

Parameters

Required Parameters:

fileid

specifies the CMS file identifier.

'fn ft fm'

identifies the file ID enclosed in single quotation marks and separated by blanks. If *fm* is omitted, A1 is assumed.

(reg)

identifies the register (other than 0 or 1) containing the address of the file ID. The file ID must be exactly 18 characters in length; 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be padded with blanks.

An asterisk (*) is not allowed for the file name or file type. When the *fileid* specified has a file mode of blank or *, the file with the specified file name and file type on the first accessed mode (in alphabetic order) will be opened. If no file is found to match, the open will fail, regardless of the value of OPENTYP. Note that for OPENTYP=NEW, a file mode of blank or * is invalid. Also, when OPENTYP=NEW, REPLACE, or WRITE only the file modes accessed as Read/Write will be checked for a match.

For OPENTYP=READ, WRITE, or REPLACE, the file mode where the match occurs will be returned in the *fileid* in the FSCB. For OPENTYP=READ (with a file mode of blank or *), if the match occurs on a read-only extension of another file mode, the file mode that is returned is the file mode of the parent disk, and not the file mode of the actual disk or directory containing the file.

The file mode of blank defaults to 'A' when 'fn ft' (omitting file mode) is used to specify a file ID. This is passed through the file system and subsequently treated as an asterisk (*) only when:

The file mode number in *fileid* (whether specified on FSOPEN or on the corresponding FSCB) is assigned to a new file. The rules for determining the file mode number are as follows:

For a file which did not previously exist:

- When specified (a number, 0-6), it is used for OPENTYP=NEW, WRITE, or REPLACE.
- When omitted (blank), it defaults to 1 for OPENTYP=NEW, WRITE, or REPLACE.
- For OPENTYP=READ or NONE, it is an error if the file does not exist, and the specified file mode number is not used.

For a file which did previously exist:

- When specified (a number, 0-6), it is for OPENTYP=REPLACE; otherwise, it is ignored.
- When omitted (blank), it defaults to the previous file mode number of the replaced file for OPENTYP=REPLACE.
- For OPENTYP=READ, WRITE, or NONE, the specified value is not used. The file mode number of the existing file remains in effect.
- For OPENTYP=NEW, it is an error if the file exists, and the specified file mode number is not used.

Regardless of whether the specified file mode number is used, or whether the file exists, the specified file mode number must be valid, or an error occurs. Valid file mode numbers are 0-6 and blank (omitted). If the file mode is specified as *, the file mode number must be blank.

FSCB=

specifies the address of an FSCB. Acceptable values are:

fscb_label

specifies the label on an FSCB macroinstruction.

(*reg*)

specifies a register that contains the address of an FSCB.

Note: The referenced FSCB must have the same specification for FORM as this FSOPEN.

Optional Parameters:

label

is an optional assembler label for the statement.

BUFFER=

specifies the address of the I/O buffer for reading or writing records. Acceptable values are:

addr

specifies the address of the I/O buffer as a relocatable expression.

(*reg*)

specifies the register (other than register 1) containing the address of the I/O buffer.

Note: buffer address is interpreted as a 31-bit field and the high order bit is ignored.

BSIZE=

specifies the number of bytes to be read or written for each read or write request. Acceptable values are:

size

specifies the number of bytes to be read or written as an absolute expression.

(*reg*)

specifies the register (other than register 1) containing the number of bytes to be read or written.

RECFM=

specifies the format of the records in the file. Acceptable values are:

F

specifies the fixed-length format (RECFM=F). If omitted, RECFM assumes the value specified in the FSCB. This is the default value if FSCB is not specified.

V

specifies variable length format (RECFM=V).

(*reg*)

specifies the register (other than register 1) whose low-order byte contains the record format (C'F' or C'V').

RECNO=

specifies the record number of the next record to be accessed, relative to the beginning of the file (record 1). If FSCB is not specified, the default is 0, which indicates that the next sequential record is accessed. Acceptable values are:

number

specifies the record number as an absolute expression.

(*reg*)

specifies the register (other than register 1) containing the record number.

NOREC=

specifies the number of records to be read in the next read operation. If FSCB is not specified, the default is 1, which is also the only valid value for files with variable-length records. Acceptable values are:

numrec

specifies the number of records as an absolute expression.

(reg)

specifies the register (other than register 1) containing the number of records.

MSG=

indicates whether an error message is displayed if an error occurs. Acceptable values are:

NO

means no messages will be issued. NO is the default. If OPENTYP=NONE, MSG=NO is treated the same as with FSSTATE.

YES

means all messages will be issued (except for those suppressed by the NOMSG option).

(reg)

the macro checks the value of the specified register (other than register 1) and, if it is 0, sets MSG to NO. If the register contains a nonzero value, the macro sets MSG to YES.

(addr,mask)

defines a single bit in storage that sets the value of the MSG parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then MSG is set to NO. If the bit is 1, then MSG is set to YES. For example, to test the first bit in the single byte of storage at location MSGFLAG, specify the MSG parameter as

```
MSG=(MSGFLAG,X'80')
```

To set the value of the MSG parameter at assembly time, specify MSG=YES or MSG=NO. To set the value at execution time, specify MSG=(reg) or MSG=(addr,mask).

NOMSG=

indicates that although MSG=YES is in effect, error messages are to be suppressed in one or more cases. This allows the MSG=YES parameter to be used so FSOPEN can issue error messages except for the case(s) where failure to open the file is not considered an error. If MSG=YES is not in effect, this parameter has no meaning and is ignored. FORM=E is required when specifying this parameter. NOMSG does not apply when OPENTYP=NONE. The acceptable values are:

ACTIVE

indicates that a message should not be issued if you already have the file open by an FSOPEN, FSREAD, FSWRITE or FSPOINT (or EXECIO command). A return code of 37 is returned in this situation, whether or not a message is issued.

NOTFOUND

indicates that a message should not be issued when trying to open a file that does not exist or when trying to open a file that you do not have authority for. A return code of 28 is returned in this situation, whether or not a message is issued.

OSDOS

indicates that a message should not be issued when trying to open a file on an OS or DOS formatted disk. A return code of 84 is returned in this situation, whether or not a message is issued.

You cannot read from or write to an OS or DOS disk using the FS macros. Return codes 80 through 83 additionally imply that the file is not accessible through CMS OS or DOS simulation access methods.

ACTIVE, NOTFOUND, or OSDOS may be specified individually or they may be specified in combination. The order of specification is of no importance as long as the parameters are separated by a comma and enclosed in parentheses when combined. The (*address,mask*) and (*reg*) formats are not supported.

NOMSG overrides MSG=YES in the following cases:

- ACTIVE
 - file was already active through the FS macro interface and

- return code was 37 and
- OPENTYP was READ, WRITE, NEW or REPLACE
- NOTFOUND
 - file was not found or not authorized and
 - return code was 28 and
 - OPENTYP was
 - READ or
 - WRITE or REPLACE and file mode was *

Note: NOMSG=NOTFOUND does not apply when:

- file was not authorized and
- return code was 28 and
- OPENTYP was WRITE or REPLACE and
- file mode was a letter
- OSDOS
 - file is on an OS or DOS disk and
 - return code was 84 and
 - OPENTYP was READ

FORM=E

must be specified when the extended format is being used. An extended format FSCB lets you specify a value up to $2^{31} - 1$ for RECNO and NOREC. If you do not specify FORM=E, the RECNO and NOREC values cannot exceed 65535. Specifying FORM=E also results in more efficient code being generated, whether or not you need the larger values for RECNO and NOREC.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

You can specify any general register other than 0, 1, or 15.

CACHE=

indicates whether caching of multiple data blocks is to be performed for this file. This option applies only to SFS files and EDF minidisk files.

The CACHE parameter is pertinent only for explicit opens of the file. CACHE is ignored if OPENTYPE=NONE. It cannot be changed on later FSREADs or FSWRITEs by using an FSCB with a different CACHE value specified. The value for CACHE that is in the FSCB when the file is explicitly opened will be used. The file would be explicitly opened by an FSOPEN with OPENTYP=READ, WRITE, NEW, or REPLACE in effect.

To use the CACHE parameter, FORM=E must also be specified. If FORM=E is not specified, or the file is not explicitly opened, then CACHE=DEFAULT is assumed when the file is opened by way of the first FSREAD, FSWRITE, or FSPOINT.

Acceptable values are:

YES

indicates the file system should cache multiple data blocks for the file. When specified, the file system will employ a 'read-ahead' and 'write-behind' method of I/O to the file. This will generally reduce the number of separate I/O operations performed on the file.

NO

indicates that the file system should not cache multiple data blocks.

DEFAULT

indicates that the file system should determine whether to cache multiple data blocks, based on the file's characteristics and the actual or anticipated accesses to the file. In most cases, this will be equivalent to CACHE=YES. This is the default value when FSCB is not coded. If an FSCB was coded, the value specified in it is used.

Under some conditions, the file system does not cache multiple data blocks for a file, even when CACHE=YES. For example, 'read-ahead' may not be done if the caller's request is completely satisfied by reading all data directly into the caller's buffer.

For more information on the CACHE option, see [z/VM: CMS Application Development Guide for Assembler](#).

OPENTYP=

is the type of open to be performed on the file. Acceptable values are:

READ

indicates that the file exists and will only be read.

WRITE

indicates that the file may be written to or read from. All changed and added records are written. Other records remain unchanged. If the file does not exist, it is created.

NEW

indicates that the file does not exist and is then created. It may then be written to or read from. If the file already exists, it is an error and the file is not opened.

REPLACE

indicates that the file is replaced with only the subsequently written records. If the file does not exist, it is created.

NONE

indicates that the file is not actually opened. The file is implicitly opened when the first FSREAD, FSWRITE, or FSPOINT is issued to the file. Use of FSOPEN with OPENTYP=NONE is essentially equivalent to an FSSTATE, and differs from an FSSTATE in that it may be used to create an FSCB for the file. The CACHE parameter is ignored if OPENTYPE=NONE.

(reg)

indicates the register (other than register 1) whose low-order byte contains the OPENTYP value, as defined in the FSCBD macro:

Field**Value****FSCBTNON**

X'00' (OPENTYP=NONE)

FSCBTRD

C'R' (OPENTYP=READ)

FSCBTWR

C'W' (OPENTYP=WRITE)

FSCBTNEW

C'N' (OPENTYP=NEW)

FSCBTREP

C'X' (OPENTYP=REPLACE)

If omitted, OPENTYP assumes the value specified in the FSCB. If no FSCB parameter is coded, the default value for OPENTYP is NONE. If OPENTYP=WRITE, REPLACE, or NEW is specified, the file mode

specified in the *fileid* must be accessed as Read/Write. To use OPENTYP=READ, WRITE, NEW, or REPLACE, FORM=E must also be specified.

Usage Notes

1. On return from the FSOPEN macro, register 1 contains the address of the FSCB for the file. If you did not specify FSCB on the FSOPEN macro, one is created for you as part of the generated code. Thus, you can use FSOPEN to create an FSCB for a file.
2. If you use FSOPEN on an existing file, the BSIZE, RECFM, and file mode fields in the FSCB are set to reflect the actual file characteristics (note that BSIZE is set to the logical record length).
3. If you code both *fileid* and the FSCB parameter, CMS uses *fileid* to fill in the FSCB. This applies to the BUFFER, BSIZE, CACHE, FORM, OPENTYP, RECFM, RECNO, and NOREC parameters as well.
4. If you want reentrant code, you must specify the FSCB parameter.
5. If a file is opened for NEW or REPLACE, or is new and opened for WRITE, an FSREAD issued before any FSWRITE to the file returns an *end of file* return code.
6. If a new file is being created and is closed before an FSWRITE is issued, the file will not exist after it is closed.
7. On return from FSOPEN, FSCBFST in the FSCB is updated as follows:
 - For a successful open of an existing file:
 - For OPENTYP=READ, WRITE, or NONE (note that a nonextended form FSCB implies an OPENTYP=NONE), FSCBFST will contain the address of a copy of the file's FST.
 - For OPENTYP=REPLACE, FSCBFST=-1.
 - For OPENTYP=NEW, it is an error if the file exists, and the contents of FSCBFST are not modified.
 - For a successful open of a new file:
 - For OPENTYP=NEW, WRITE, or REPLACE, FSCBFST=0.
 - For OPENTYP=READ or NONE (or if a nonextended form FSCB is used), it is an error if the file does not exist, and FSCBFST remains unchanged.
 - For RC=37, FSCBFST will contain the address of a copy of the opened file's FST. If the file is active for write (for example, you previously opened it for NEW, WRITE, or REPLACE with FSOPEN or FSWRITE), the FST will reflect the updates you have made to the file. Note that RC=37 does not occur for OPENTYP=NONE. The logical record length field in the FST may not be available on a new file currently open for output. This may be detected by a number of records of zero.

When a copy of the FST is returned, it is up to the caller to extract any information from this FST copy immediately, as any use of the file system will potentially change its contents. Use the FSTD macro to map the information in the FST. The format of the FST returned depends on whether FORM=E was specified, and is identical to that returned by FSSTATE. See FSSTATE for a description of the FST and see FSCBD for a description of the FSCB.

8. FSOPEN will produce different results for files in the Shared File System and files on minidisks. FSOPEN for REPLACE causes immediate erasure of a minidisk file. However, if the file resides on an SFS directory, the old version of the file will be available until an FSWRITE has occurred and updates on that work unit are subsequently committed. For example, an FSOPEN for REPLACE followed immediately by an FSCLOSE to that same file will create different results depending upon the location of that file. If the specified file resides on a:
 - minidisk—the file will be erased
 - SFS directory—the original version will be unaltered.

Prior to Release 2.1, applications that used FSOPEN for REPLACE for an SFS file were allowed to continue to write records even when the SFS file space limit was exceeded. The attempt to commit when the file space limit was exceeded would result in a rollback of all the changes.

When writing to a file in a Release 2.1 or above SFS file pool server, the behavior of FSWRITE has changed. FSWRITE will return an error when it detects that the file space limit is reached. An attempt

to commit at that point will commit all changes. If you want your application to restore the original file when using FSOPEN for REPLACE of an SFS file, you will need to issue a rollback request.

9. An SFS file can be opened only once for output at any given time. Hence, an attempt to process a file with FSOPEN OPENTYP=WRITE, REPLACE or NEW when it is already opened for WRITE, REPLACE, or NEW will fail regardless of whether it was previously opened with the FSOPEN macro, FSWRITE macro, or the DMSOPEN callable services library (CSL) routine. Open for output will fail even if another user has the file opened for output.
10. A minidisk file opened for output cannot be open for input simultaneously. Hence, an attempt to process a file with FSOPEN OPENTYP=WRITE, REPLACE or NEW will fail when it has been opened for any intent, regardless of whether it was previously opened by FSOPEN, FSREAD, FSWRITE or FSPOINT macros, or the DMSOPEN or DMSOPDBK CSL routines. Any attempt to process a minidisk file with FSOPEN will fail if it is already open for output by any CMS file system service.
11. For SFS files, the file is opened using the current default work unit ID.
12. The 'update-in-place' facility lets you write blocks back to their previous location on disk. For files on minidisks, the 'update-in-place' attribute is indicated by a file mode number of 6.



Attention: Neither the integrity of the file nor of the disk on which it resides is guaranteed when updating an existing file mode number 6 on a minidisk. For details, see 'EDF Data Integrity' in *z/VM: CMS Application Development Guide for Assembler*. For SFS files, file mode number 6 is treated the same as file mode number 1. 'Update-in-place' on SFS files is achieved by specifying the overwrite attribute as INPLACE. For details, see 'Overwrite Attribute' in *z/VM: CMS Application Development Guide*.

Note: For a variable format file, 'update-in-place' applies only if a record is replaced by a record with the same length.

13. When opening and updating an SFS file with FSOPEN:
 - In general, you cannot see another user's uncommitted changes. However, as a reader of an update-in-place file, this is not necessarily true. It is possible that a reader may see a writer's updates without either one closing the file or committing the data. Note that because CMS buffers the file's data, the writer's updates must first be written to DASD and the reader's buffers must then be read from DASD before the reader sees these updates. However, the timing of when the buffers get read or written is highly dependent on the file size, the caching options specified when the file was opened, and the record access patterns of both the reader and the writer.
 - Although you generally cannot see another user's uncommitted changes you can see your own uncommitted changes on the same work unit. If you update and close a file and then reopen it on the same work unit, you will see the updated version. If you do not close the file and open it a second time, or if you open it on another work unit, you will only see the last committed version of the file.
14. A single user cannot open a given *fileid* more than once using the file system macros (FSOPEN, FSWRITE, FSREAD, FSPOINT). Thus, an attempt to open a file with FSOPEN OPENTYP=READ, WRITE, NEW or REPLACE is an error if the file is currently open as a result of a previous:
 - FSOPEN with OPENTYP=READ, WRITE, NEW, or REPLACE
 - FSREAD, FSWRITE, or FSPOINT.

In these cases, it would be necessary to close the file with FSCLOSE prior to reopening it. Note that an earlier FSOPEN with OPENTYP=NONE specified does not actually open the file, and thus would not cause an error on the subsequent FSOPEN with OPENTYP=READ, WRITE, NEW, or REPLACE specified. Likewise, an FSOPEN with OPENTYP=NONE specified would not result in an error if the file was already opened.

15. For OPENTYP=NEW or REPLACE, or when creating a new file with OPENTYP=WRITE, the RECFM in the FSCB establishes the record format of the file at time of FSOPEN. For an existing file on OPENTYP=READ, WRITE, or NONE, the RECFM in the FSCB is updated to reflect the actual file characteristics.

Note that for OPENTYP=WRITE, NEW, or REPLACE, the RECFM in the FSCB must contain an F or V (even for OPENTYP=WRITE of an existing file). For OPENTYP=READ or NONE, the value is ignored.

16. For OPENTYP=READ, WRITE, NEW or REPLACE, the read and write pointers in the FSCBRPTR (extended read pointer) and FSCBWPTR (extended write pointer) will be returned in the FSCB field, even when the return code is 37. This information is not returned for OPENTYP=NONE, or when the FORM=E is omitted (which implies OPENTYP=NONE).
17. For return code 37 (meaning the file has already been opened through the file system macro interface) the FST will be updated with return information even though the open was not performed. The file mode letter, file mode number, record format, logical record length, read and write pointers, and address of a copy of the FST are all returned.
18. By default, SFS files created by FSOPEN have the RECOVER and NOTINPLACE attributes. To override these defaults on FSOPEN, you must use the DMSPUSHA (SFS Push Attributes) CSL routine to set the default recoverability and overwrite attributes you want for a specific file mode number. For information about DMSPUSHA, see [z/VM: CMS Callable Services Reference](#).

Return Codes

Register 15 contains one of the following return codes:

Code

Meaning

0

Open was successful.

3

Failing I/O operation to an existing minidisk file for OPENTYP=READ or WRITE.

7

The file has an invalid record format.

11

Invalid RECFM specified (neither F nor V) for OPENTYP=WRITE, NEW, or REPLACE.

12

Disk or directory not accessed R/W for OPENTYP=WRITE, NEW, or REPLACE.

20

Invalid character in file name or file type.

24

Invalid file mode. Allowable file modes are any alphabetic character, blank, or *, except that blank and * are not allowed for OPENTYP=NEW. When file mode is alphabetic, an optional file mode number (0-6) may also be specified.

25

Insufficient virtual storage available.

28

File not found for one of the following reasons:

- OPENTYP=WRITE or REPLACE with file mode of blank or * specified
- OPENTYP=NONE or READ with any legal file mode
- Insufficient authority for any OPENTYP and legal file mode.

30

Error opening an SFS file (other than listed specifically) but no rollback occurred.

31

Error opening an SFS file, and a rollback has occurred on the current default work unit ID.

An application error, system error, or lack of required resource can be the cause of this return code. If the error persists, refer to the *z/VM: Diagnosis Guide* for more information about diagnosing the problem.

- 33** Invalid OPENTYP specified.
- 34** Invalid CACHE specified.
- 35** File already exists for OPENTYP=NEW.
- 36** Disk or directory not accessed.
- 37** File is already opened through macro interface, and you specified OPENTYP=READ, WRITE, NEW, or REPLACE on this request.
- 40** One of the following errors occurred:
- A required CSL routine was dropped.
 - A required CSL routine was not loaded.
 - There was an error in a user exit routine.
 - There was an error calling the user accounting exit routine (DMS2AB).
 - You are already writing to a different resource such as another SFS file pool, and your environment is not set up for CRR. For example, the CRR Recovery Server is not available, or you are writing to a file in a VM/SP 6 file pool.
- 49** External object cannot be opened.
- 50** File is in DFSMS/VM migrated status and implicit RECALL is set to OFF.
- 51** Error occurred during DFSMS/VM file recall processing.
- 55** APPC/VM error.
- 70** One of the following sharing conflicts occurred:
- The file is locked.
 - A deadlock was detected.
 - The file is open for write through SFS OPEN and OPENTYP of WRITE or REPLACE is specified on this request.
 - The file is open for write by another user and OPENTYP of WRITE or REPLACE is specified on this open.
 - There was an attempt to make uncommitted updates to more than one file pool on a single work unit.
 - The minidisk file is already open by DMSOPEN or DMSOPDBK with an output intent when issuing an FSOPEN for intent of READ.
 - The minidisk file is already open by DMSOPEN or DMSOPDBK when issuing an FSOPEN for intents NEW, WRITE, or REPLACE.
- 80** I/O error accessing OS dataset.
- 81** OS read password protected dataset.
- 82** OS dataset organization is not BSAM, QSAM, or BPAM.

83

OS dataset has more than 16 extents.

84

Attempt to open a file on an OS or DOS formatted minidisk.

88

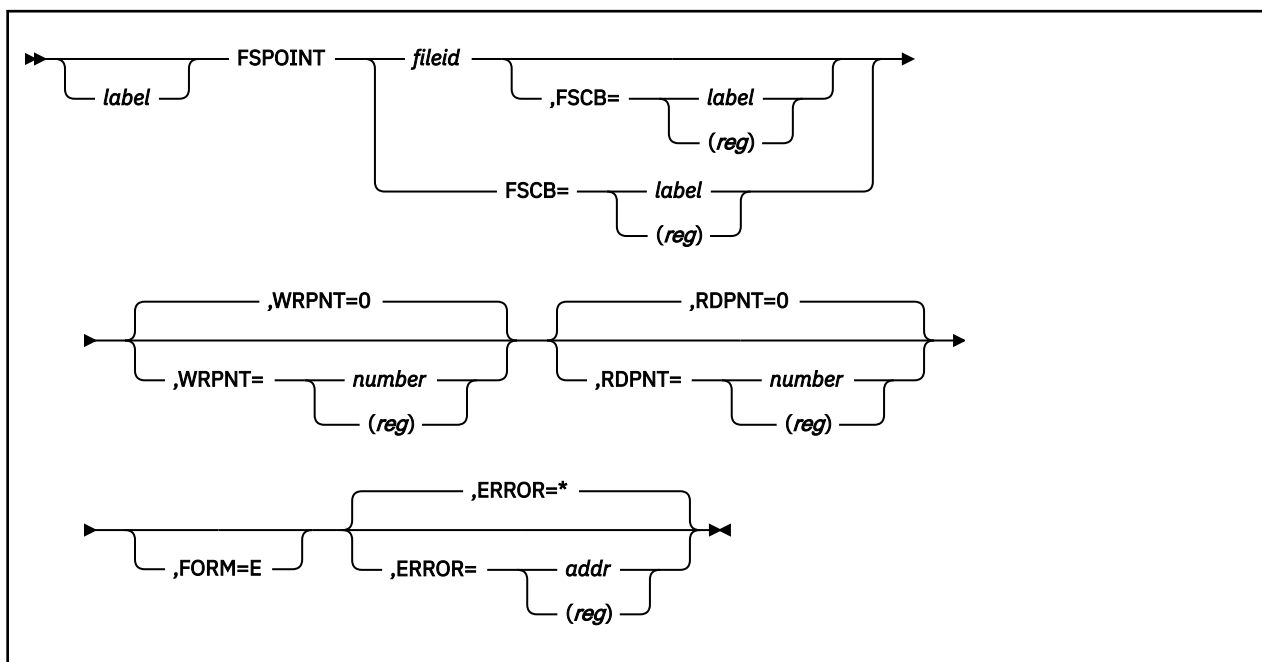
Nonextended format FSCB supplied and a nonextended format copy of the FST cannot be built (number of records or number of data blocks exceeds 65535).

99

A required system resource is unavailable for one of the following reasons:

- There is insufficient virtual storage for the file pool server.
- The file pool server is unavailable.
- File is in migrated status and DFSMS is not enabled.

FSPOINT



Purpose

Use the FSPOINT macroinstruction to reset the write and read pointers for a file. You must have read or write authority on the target file.

Parameters

Required Parameters:

fileid

specifies the CMS file identifier. It cannot be an erased or revoked alias. Acceptable values are:

'fn ft fm'

specifies the file ID enclosed in quotation marks and separated by blanks. If *fm* is omitted, A1 is assumed.

(reg)

specifies a register, other than 0 or 1, that contains the address of the file ID (18 characters). When you specify *(reg)*, the file ID must be exactly 18 characters in length; 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be padded with blanks. If the file mode is left blank, it is treated the same as an asterisk.

An asterisk (*) is not allowed for the file name or file type. An asterisk is allowed for the file mode, but is not generally recommended—see Usage Note “5” on page 214.

FSCB=

specifies the address of an FSCB. Acceptable values are:

label

specifies the label of an FSCB macroinstruction.

(reg)

specifies a register containing the address of an FSCB.

Note: The referenced FSCB should have the same specification for FORM as this FSPOINT.

Optional Parameters:

label

is an optional assembler label for the statement.

WRPNT=

specifies the new value of the write pointer. A write pointer of negative 1 (-1) indicates that the next item is to be put at the end of the file. A value of 0 specifies no change. If WRPNT is not specified, WRPNT=0 is the default. Acceptable values are:

number

specifies the new value of the write pointer as an absolute expression.

(reg)

specifies a register other than 1 containing the binary number.

RDPNT=

specifies the new value of the read pointer. A value of 0 specifies no change. If RDPNT is not specified, RDPNT=0 is the default. Acceptable values are:

number

specifies the new value of the read pointer as an absolute expression.

(reg)

specifies a register other than 1 containing the binary number.

FORM=E

specifies that an extended format FSCB is used. An extended format FSCB lets you specify a value up to $2^{31} - 1$ for WRPNT and RDPNT. If you do not specify FORM=E, the RDPNT value cannot exceed 65535 and the WRPNT value cannot exceed 65534.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

You can specify any general register other than 0, 1, or 15.

Usage Notes

1. You can use the same macroinstruction to change both the write and read pointers.
2. The file is implicitly opened for read in the following cases:
 - The file has not been explicitly opened with an earlier FSOPEN OPENTYP=READ, WRITE, NEW, or REPLACE.
 - The file has not been implicitly opened by an earlier FSREAD, FSWRITE, or FSPOINT.
3. When accessing SFS files, if the file is not already open, it is opened using the current default work unit identifier.
4. If you want reentrant code, you must specify the FSCB parameter.
5. When you specify the file mode as an asterisk, this usually means to locate the file with the specified file name and file type on the first accessed mode (in alphabetic order). However, if there are any files opened through an earlier FSOPEN, FSREAD, FSWRITE, or FSPOINT which match the file name and file type, one of these will be located first (because CMS checks for open files first) and the match is not guaranteed to follow the normal CMS file mode search order. Unexpected results may occur when you

access the same file name and file type inconsistently, using a specific file mode letter in some cases and an asterisk in others, and the given file ID exists on more than one accessed file mode.

If you want to access the file on the first accessed mode where it exists, the recommended approach is to:

- a. Set up an FSCB for the given file.
- b. Use FSOPEN to open it, specifying the file mode as an asterisk. If the FSOPEN is successful, FSOPEN will update the file mode appropriately in the FSCB.
- c. Use FSPOINT, specifying the same FSCB you used to open the file with FSOPEN. The file ID is already in the FSCB (and thus you would not specify it on FSPOINT) and has the file mode filled in (no longer an asterisk).

Return Codes

Register 15 contains one of the following return codes:

Code

Meaning

0

Successful operation.

1

File not found or not authorized.

2

Invalid read pointer or write pointer specified:

- The read pointer was not in the range of 0 to $2^{31} - 1$ when FORM=E was specified.
- The write pointer was not in the range of -1 to $2^{31} - 1$ when FORM=E was specified.

Note: If FORM=E is not coded, out-of-range conditions are not checked.

3

I/O operation to a minidisk failed.

4

First character of file mode is illegal.

7

The file was not previously opened and the file has an invalid record format.

20

Invalid character detected in file name.

21

Invalid character detected in file type.

25

Insufficient virtual storage available.

30

Some error, other than those in this list of codes, occurred when trying to implicitly open an SFS file. No rollback occurred.

31

Rollback occurred trying to implicitly open an SFS file. The work unit ID on which the rollback occurred is the current default work unit ID.

36

Disk or directory not accessed.

40

One of the following errors occurred:

- A required CSL routine was dropped.
- A required CSL routine was not loaded.

- There was an error in a user exit routine.
- There was an error calling the user accounting exit routine (DMS2AB).

55

APPC/VM error.

70

SFS file sharing conflict or minidisk file is already open by DMSOPEN or DMSOPDBK

80

I/O error accessing OS dataset.

81

OS read password protected dataset.

82

OS dataset organization is not BSAM, QSAM, or BPAM.

83

OS dataset has more than 16 extents.

84

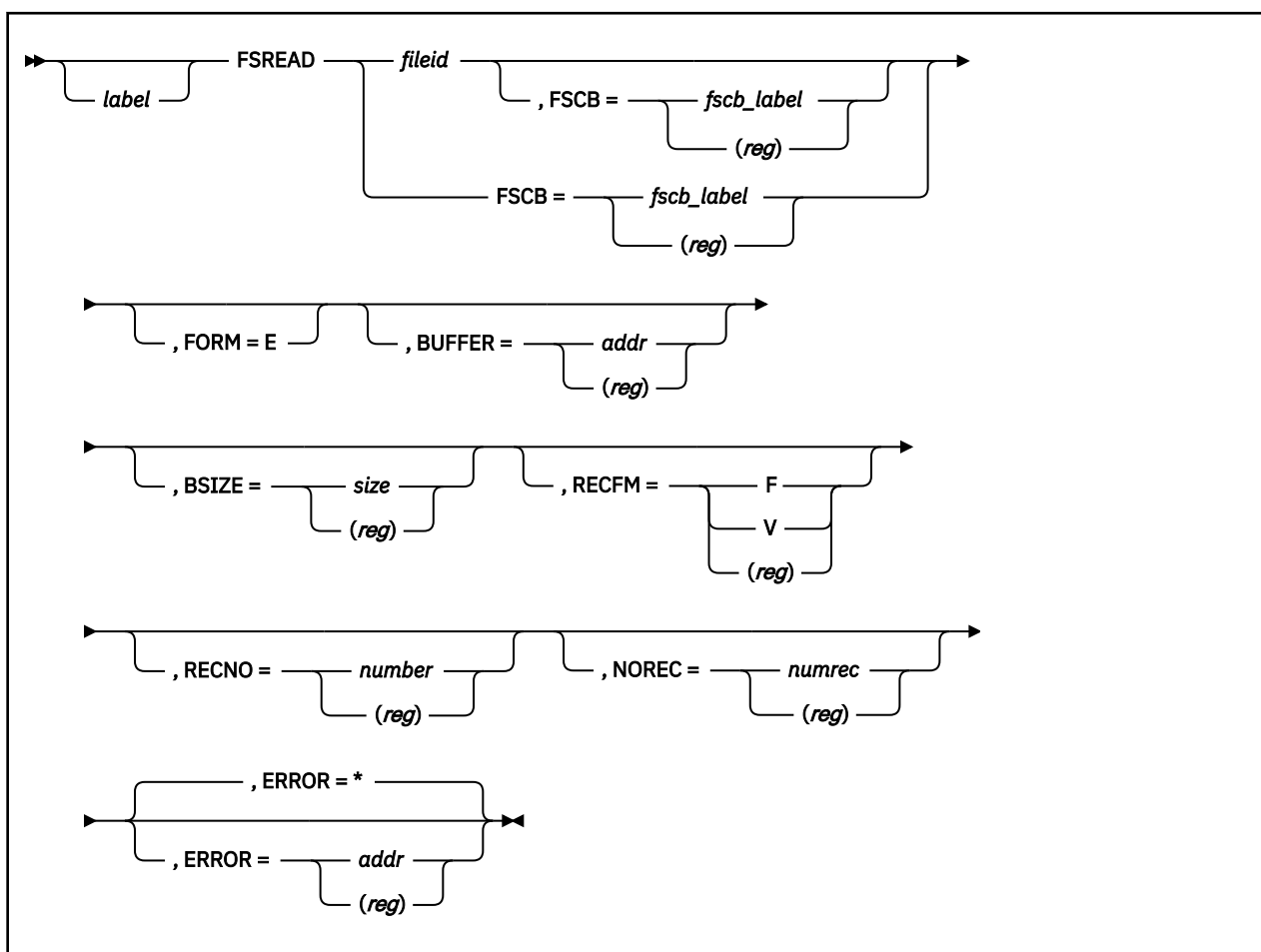
Attempt to point to a file on an OS or DOS formatted minidisk.

99

A required system resource is unavailable for one of the following reasons:

- There is insufficient virtual storage for the file pool.
- The file pool server is unavailable.

FSREAD



Purpose

Use the FSREAD macroinstruction to read one or more records from a file into your I/O buffer. The file must be on a minidisk or in an accessed directory to which you have read or write authority.

Parameters

Required Parameters:

fileid

specifies the CMS file identifier. It cannot be an erased or revoked alias. Also, you must have read or write authority to the file. Acceptable values are:

'fn ft fm'

the file ID enclosed in single quotation marks and separated by blanks. If *fm* is omitted, A1 is assumed.

(reg)

a register, other than 0 or 1, containing the address of the file ID (18 characters). When you specify *(reg)*, the file ID must be exactly 18 characters in length: 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be padded with blanks. If the file mode is left blank, it is treated the same as an asterisk.

An asterisk is not allowed for the file name or file type. An asterisk is allowed for the file mode, but is not generally recommended— see Usage Note [“19”](#) on page 220.

FSCB=

specifies the address of an FSCB. Acceptable values are:

fscb_label

specifies the label of an FSCB macroinstruction.

(reg)

specifies a register containing the address of an FSCB.

Note: The referenced FSCB should have the same specification for FORM as this FSREAD.

Optional Parameters:

label

is an optional assembler label for the statement.

FORM=E

must be specified when you use the extended format FSCB. (An extended format FSCB lets you specify a value up to $2^{31} - 1$ for RECNO and NOREC. If you do not specify FORM=E, the RECNO and NOREC values cannot exceed 65535.

BUFFER=

specifies the address of the I/O buffer into which the records are to be read. Acceptable values are:

addr

specifies the address of the I/O buffer as a relocatable expression.

(reg)

specifies the register (other than register 1) containing the address of the I/O buffer.

Note: The buffer address is interpreted as a 31-bit field and the high order bit is ignored.

BSIZE=

specifies the number of bytes to be read. For a file with variable-length records, this parameter specifies the length of the record to be read. For a file with fixed-length records, this parameter must be equal to the product of the NOREC parameter and the logical record length. This must be a positive signed binary integer. Acceptable values are:

size

specifies the number of bytes to be read or written as an absolute expression.

(reg)

specifies the register (other than register 1) containing the number of bytes to be read or written.

RECFM=

specifies the format in the file. Acceptable values are:

F

specifies that every record in the file has the same length. This is the default value if FSCB is not specified.

V

specifies that records in the file may have different lengths.

(reg)

specifies the register (other than register 1) whose low-order byte contains the record format (C'F' or C'V').

RECNO=

specifies the record number of the first (or only) record to be read, relative to the beginning of the file, record 1. If FSCB is not specified, the default is 0, which indicates that the first (or only) record to be read is the record which follows the last record read by the previous FSREAD. Acceptable values are:

number

specifies the record number as an absolute expression.

(reg)

specifies the register (other than register 1) containing the record number.

NOREC=

specifies the number of records to be read. If FSCB is not specified, the default is 1, which is also the only valid value for files with variable-length records. Acceptable values are:

numrec

specifies the number of records as an absolute expression.

(reg)

specifies the register (other than register 1) containing the number of records.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

You can specify any general register other than 0, 1, or 15.

Usage Notes

1. FSREAD updates the read pointer so that, if RECNO=0 on the next FSREAD operation, reading begins following the last record read by this FSREAD.
2. On return from the FSREAD macro, register 1 contains the address of the FSCB for the file. (If the FSCB parameter was not specified, the FSREAD macro creates one as part of the generated code.) Register 0 contains the number of bytes actually read. Register 0 is set on both the error and nonerror paths. The number of bytes read is zero for any return code other than 0 or 8. This information is also contained in the FSCBNORD field of the FSCB.
3. If an FSCB macroinstruction has not been coded for a file (and you do not code the FSCB parameter on FSREAD), you must specify the BUFFER and BSIZE parameters. If the file contains variable-length records, you must also specify RECFM=V.
4. If you code both *fileid* and the FSCB parameter, CMS uses *fileid* to fill in the FSCB. This applies to the BUFFER, BSIZE, FORM, RECFM, RECNO, and NOREC parameters as well.
5. If you want reentrant code, you must specify the FSCB parameter.
6. For the first read operation in a newly-opened existing file, reading begins with record 1 unless otherwise specified by the RECNO parameter. For subsequent read operations, reading begins following the last record read unless the RECNO parameter is specified with a nonzero value to indicate the number of the record to be read.

To read records sequentially beginning with a particular record number, use the RECNO parameter to specify the first record to be read. On the next FSREAD macroinstruction, use RECNO=0 so that reading continues sequentially, following the first record read. This can also be accomplished by coding an FSPPOINT macro with the RDPNT operand set to the record number of the first record to be read.

7. If an attempt is made to read a record which has never been written (referred to as a sparse record) in a file of fixed-length records, CMS returns a record of all X'00'. You can read a sparse record by specifying an unused record number for the RECNO parameter.
8. The CMS file system does not support null (zero-length) records. The FSREAD macro cannot be used to read records with a length of 0.

9. To read more than one record on a single FSREAD (valid for fixed-length files only), use the BSIZE and NOREC parameters to specify the sum of the lengths of the records to be read and the number of records to be read, respectively. For example, to read ten 80-byte records, you should specify BSIZE=800 and NOREC=10. The buffer you use must be at least 800 bytes long.
10. Variable-length records can be up to 65,535 bytes. Variable-length records are read one at a time. When reading variable-length records, a record that is longer than the buffer length is truncated.
11. If a record is longer than the buffer length, FSREAD truncates the record. If the length of a group of records to be read is longer than the buffer length, all data beyond the length of the buffer is truncated, but the read pointer is still positioned at the end of the group of records read.

To avoid a truncation warning, the value specified on BSIZE must be the product of the logical record length and NOREC. However, it is not considered an error if the BSIZE value is not equal to this product. When it is not, it is possible to skip data during sequential reads. For example, suppose a file has twenty 80-byte fixed-length records. If the first read from the file requests five records and specifies a data length of 300, the data from the first three records and the first 60 bytes of the fourth record will be placed in the buffer and a truncation warning will be returned. If the next read does not specify a position number, reading will begin with record 6, thereby skipping the last 20 bytes in record 4 and all of record 5.
12. The BSIZE parameter specifies the maximum number of bytes to be read. The amount of data placed in the buffer is less than the value specified in the BSIZE parameter when:
 - The *position* parameter (or its default value) specifies a record beyond the current end of the file. In this case, no data is placed in BUFFER and an end-of-file warning is returned to the caller.
 - The end of the file is reached before filling BUFFER because more records were requested than remained in the file. No end-of-file warning is returned.
 - For a file with fixed-length records, the product of the NOREC parameter times the logical record length is less than the value specified by the BSIZE parameter. In this case, the product specifies the number of bytes placed in the user's buffer and no warning is returned to the caller.
 - The file has variable-length records and the length of the record being read is less than the value in the BSIZE parameter.

When the read is successful but the amount of data placed in the buffer is less than the size specified, the buffer space beyond the record(s) read in may contain unpredictable characters because FSREAD reads the record(s) **without** clearing the buffer first.
13. An end-of-file warning (return code 12) is returned **only** when no data is placed in the buffer. FSREAD does not return a warning (return code 12) when it reads fewer records than requested because of reaching the end of the file.
14. The read is considered successful if the return code is 0, 8, or 12.
15. The contents of the buffer are unpredictable when the read is **not** successful; data in the buffer before the FSREAD may be partially modified.
16. The file is implicitly opened for read when the file has not been:
 - Explicitly opened with an earlier FSOPEN OPENTYP=READ, WRITE, NEW, or REPLACE
 - Implicitly opened by an earlier FSREAD, FSWRITE, or FSPOINT.
17. When accessing SFS files, if the file is not already open, it is opened using the current default work unit identifier.
18. For a file open for output, the FSCBTHEx (X'80') indicator bit of the FSCBFLG byte indicates when you have reached your SFS file space threshold. (For more information on the SFS file space threshold, see the SET THRESHOLD command in *z/VM: CMS Commands and Utilities Reference*.) Because the CMS portion of the file system does buffering, you will only see the indicator when it is necessary to write the buffers to the file pool. This can occur during a read, write, or close. For small files, the indicator might not be returned until the close.
19. When you specify the file mode as an asterisk, this ordinarily means to locate the file with the specified file name and file type on the first accessed mode (in alphabetic order). However, if there are any files opened through an earlier FSOPEN, FSREAD, FSWRITE, or FSPOINT which match the file

name and file type, one of these will be located first (because CMS checks for open files first) and the match is not guaranteed to follow the normal CMS file mode search order. Unexpected results may occur when you access the same file name and file type inconsistently, using a specific file mode letter in some cases and an asterisk in others, and the given file ID exists on more than one accessed file mode.

If you want to access the file on the first accessed mode where it exists, the recommended approach is to:

- Set up an FSCB for the given file.
- Use FSOPEN to open it, specifying the file mode as an asterisk. If the FSOPEN is successful, FSOPEN will update the file mode appropriately in the FSCB.
- Use FSREAD, specifying the same FSCB you used to open the file with FSOPEN. The file ID is already in the FSCB (and thus you would not specify it on FSREAD) and has the file mode filled in (no longer an asterisk).

Return Codes

Register 15 contains one of the following return codes:

Code

Meaning

0

Successful execution (also used for SFS reason code 51050 = successful operation, but SFS file pool block threshold was reached during the operation).

1

File not found, disk not accessed, or insufficient authority.

2

Invalid buffer address.

3

I/O operation to a minidisk failed.

This may occur if you link to and access another user's disk, then try to read a file that was refiled by its owner after you issued the ACCESS command. Re-issue the ACCESS command and try to read the file again.

It is also possible that the disk was detached (through the DETACH command) without having been released (through the RELEASE command), or the disk is an unsupported device.

4

First character of file mode is illegal.

5

Number of records to read is equal to zero.

7

AFT is not marked with a record format of F or of V. If the file was not previously opened, this indicates that the file has an invalid record format.

8

Successful operation, but the buffer was too small to hold all of the requested data. The buffer was filled with as much data as it would hold.

11

Number of records to read is not exactly one for a file with variable-length records.

12

No records were read because end of file was reached or because the position parameter specified a record number greater than the number of records in the file.

13

Found an invalid displacement in the AFT for a file with variable-length records (this indicates a coding error: it should not occur).

- 20**
Invalid character detected in file name.
- 21**
Invalid character detected in file type.
- 25**
Insufficient free virtual storage available for file system control blocks (also used for SFS reason code 91028 = unable to obtain space on the system stack).
- 26**
Position is negative, the number of records to read is negative, or position plus the number of records to process exceeds $2^{31} - 1$, the file system capacity.
- 29**
Storage group space limit reached.
- 30**
Some error, other than those in this list of codes, occurred while accessing an SFS file. No rollback occurred.
- 31**
Rollback occurred while trying to access an SFS file. The work unit ID on which the rollback occurred is the default work unit ID at the time the file was opened by the first operation to the file.

An application error, system error, or lack of required resource can be the cause of this return code. If the error persists, refer to the *z/VM: Diagnosis Guide* for more information about diagnosing the problem.
- 40**
One of the following errors occurred:
- A required CSL routine was dropped.
 - A required CSL routine was not loaded.
 - There was an error in a user exit routine.
 - There was an error calling the user accounting exit routine (DMS2AB).
- 42**
Invalid record length detected while attempting to read a variable length record.
- 43**
Logical record length is non-positive.
- 44**
Last record number is non-positive.
- 47**
File type is unsupported for a block operation.
- 48**
File is empty.
- 49**
External object cannot be opened.
- 50**
File is in DFSMS/VM migrated status and implicit RECALL is set to OFF.
- 51**
Error occurred during DFSMS/VM file recall processing.
- 55**
APPC/VM error.
- 70**
SFS file sharing conflict or minidisk file is already open by DMSOPEN or DMSOPDBK with an output intent.

80

I/O error accessing OS dataset.

81

OS read password protected dataset.

82

OS dataset organization is not BSAM, QSAM, or BPAM.

83

OS dataset has more than 16 extents.

84

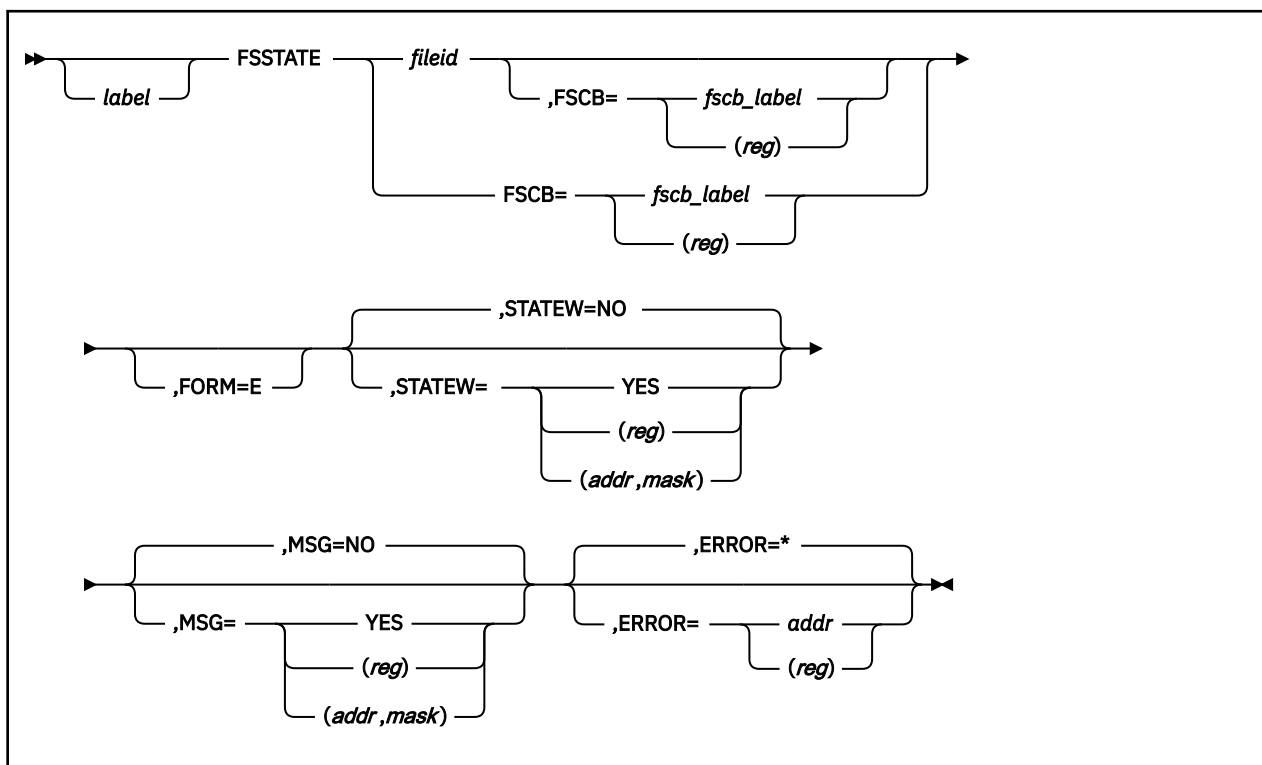
Attempt to read a file on an OS or DOS formatted minidisk.

99

A required system resource is unavailable for one of the following reasons:

- There is insufficient virtual storage for the file pool server.
- The file pool server is unavailable.
- File is in migrated status and DFSMS is not enabled.

FSSTATE



Purpose

Use the FSSTATE macroinstruction to determine whether a particular file exists.

Parameters

Required Parameters:

fileid

specifies the CMS file identifier. Acceptable values are:

'fn ft fm'

specifies the file ID enclosed in single quotation marks and separated by blanks. If *fm* is omitted, A1 is assumed. An asterisk (*) can be specified for *fn*, *ft*, or *fm*, or any combination. Specifying asterisks for both *fn* and *ft* indicates that you want to check for the existence of any file. Specifying an asterisk for *fm* means that the file with the specified file name and file type on the first accessed mode (in alphabetic order) will be found.

(reg)

specifies a register, other than 0 or 1, that contains the address of the file ID (18 characters). When you specify *(reg)*, the file ID must be exactly 18 characters in length: 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be padded with blanks. If the file mode is left blank, it is treated the same as an asterisk, meaning that the file with the specified file name and file type on the first accessed mode (in alphabetic order) will be found.

Note: Because CMS checks for open files first, you may get unexpected results when specifying an asterisk for *fm* if there are open files matching the file name and file type specified.

FSCB=

specifies the address of an FSCB. Acceptable values are:

fscb_label

specifies the label on an FSCB macroinstruction.

(reg)

specifies a register that contains the address of an FSCB.

Note: The referenced FSCB should have the same specification for FORM as this FSSTATE.

Optional Parameters:

label

is an optional assembler label for the statement.

FORM=E

must be specified when the extended format FSCB is being used. An extended format FSCB is required to process files with more than 65535 records or more than 65535 data blocks.

STATEW=

specifies whether STATEW processing is to be performed. STATEW is a CMS command that you can use to verify the existence of a file on a disk. However, STATEW will not find a file for which you do not have write authority nor a file which is on a file mode accessed read only. For more information, see [z/VM: CMS Commands and Utilities Reference](#). Acceptable values are:

NO

specifies that STATEW processing is not to be performed. This is the default value.

YES

specifies that STATEW processing is to be performed.

(reg)

the macro checks the value of the specified register (other than register 1) and, if it is 0, sets STATEW to NO. If the register contains a nonzero value, the macro sets STATEW to YES.

(addr,mask)

defines a single bit in storage that sets the value of the STATEW parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then STATEW is set to NO. If the bit is 1, then STATEW is set to YES. For example, to test the first bit in the single byte of storage at location FLAGS, specify the STATEW parameter as

```
STATEW=(FLAGS,X'80')
```

To set the value of the STATEW parameter at assembly time, specify STATEW=YES or STATEW=NO. To set the value at execution time, specify STATEW=(*reg*) or STATEW=(*addr,mask*).

MSG=

specifies whether messages are to be displayed during STATE processing. Acceptable values are:

NO

specifies that messages are not to be displayed. This is the default value.

YES

specifies that messages are to be displayed.

(reg)

the macro checks the value of the specified register (other than register 1) and, if it is 0, sets MSG to NO. If the register contains a nonzero value, the macro sets MSG to YES.

(addr,mask)

defines a single bit in storage that sets the value of the MSG parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the

specified bit is 0, then MSG is set to NO. If the bit is 1, then MSG is set to YES. For example, to test the first bit in the single byte of storage at location MSGFLAG, specify the MSG parameter as

```
MSG=(MSGFLAG,X'80')
```

To set the value of the MSG parameter at assembly time, specify MSG=YES or MSG=NO. To set the value at execution time, specify MSG=(reg) or MSG=(addr,mask).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

You can specify any general register other than 0, 1, or 15.

Usage Notes

1. FSSTATE will not find aliases that have been erased or revoked, nor files for which the user does not have read (STATEW=NO) or write (STATEW=YES) authority.
2. If the specified file exists, FSSTATE returns a 0 return code in register 15.

Register 1 contains the address of a copy of the file status table (FST) information for the specified file. Use the FSTD macro to map information about CMS disk files in the FST returned by FSSTATE. Any information needed from the FST should be extracted immediately after the FSSTATE call because any use of the file system may change its contents.

For FORM=E, the FST contains the following information:

Decimal Displace- ment	Field Description
0	File name
8	File type
16	Reserved
24	File mode
26	Reserved
30	Record format (F/V)
31	FST Flag Byte
32	Logical record length
36	Reserved
40	Alternate file origin pointer
44	Alternate number of data blocks
48	Alternate item count
52	Number of pointer block levels
53	Length of pointer element

Decimal Displace- ment	Field Description
54	Alternate date/time (yy mm dd hh mm ss)
60	Real file mode

When FORM=E is not specified, the FST contains the following information:

Decimal Displace- ment	Field Description
0	File name
8	File type
16	Date (mmdd) last written
18	Time (hhmm) last written
20	Write pointer (number of item)
22	Read pointer (number of item)
24	File mode
26	Number of records in file
28	Disk address of first chain link
30	Record format (F/V)
31	FST Flag Byte
32	Logical record length
36	Number of 800-byte data blocks
38	Year (yy) last written

Flag settings for the FST Flag Byte for both forms:

Bits 0 and 1 describe how the disk containing the file is accessed:

**Value
Meaning**

X'C0'
Extension of read/write disk

X'80'
Read/write disk

X'40'
Extension of read-only disk

X'00'
Read-only disk

Bit 3 indicates whether the file is a Shared File System (SFS) file or a minidisk file:

**Value
Meaning**

X'10'
The file resides in the Shared File System

Bit 4 indicates the century the file was last written or updated. If bit 4 is off, then the year is in the 1900s. If bit 4 is on, then the year is in the 2000s.

Value
Meaning

X'08'

Century for date last written (0 = 19, 1 = 20). This corresponds to both Alternate date/time year yy and the Year (yy) last written.

Bits 5, 6, and 7 describe whether the file is open (active) through the file system macro interface (FSOPEN, FSREAD, FSWRITE, and FSPOINT):

Value
Meaning

X'04'

Active for reading

X'02'

Active for writing

X'01'

Active from a point

Note: The logical record length of a file field in the FST may not be available on an FSSTATE of a new file currently open for output. This may be detected by a number of records of zero.

3. The FSSTATE macroinstruction disregards the file mode number specified when both the file name and file type are explicitly specified. When the file name or file type (or both) are specified as asterisk (*), the file mode number is respected.
4. The MSG parameter of the FSSTATE macroinstruction provides message control. MSG=YES issues error messages for any nonzero return code. MSG=NO issues all error messages except:
 - 002E File not found, RC=28.
 - 069E Mode not accessed, RC=36.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code
Meaning

20

Invalid character in file name or file type.

24

Invalid file mode. Allowable file modes are any alphabetic character, blank, or *. When file mode is alphabetic, an optional file mode number (0-6) may also be specified.

28

File not found or not authorized, or not accessed R/W when STATEW=YES.

36

Disk or directory not accessed.

80

I/O error accessing OS dataset.

81

OS read password protected dataset.

82

OS dataset organization is not BSAM, QSAM, or BPAM.

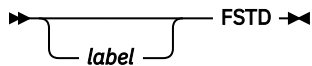
83

OS dataset has more than 16 extents.

88

Non-extended format FSCB supplied and a non-extended CDF format copy of the FST cannot be built (number of records or number of data blocks exceeds 65535).

FSTD



Purpose

Use the FSTD macro to generate a DSECT for the FST control block. FSTD maps information about CMS disk files in the FST returned by FSSTATE and FSOPEN.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the FSTD macro expansion is labeled FSTD.

Usage Notes

1. The FSTD macroinstruction expands as follows:

```

FSTD      FSTD
FSTD      DSECT
FSTDFNFT DS 0CL16      filename and filetype
FSTFNAME DS CL8 -      filename
FSTFTYPE DS CL8 -      filetype
FSTDATEW DS 1H -      DATE LAST WRITTEN - MMDD
FSTTIMEW DS 1H -      TIME LAST WRITTEN - HHMM
FSTWRPNT DS 1H -      WRITE POINTER - ITEM NUMBER
FSTRDPNT DS 1H -      READ POINTER - ITEM NUMBER
FSTFMODE DS 1H -      FILE MODE - LETTER AND NUMBER
FSTRECCT DS 1H -      NUMBER OF LOGICAL RECORDS
FSTFCLPT DS 1H -      FIRST CHAIN LINK POINTER
FSTRECFM DS 1C -      F*1 - RECORD FORMAT - F OR V
*
*      FSTRECFM flag byte definitions
*
FSTDFIX EQU C'F' -      Fixed record format
FSTDVAR EQU C'V' -      Variable record format
*
FSTFLAGS DS 1X -      F*2 - FST FLAG BYTE
*
*      FSTFLAGS DESCRIPTION
*
FSTRWDSK EQU X'80' -      READ/WRITE DISK
FSTRODSK EQU X'00' -      READ/ONLY DISK
FSTDSPFS EQU X'10' -      Shared File FST
FSTXRDSK EQU X'40' -      EXTENSION OF R/O DISK
FSTXWDSK EQU X'C0' -      EXTENSION OF R/W DISK
FSTEPL EQU X'20' -      EXTENDED PLIST
FSTDIA EQU X'40' -      ITEM AVAILABLE
FSTDRA EQU X'01' -      PREVIOUS RECORD NULL
SPACE 1
FSTCNTRY EQU X'08' -      Century for date last written
*                          (0=19, 1=20), corresponds to
*                          FSTYEARW, FSTADATI.
*
SPACE 1
FSTACTRD EQU X'04' -      ACTIVE FOR READING
FSTACTWR EQU X'02' -      ACTIVE FOR WRITING
FSTACTPT EQU X'01' -      ACTIVE FROM A POINT
FSTFILEA EQU X'07' -      THE FILE IS ACTIVE
*
FSTLRECL DS 1F -      LOGICAL RECORD LENGTH
FSTBLKCT DS 1H -      NUMBER OF 800 BYTE BLOCKS
FSTYEARW DS 1H -      YEAR LAST WRITTEN
FSTFOP DS F -      ALT. FILE ORIGIN POINTER
FSTADBC DS F -      ALT. NUMBER OF DATA BLOCKS

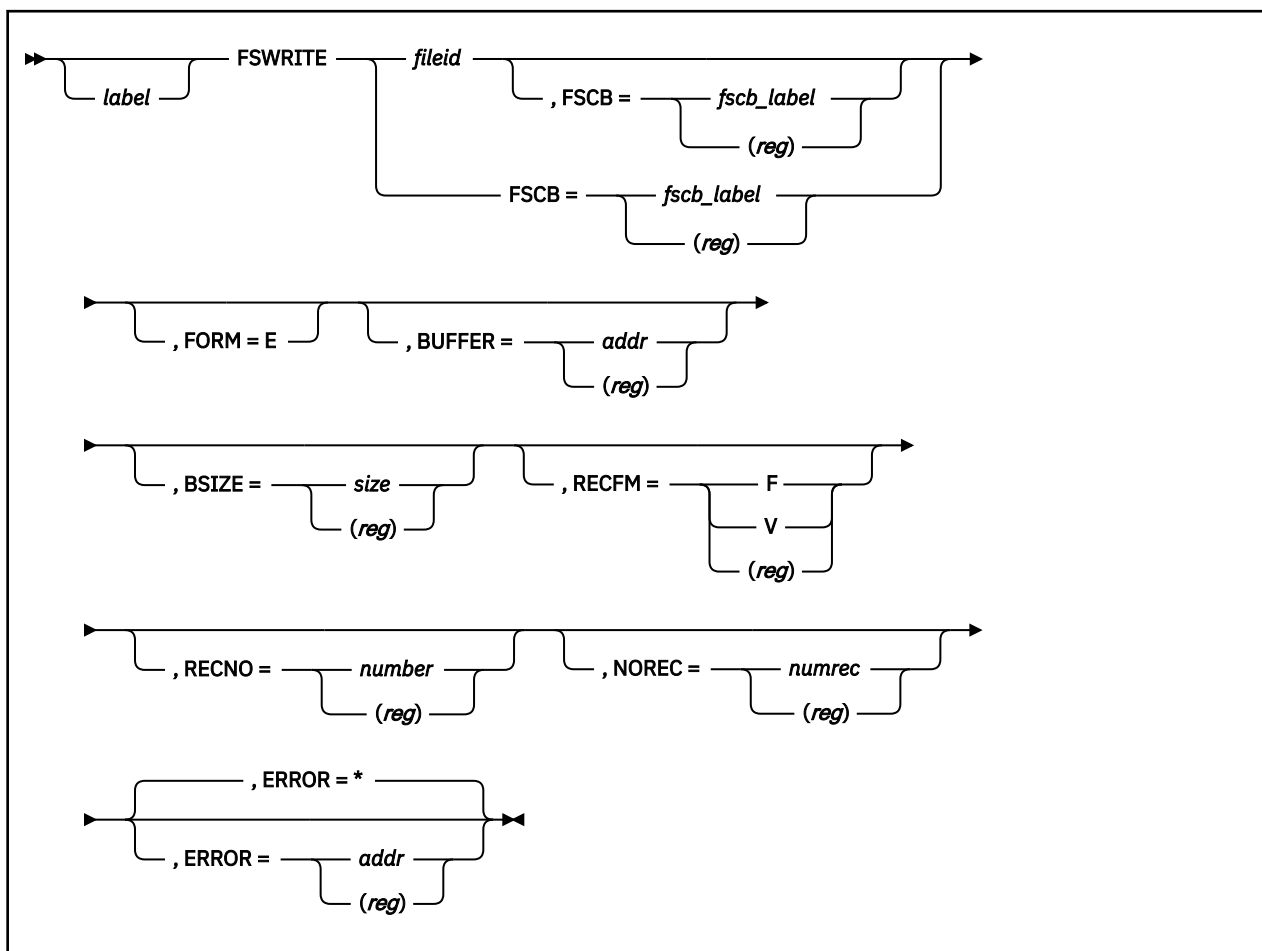
```

```

FSTAIC  DS    F          ALT. ITEM COUNT
FSTNLVL DS    XL1        NUMBER OF POINTER BLOCK LEVELS
FSTPTRSZ DS    XL1        LENGTH OF A POINTER ELEMENT
FSTADATI DS    CL6        ALT. DATE/TIME(YY MM DD HH MM SS)
FSTREALM DS    CL1        Real filemode
FSTFLAG2 DS    1X -      F*3 - FST FLAG BYTE 2
*        FSTFLAG2 DESCRIPTION
*
FSTPIPEU EQU    X'10' -   Reserved for CMS PIPELINES usage
DS          CL2          - reserved -
FSTDSIZE EQU    (*-FSTD) - FST SIZE IN BYTES

```

FSWRITE



Purpose

Use the FSWRITE macroinstruction to write a record from an I/O buffer to a CMS file. The file must be on a read/write minidisk or in an SFS directory accessed in read/write status. If the file is in an SFS directory, you must have the proper authorities. (Use the FORCERW option of the ACCESS command to access another user's directory in read/write status.)

Parameters

Required Parameters:

fileid

specifies the CMS file identifier. The file specified cannot be an erased or revoked alias.

'fn ft fm'

the file ID enclosed in single quotation marks and separated by blanks. If *fm* is omitted, A1 is assumed.

(reg)

a register, other than 0 or 1, containing the address of the file ID (18 characters). When you specify *(reg)*, the file ID must be exactly 18 characters in length: 8 for the file name, 8 for the file type, and 2 for the file mode. Shorter names must be padded with blanks. The file mode letter must be specified; if the file mode number is left blank, it is assumed to be the same as the existing file or 1 in the case of a new file.

An asterisk (*) is not allowed for *fn*, *ft*, or *fm*.

FSCB=

specifies the address of an FSCB. Acceptable values are:

fscb_label

specifies the label on an FSCB macroinstruction.

(reg)

specifies a register containing the address of an FSCB.

Note: The referenced FSCB must have the same specification for FORM as this FSWRITE.

Optional Parameters:

label

is an optional assembler label for the statement.

FORM=E

must be specified when the extended format FSCB is being used. (An extended format FSCB lets you specify a value up to $2^{31} - 1$ for RECNO and NOREC. If you do not specify FORM=E, the RECNO and NOREC values cannot exceed 65535.

BUFFER=

specifies the address of the I/O buffer containing the record(s) to be written. Acceptable values are:

addr

specifies the address of the I/O buffer as a relocatable expression.

(reg)

specifies the register (other than register 1) containing the address of the I/O buffer.

Note: The buffer address is interpreted as a 31-bit field and the high order bit is ignored.

BSIZE=

specifies the size in bytes to be written. For a file with variable-length records, this parameter specifies the length of the record to be written. For a file with fixed-length records, this parameter must be equal to the product of the *records* parameter and the logical record length. This must be a positive signed binary integer. Acceptable values are:

size

specifies the number of bytes to be read or written as an absolute expression.

(reg)

specifies the register (other than register 1) containing the number of bytes to be read or written.

RECFM=

specifies the format in the file. Acceptable values are:

F

specifies that every record in the file has the same length. This is the default value if FSCB is omitted.

V

specifies that records in the file may have different lengths.

(reg)

specifies the register (other than register 1) whose low-order byte contains the record format (C'F' or C'V').

RECNO=

specifies the record number of the first (or only) record to be written, relative to the beginning of the file, record 1. The default is 0 if FSCB is omitted, which indicates that the first (or only) record to be written is to follow the last record written by the previous FSWRITE. Acceptable values are:

number

specifies the record number as an absolute expression.

(reg)

specifies the register (other than register 1) containing the record number.

NOREC=

specifies the number of records to be written. The default is 1 if FSCB is omitted, which is also the only valid value for a file with variable-length records. Acceptable values are:

numrec

specifies the number of records as an absolute expression.

(reg)

specifies the register (other than register 1) containing the number of records.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

n specify any general register other than 0, 1, or 15.

Usage Notes

1. FSWRITE updates the write pointer so that, if RECNO=0 is specified on the next FSWRITE operation, writing begins following the last record written by this FSWRITE.
2. On return from the FSWRITE macroinstruction, register 1 contains the address of the FSCB for the file. (If the FSCB parameter was not specified, the FSWRITE macro creates one as part of the generated code.)
3. If an FSCB macroinstruction has not been coded for a file (and you do not code the FSCB parameter on FSWRITE), you must specify the BUFFER and BSIZE parameters. If the file has variable-length records, you must also specify RECFM=V.
4. If you code both *fileid* and the FSCB parameter, CMS uses *fileid* to fill in the FSCB. This applies to FORM, BUFFER, BSIZE, RECFM, RECNO, and NOREC parameters as well.
5. If you want reentrant code, you must specify the FSCB parameter.
6. For new files, writing begins with record 1 unless otherwise specified by the RECNO parameter. For existing files, writing begins following the last record written unless the RECNO parameter is specified with a nonzero value to indicate the number of the record to be written.

To write records sequentially beginning with a particular record number, use the RECNO parameter to specify the first record to be written. On the next FSWRITE macroinstruction, use RECNO=0 so that writing continues sequentially, following the first record written.

7. For files with fixed-length records only, it is permissible to write a record with a position number more than one greater than the number of the last record. Records that have a skipped position numbers are referred to as sparse records. Sparse records are not written to a file, however, when you open a file you can write to a record that was previously sparse. If an attempt is made to read a sparse record it will be retrieved as all X'00' bytes. You can read a sparse record by specifying an unused record number for the RECNO parameter.
8. The CMS file system does not support null (zero-length) records. The FSWRITE macroinstruction cannot be used to write records with a length of 0.
9. To write more than one record on a single FSWRITE (valid for fixed-length files only), use the BSIZE and NOREC parameters to specify the sum of the lengths of the records to be written and the number

of records to be written, respectively. For example, to write ten 80-byte records, you should specify BSIZE=800 and NOREC=10. The buffer you use must be at least 800 bytes long.

10. Variable-length records can be up to 65,535 bytes long. When you use the FSWRITE macroinstruction to update an existing file of variable-length records, the replacement record must be the same length as the original record. If it is not, the results are as follows:
 - In the EDF file system, an attempt to write a record shorter or longer than the original record on a disk formatted with 512-byte, 1 KB, 2 KB, or 4 KB block size results in truncation of the file at the specified record number with no error return codes.
 - An attempt to write a record shorter or longer than the original record in an SFS file with the FSWRITE macro results in truncation of the file at the specified record number with no error return codes. (If you are using the shared file system and do not need to use minidisk files, it would be worthwhile for you to consider using DMSWRITE, the callable services library (CSL) interface.)
11. The 'update-in-place' facility lets you write blocks back to their previous location on disk. For files on minidisks, the 'update-in-place' attribute is indicated by a file mode number of 6.



Attention: Neither the integrity of the file nor of the disk on which it resides is guaranteed when updating an existing file mode number 6 on a minidisk. For details, see 'EDF Data Integrity' in *z/VM: CMS Application Development Guide for Assembler*. For SFS files, file mode number 6 is treated the same as file mode number 1. 'Update-in-place' on SFS files is achieved by specifying the overwrite attribute as INPLACE. For details, see 'Overwrite Attribute' in *z/VM: CMS Application Development Guide*.

Note: For a variable format file, 'update-in-place' applies only if a record is replaced by a record with the same length.

12. The FSWRITE fails if the file has been explicitly opened with an FSOPEN OPENTYP=READ.
13. FSWRITE causes the file to be implicitly opened for write if the file has not been explicitly opened with an FSOPEN OPENTYP=WRITE, NEW, or REPLACE, or if the file has not been opened by an earlier FSWRITE. If the file has been implicitly opened for read by an earlier FSREAD or FSPOINT, the file is then upgraded to an implicit open for write.
14. When accessing SFS files, if the file is not already open, it is opened using the current default work unit identifier.
15. If the file is in an SFS directory, you must have write authority to the file. If you are creating a new file in the directory, you must have write authority to the directory.
16. For a file open for output, the FSCBTHEX (X'80') indicator bit of the FSCBFLG byte indicates when you have reached your SFS file space threshold. (For more information on the SFS file space threshold, see the SET THRESHOLD command in *z/VM: CMS Commands and Utilities Reference*.) Because the CMS portion of the file system does buffering, you will only see the indicator when it is necessary to write the buffers to the file pool. This can occur during a read, write, or close. For small files, the indicator might not be returned until the close.
17. An FSWRITE to a file in an SFS directory will fail when CMS detects that the file space is full². If all the conditions listed below are true, a file space full condition will imply that there is sufficient space left on the SFS file space to close and commit all updated files. In other words, a subsequent commit will commit all updates prior to the failing FSWRITE.
 - There is no concurrent write sharing to the file space (that is file pool and user ID) among users.
 - All SFS files being updated by your applications are at VM/ESA Release 2.1 or above. Note that files **can** reside in more than one file space, and in more than one file pool.
 - Your application modifies SFS files exclusively using:
 - FS macros,

² FSWRITE will not fail for a file space full condition when writing to pre-VM/ESA Release 2.1 SFS file pool servers.

- EXECIO, or
- OS Simulation WRITE/PUT macros.
- Your application does not acquire work unit IDs for SFS processing. In other words, your application does not use the DMSGETWU, DMSPUSWU, or DMSPOPWU CSL routines to manipulate work unit IDs.

If the above conditions are not met and a file space full condition is detected, you may have written more blocks than are available in the file space. In the case where you have written more blocks than are available, a rollback will be performed.

If you cannot guarantee that your application will be run exclusively in an environment where the conditions listed above are met, you may use the FSCBTHEX (X'80') indicator bit of the FSCBFLG byte to monitor file space usage to better anticipate a file space full condition.

18. It may appear from QUERY DISK or QUERY LIMITS output that there are enough blocks available to write a record, but the FSWRITE fails with a return code 13. CMS is conservative in determining the number of blocks needed to complete a write. Several factors influence this determination, including:
- system blocks written in addition to the data blocks,
 - shadowing required to preserve data integrity, and
 - system buffering of write requests.

Return Codes

Register 15 contains one of the following return codes:

Code

Meaning

0

Successful execution (also used for SFS reason code 51050 = successful operation, but SFS file pool block threshold was reached during the operation).

1

Not authorized to write to file.

2

Invalid buffer address.

3

I/O operation to a minidisk failed.

This may occur if the disk was detached (through the DETACH command) without having been released (through the RELEASE command), or the disk is an unsupported device.

4

First character of file mode is illegal or disk not accessed.

5

Second character of file mode is illegal.

6

The last record number to be written is too large (more than 65535) to fit in a halfword and an extended plist is not specified.

7

Position specifies a record number that is more than one greater than the current number of records in a file with variable-length records.

8

Size of output buffer is not greater than zero or an attempt was made to write a null record to a file with variable length records.

11

FSCB is not marked with a record format of F nor of V.

- 12** Disk or directory not accessed R/W.
- 13** Your minidisk is full or your SFS file space limit is reached.
- 14** Size of output buffer is not evenly divisible by the number of records for a file with fixed-length records.
- 15** Attempt to alter the record length of a file with fixed-length records.
- 16** Record format specified not the same as file.
- 17** Size of output buffer is greater than 65535 for a file with variable-length records.
- 18** Number of records to write is not exactly one for a file with variable-length records.
- 20** Invalid character detected in file name.
- 21** Invalid character detected in file type.
- 24** File specified on FSCB does not satisfy input conditions.
- 25** Insufficient free storage available for file system control blocks (also used for SFS reason code 91028 = unable to obtain space on the system stack).
- 26** Position specifies a negative record number or number of records to write is negative or position plus the number of records exceeds the file system capacity ($2^{31} - 1$) or logical block number computed by system exceeds the file system capacity ($2^{31} - 1$).
- 29** The storage group space limit was reached.
- 30** Some error, other than those in this list of codes, occurred while accessing an SFS file. No rollback occurred.
- 31** Rollback occurred while trying to access an SFS file. The work unit on which the rollback occurred is the default work unit at the time the file was opened by the first operation to the file.
- An application error, system error, or lack of required resource can be the cause of this return code. If the error persists, refer to the *z/VM: Diagnosis Guide* for more information about diagnosing the problem.
- 38** File explicitly opened with read intent.
- 39** A disk is accessed as a read only extension of another, and a given file exists on the extension disk but not on the parent disk. The file has been opened through FSREAD, FSPOINT or FSOPEN with a read intent and the file mode specified on the original FS Macro is that of the parent disk. An FSWRITE was subsequently issued using the same file ID. This may occur when the parent disk is accessed as Read Only or Read Write.
- 40** One of the following errors occurred:
- A required CSL routine was dropped.

- A required CSL routine was not loaded.
- There was an error in a user exit routine.
- There was an error calling the accounting exit routine (DMS2AB).

42

Block contains a variable-length record whose length is outside the range 1. n , where n is the value specified during block interface initialization as the maximum length of any record in the file.

43

Logical record length is non-positive.

44

Last record number is non-positive.

47

File type is unsupported for a block operation.

49

External object cannot be opened.

50

File is in DFSMS/VM migrated status and implicit RECALL is set to OFF.

51

Error occurred during DFSMS/VM file recall processing.

55

APPC/VM error.

70

One of the following sharing conflicts occurred:

- The file is locked.
- The file pool server detected a deadlock.
- The file is open for write through SFS OPEN.
- The file is open for write by another user.
- You attempted to write to a file that is currently implicitly open for READ, but the file has been changed since it was originally opened.
- The minidisk file is already open by DMSOPEN or DMSOPDBK when issuing an FSWRITE.

80

I/O error accessing OS dataset.

81

OS read password protected dataset.

82

OS dataset organization is not BSAM, QSAM, or BPAM.

83

OS dataset has more than 16 extents.

84

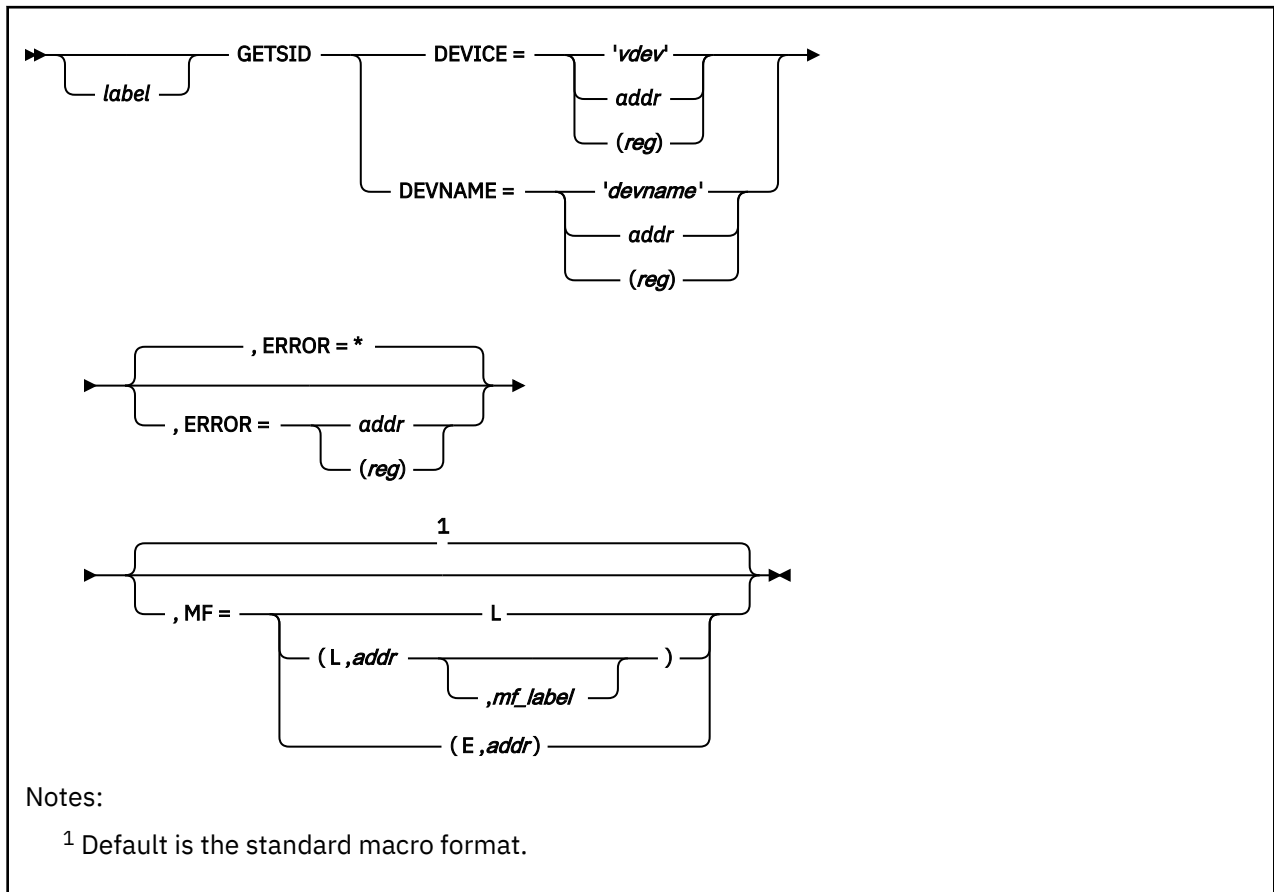
Attempt to write a file on an OS or DOS formatted minidisk.

99

A required system resource is unavailable for one of the following reasons:

- There is insufficient virtual storage for the file pool server.
- The file pool server is unavailable.
- File is in migrated status and DFSMS is not enabled.

GETSID



Purpose

Use the GETSID macro to store in register 1 the Subsystem-Identification word (SID) for a device number or name.

Parameters

Required Parameters:

DEVICE=

specifies the virtual device number of the device for which the Subsystem-Identification (SID) is to be stored in register 1. The virtual device must be in the virtual device configuration. Acceptable values are:

'vdev'

is a quoted string of up to 4 characters designating the virtual device number of the device.

addr

is the address of a fullword containing the device number.

(reg)

specifies a register containing the device number. Valid registers are 2-12 enclosed in parentheses.

DEVNAME=

specifies the symbolic name of the device for which the SID is stored in register 1. The device name must be one of the standard CMS device names (for example, TAP1) or have been defined by a previous HNDIO or HNDINT macroinstruction. Acceptable values are:

'devname'

is a quoted string of up to 4 characters designating the symbolic device name of the device.

addr

is the address of a four-byte character string containing the symbolic device name.

(reg)

specifies a register containing the address of the symbolic device name (a four-byte character string). Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. All I/O instructions that reference a subchannel require register 1 to contain the SID that corresponds to a previously-defined virtual device or CMS device name. Because the GETSID macro returns the SID in register 1, it should be invoked before issuing the first I/O instruction.

For example, the following sequence of instructions starts I/O to device TAP1.

```
GETSID DEVNAME='TAP1'      * Get SID of TAP1 in R1
SSCH  CCWS                 * Start I/O using the CCWs at loc CCWS.
```

Return Codes

If an error occurs, register 15 contains one of the following return codes:

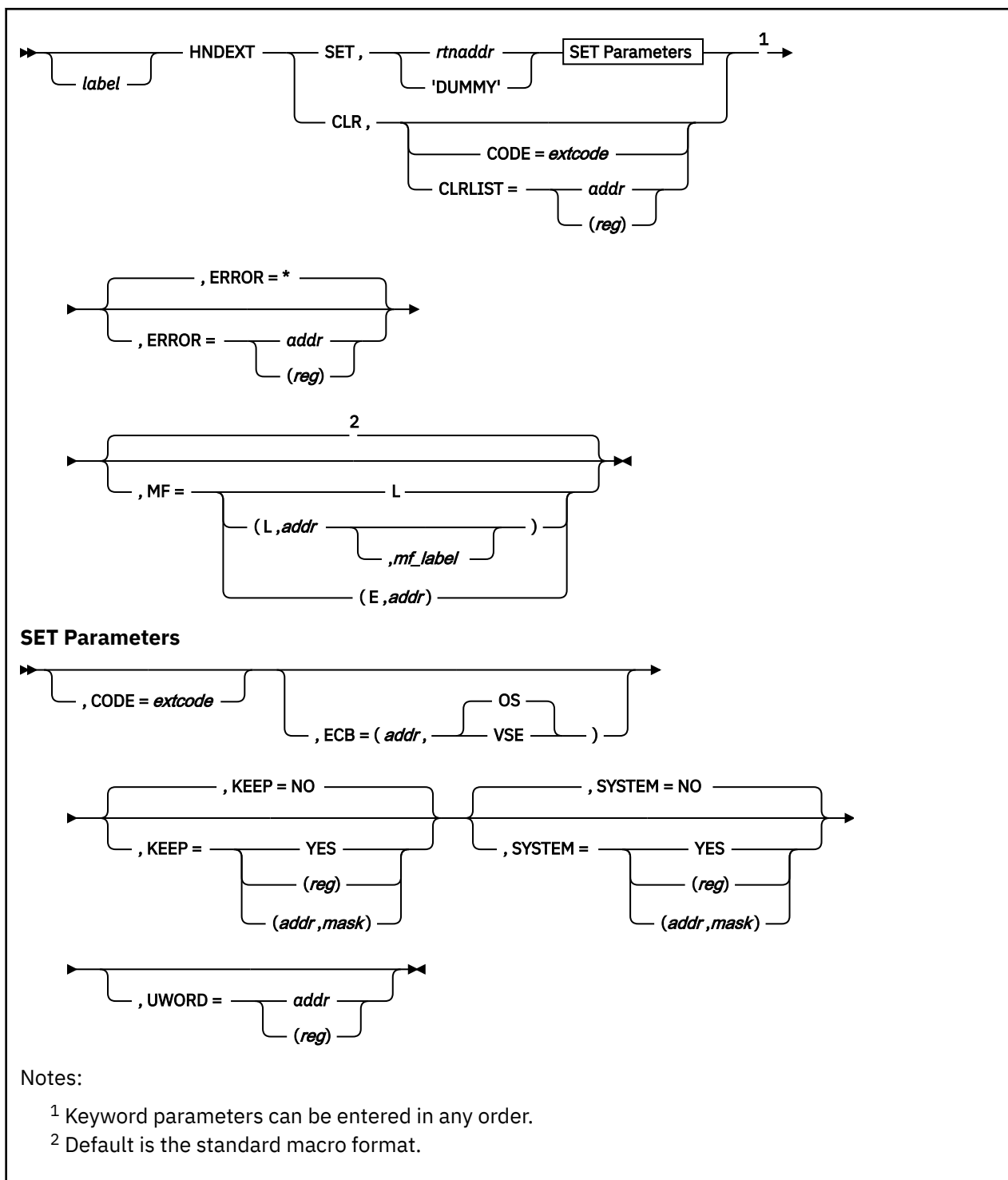
Code**Meaning****4**

The specified device was not found.

16

Reserved Not used.

HNDEXT

**Purpose**

Use the HNDEXT macroinstruction to create or delete external interrupt handlers. You can create interrupt handlers for specific interrupt codes and you can establish a default external interrupt handler to process interrupts that do not have specific handlers.

Parameters

Required Parameters:

SET

defines an external interrupt handler. The CODE parameter implies the handler is for a specific interrupt code. Omitting the CODE parameter defines a default external interrupt handler.

CLR

clears the interrupt handler routine for the specified code. Interrupt handling routines should not issue HNDEXT CLR. If you do not specify a code to be cleared, the default handler is cleared.

rtnaddr

specifies the address of the interrupt handler.

'DUMMY'

specifies that there is no handling routine for the defined interrupt code. When you specify 'DUMMY', you must also specify ECB. If the interrupt for which you specify 'DUMMY' occurs, the first-level interrupt handler posts the ECB for this code. Use the WAITECB macro to determine when this happens.

CODE=*extcode*

specifies the external interrupt code you wish to handle or clear. Codes may be specified in the range of X'0000' to X'FFFE'. If you specify interrupt code X'0000' for SET, HNDEXT creates a default external interrupt handler.

CLRLIST=

clears the interrupt handlers listed at the specified address. The list should contain 2-byte external interrupt codes for the handlers you want cleared. It should end with an 8-byte fence of X'FF'.

Acceptable values are:

addr

specifies the list of handlers to be cleared as an address.

(*reg*)

specifies the register containing the address of the list of handlers.

Optional Parameters:

label

is an optional assembler label for the statement.

ECB=

specifies the address and format (OS or VSE) of an event control block (ECB) to be posted in connection with each such interrupt. Acceptable values are:

addr

specifies the address of an event control block.

OS

specifies the event control block in OS format. This is the default value.

VSE

specifies the event control block in VSE format.

(To suppress posting, the second-level interrupt handler can issue a return code of 4.)

If you specify the 'DUMMY' and ECB parameters, CMS posts the event control block as soon as the interrupt is detected. To suspend execution until the specified event control block is posted, a program can issue the WAITECB macro.

For more information on OS and VSE format event control blocks, see ["WAITECB" on page 432](#).

KEEP=

specifies whether the interrupt handler is cleared at end-of-command. Acceptable values are:

NO

specifies that the interrupt handler is cleared. This is the default value.

YES

specifies that the interrupt handler is not cleared. If you issue KEEP=YES, make sure the handler routine itself survives end-of-command processing.

(reg)

specifies the register that contains the value for KEEP. The macro checks the value of the specified register and, if it is 0, sets KEEP to NO. If the register contains a nonzero value, the macro sets KEEP to YES.

(addr,mask)

defines a single bit in storage that sets the value of the KEEP parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then KEEP is set to NO. If the bit is 1, then KEEP is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the KEEP parameter as

```
KEEP=(APPFLAG,X'80')
```

To set the value of the KEEP parameter at assembly time, specify KEEP=YES or KEEP=NO. To set the value at execution time, specify KEEP=(*reg*) or KEEP=(*addr,mask*).

SYSTEM=

specifies whether the handler survives ABEND processing. Acceptable values are:

NO

specifies that the handler does not survive. This is the default value.

YES

specifies that the handler does survive. If you issue SYSTEM=YES, make sure the handler routine itself survives abend processing. (For example, you can define the handler as a nucleus extension.)

(reg)

specifies the register that contains the value for SYSTEM. The macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the SYSTEM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,mask*).

End-of-command processing follows abend processing; therefore, if you want interrupt handlers to survive abend processing and end-of-command processing, specify SYSTEM=YES **and** KEEP=YES.

UWORD=

specifies an optional fullword available to the handling routine. The address of the UWORD is contained in register 0 when the handler routine is invoked. Acceptable values are:

addr

specifies the address of the UWORD.

(reg)

specifies a register that contains the address of the UWORD. Valid registers are 2-12 enclosed in parentheses.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. External interrupt handler routines are called in the addressing mode of the program that issues the HNDEXT macro.
2. External interrupt handler routines are called with the PSW disabled for external and I/O interrupts and in storage key 0.
3. In an XC virtual machine, your second-level interrupt handler always receives control in primary space address translation mode and always must return control to CMS in primary space mode.
4. You are responsible for providing the proper entry and exit linkage for your interrupt handling routine. When your program receives control, the register contents are as follows:

Register**Contents****R0**

Address of the user word (UWORD) specified on the HNDEXT macro.

R1

Address of the area containing the state of the machine at the time of interrupt. See [“Purpose” on page 188](#) for a description of the EXTUAREA macro, which maps this area.

R2-R11

Unspecified

R12

Handling routine entry address

R13

A pointer to the user save area (label EXTUSAVE) within the EXTUAREA

R14

Return address

R15

Handling routine entry address.

Your routine must return control to the address in register 14, and must store one of the following return codes in register 15:

- Zero (0)—Indicates the second-level handler is through handling the interrupt and the first-level handler should post the ECB, if one was specified.
 - Four (4)—Indicates the second-level handler has completed, and the first-level handler should **not** post the ECB.
 - Eight (8)—Indicates that CMS passes the interrupt to a user-specified default handler, if one exists; otherwise CMS passes the interrupt to the system default handler.
5. The HNDEXT SET function cannot define handlers for codes that already have handlers. To define a new handler for an interrupt code, you must clear the existing one and then define a new one.
 6. CMS issues HNDEXT SET for all of the following external interrupt codes:

X'1202'

VCPU SIGP - use CMS Multitasking instead.

X'4000'

IUCV - use HNDIUCV instead.

X'0080'

Clock comparator or interval timer - use Timer Services or OS timer support instead.

X'2603'

VM Data Spaces -- use CSLs for Data Spaces instead.

X'2004'

Time zone change interrupt -- use Event Services instead.

Customers should use higher-level interfaces for these purposes instead of trying to intercept these external interrupts directly.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

2

A handler already exists for the specified interrupt code. Before you can define a new handler, use HNDEXT CLR to delete the existing handler.

3

The handler to be cleared was not found. If you specify the CLRLIST parameter and a handler definition in the list was not previously SET, the operation terminates with a return code of 3. Register 1 contains the code of the nonexistent handler. HNDEXT CLR processing terminates when the first invalid code in the list is detected; handlers specified after the code for the nonexistent handler are not cleared.

11

Parameter list error; invalid function specified. The function was not SET or CLR.

12

Parameter list error; invalid CODE value specified. The CODE must be in the range of X'0000' to X'FFFE'.

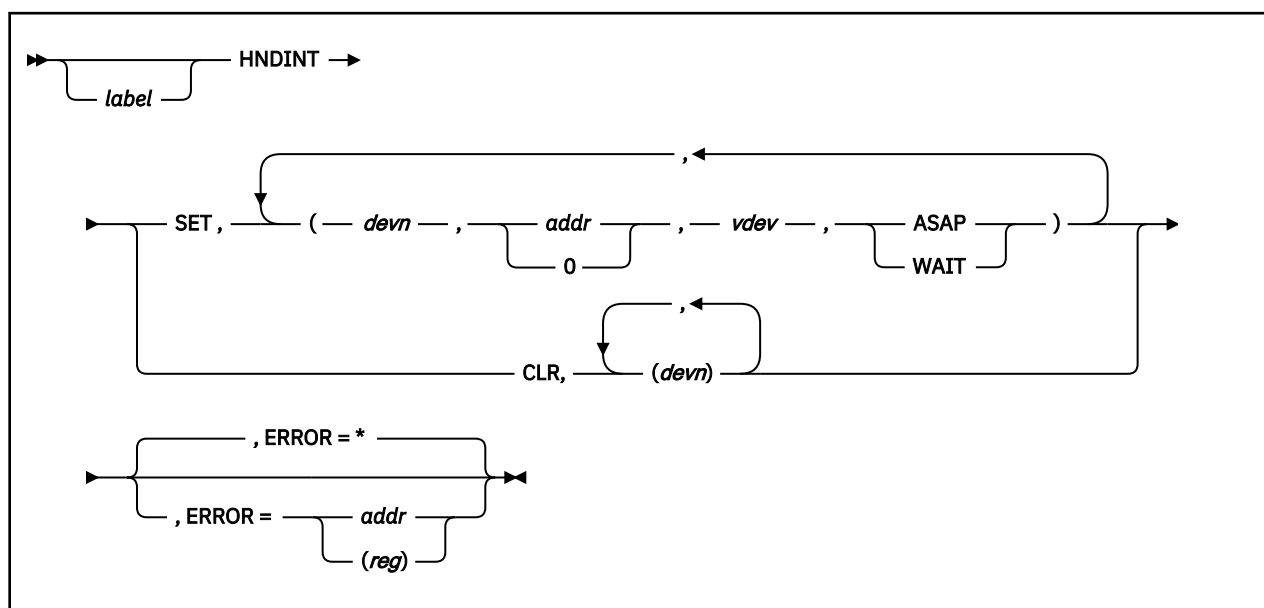
13

Parameter list error; 'DUMMY' was specified but no ECB parameter was provided.

14

Parameter list error; CLR of default handler specified with other handler entries. A parameter list was built indicating the default entry (all 0s in the parameter list) but was not followed with a fence.

HNDINT



Purpose

Use the HNDINT macroinstruction to trap interrupts for a specified I/O device. To receive device-specific information, use the HNDIO macro. HNDIO is also recommended for use in new programs.

Parameters

Required Parameters:

SET

specifies that you want to trap interrupts for the specified device.

devn

specifies a 4-character symbolic name for the device whose interrupts are to be trapped.

addr

specifies the address of the interrupt handler routine. An address of 0 indicates that interrupts for the device are to be ignored.

vdev

specifies the virtual device number, in hexadecimal, of the device whose interrupts are to be trapped.

ASAP

specifies that the routine at *addr* receive control as soon as the interrupt occurs.

WAIT

specifies that the routine at *addr* receives control after the WAITD macro is issued for the device.

CLR

specifies that you no longer want to trap interrupts for the specified device.

Note: Do not issue HNDINT CLR from within the interrupt handling routine.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Usage Notes

1. The specified handler routine runs in the addressing mode of the program that issues the HNDINT macro.
2. In an XC virtual machine, your second-level interrupt handler always receives control in primary space address translation mode and always must return control to CMS in primary space mode.
3. You are responsible for establishing proper entry and exit linkage for your interrupt handling routine. When your routine receives control, the significant registers contain:

Register**Contents****R0-R1**

I/O old PSW—This is a reconstructed PSW in basic control (BC) format. A program may extract the real I/O old PSW from the INTBLOK.

R2-R3

Channel status word (CSW)—This is a reconstructed CSW. A program may extract the real I/O status from the INTBLOK's IRB contents.

R4

Address of interrupting device

R5-R6

Contain zeros

R7-R13

Unspecified

R14

Return address

R15

Entry point address.

Your routine must return control to the address in register 14 and store a return code in register 15 to indicate whether processing is complete. A zero (0) in register 15 means that you are finished handling the interrupt; any nonzero return code indicates that you expect another interrupt.

Note: Register 13 does not point to a save area for your use.

4. When your interrupt handler receives control, all I/O interrupts and external interrupts are disabled. Your handler should not perform any I/O operations.
5. For I/O operations to a 3270-type device, use the CONSOLE macro rather than the HNDIO or HNDINT macros to handle interrupts. (For more information on interrupt handling, see *z/VM: CMS Application Development Guide for Assembler*.) Exit routines specified by the CONSOLE macro support multiple applications for a 3270-type display device, while HNDINT supports only one. Furthermore, HNDINT interrupt routines override CONSOLE only in the case of an unsolicited interrupt when no one issued a CONSOLE WAIT.
6. I/O operations initiated by some forms of the DIAGNOSE instruction do not produce I/O interrupts and are not trapped by HNDINT. If the I/O operation initiated by DIAGNOSE does produce I/O

interrupts for a device specified for HNDINT, then HNDINT traps the interrupts. For more information about I/O operations initiated by the DIAGNOSE instruction, see [z/VM: CP Programming Services](#).

7. You can use one HNDINT macroinstruction to define interrupt handling routines for more than one device. The argument list for each device must be enclosed in parentheses and separated from the next list by a comma.
8. If you specify WAIT, your interrupt handler receives control when a WAITD macroinstruction that specifies the same symbolic device name is issued. If the WAITD macroinstruction is issued before the interrupt occurs, then your interrupt handler receives control immediately after the interrupt is received.
9. If you specify HNDINT with a different DEVNAME to create an additional handler for a specific device, the new handler is inactive until all existing handlers are deleted. If you specify HNDIO with a different DEVNAME to create an additional handler for a specific device, the new handler becomes active immediately.
10. If a device for which an interrupt handler is defined is subsequently detached (using a CP DETACH command) or redefined at a new address (using a CP DEFINE or REDEFINE command), then the interrupt handler is cleared.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

1

An invalid device number (*vdev*) or interrupt handling routine address (*addr*) was specified.

2

Trap item replaces another of same device name.

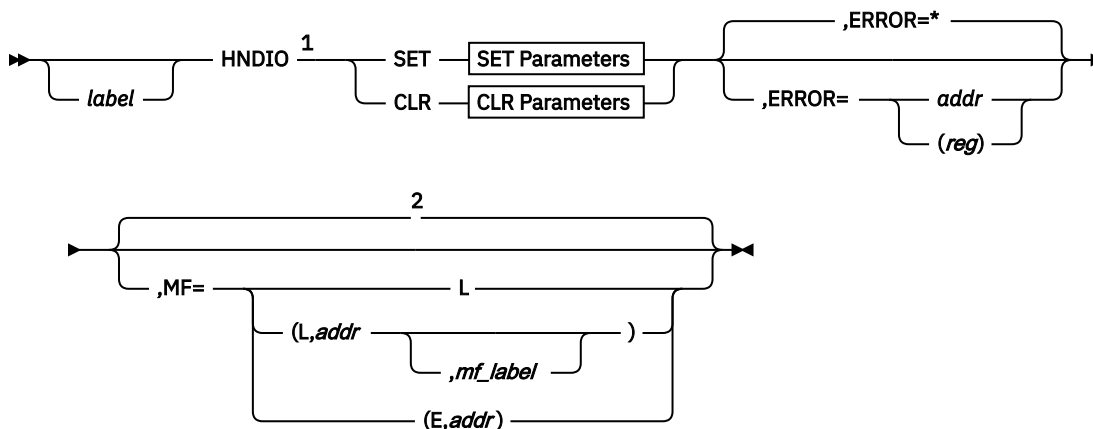
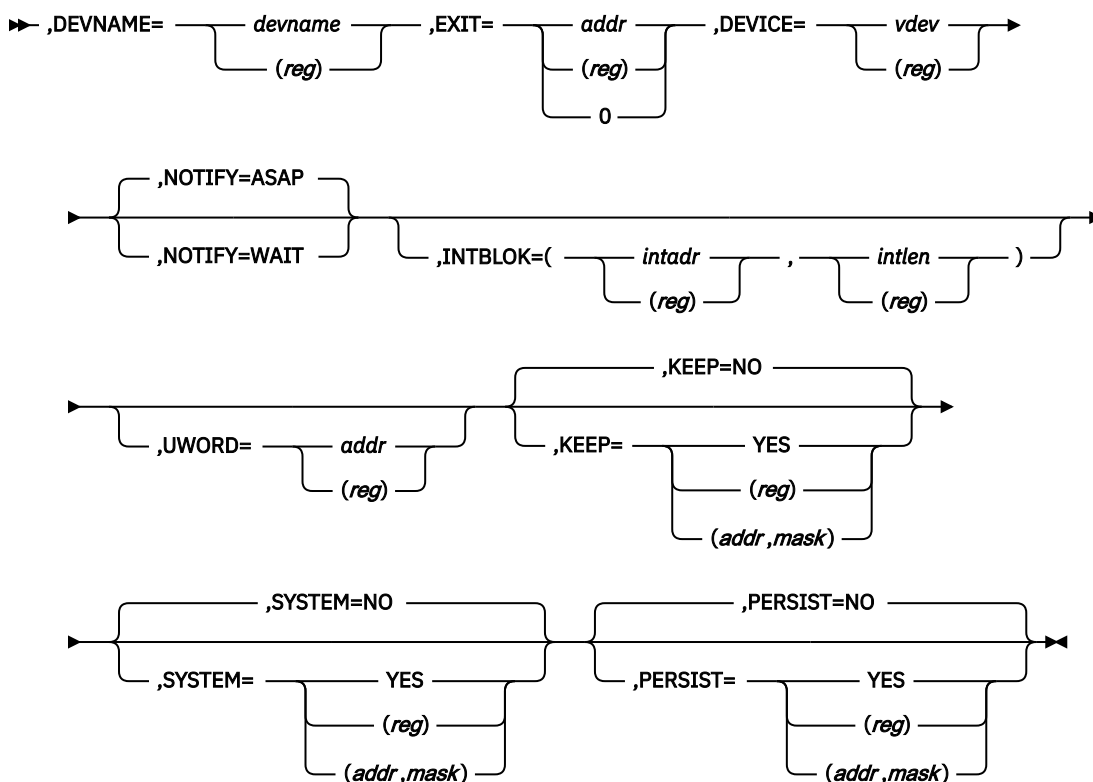
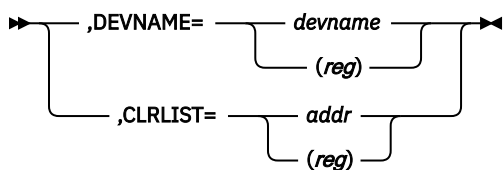
3

An attempt was made to clear a nonexistent interrupt.

104

An out of storage condition occurred during processing.

HNDIO

**SET Parameters****CLR Parameters****Notes:**

- ¹ Keyword parameters can be entered in any order.
- ² Default is the standard macro format.

Purpose

Use the HNDIO macro to handle interrupts and to obtain complete I/O interrupt status for specified I/O devices.

Parameters

Required Parameters:

SET

specifies that you want to handle interrupts for the specified device.

DEVNAME=

specifies a 4-character symbolic name for the device whose interrupts are to be handled (after SET) or whose interrupts are to be cleared (after CLR). Acceptable values are:

devname

is a 4-character symbolic name for the device.

(reg)

specifies a register that contains the address of the 4-character symbolic name for the device.

EXIT=

specifies the address of the handling routine entry point that receives control when the interrupt occurs. By default, interrupts are disabled when the handling routine assumes control. Acceptable values are:

addr

specifies the address of the handling routine entry point.

(reg)

specifies a register that contains the address of the handling routine entry point.

0

causes CMS to ignore interrupts for the specified device.

DEVICE=

specifies in hexadecimal the virtual device number of the device that you handle interrupts for. Acceptable values are:

vdev

specifies the virtual device number.

(reg)

specifies a register that contains the device address.

CLR

clears an interrupt handler routine for the specified device.

Note: Do not issue HNDIO CLR from within the interrupt handling routine.

CLRLIST=

clears the interrupt handlers listed at the specified address. Acceptable values are:

addr

specifies the address of the list.

(reg)

specifies a register that contains the address of the list. See the Usage Notes for examples.

Optional Parameters:

label

is an optional assembler label for the statement.

NOTIFY=

specifies when your program is notified of the interrupt. Acceptable values are:

ASAP

passes control to the handling routine as soon as the interrupt occurs. This is the default.

WAIT

passes control to the handling routine after the WAITD macro is issued for the device.

INTBLOK=

specifies the address of a user-provided area where, prior to transferring control to the specified handling routine, CMS copies information about the interrupt. INTBLOK returns information from the IRB (interrupt response block). Acceptable values are:

intadr

specifies the INTBLOK address

(reg)

specifies the INTBLOK address as a register that contains the address.

intlen

specifies the INTBLOK length as a label

(reg)

specifies the INTBLOK length as a register that contains the length.

The INTBLOK DSECT maps this area and contains a label, INTBLKSZ, which indicates the required size of the INTBLOK.

UWORD=

specifies an optional fullword available to the handling routine. The address of the UWORD is contained in general register 6 when the handler routine is invoked. Acceptable values are:

addr

specifies the address of the UWORD.

(reg)

specifies a register that contains an address of the UWORD. Valid registers are 2-12 enclosed in parentheses.

KEEP=

specifies whether the interrupt handler is cleared at end-of-command. Acceptable values are:

NO

specifies that the interrupt handler is cleared. This is the default value.

YES

specifies that the interrupt handler is not cleared. If you issue KEEP=YES, make sure the handler routine itself survives end-of-command processing.

(reg)

specifies the register that contains the value for KEEP. The macro checks the value of the specified register and, if it is 0, sets KEEP to NO. If the register contains a nonzero value, the macro sets KEEP to YES.

(addr,mask)

defines a single bit in storage that sets the value of the KEEP parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then KEEP is set to NO. If the bit is 1, then KEEP is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the KEEP parameter as

```
KEEP=(APPFLAG,X'80')
```

To set the value of the KEEP parameter at assembly time, specify KEEP=YES or KEEP=NO. To set the value at execution time, specify KEEP=*(reg)* or KEEP=*(addr,mask)*.

SYSTEM=

specifies whether the handler survives abend processing. Acceptable values are:

NO

specifies that the handler does not survive. This is the default value.

YES

specifies that the handler does survive. If you issue SYSTEM=YES and KEEP=YES, make sure that the trap routines themselves survive abend processing.

(reg)

specifies the register that contains the value for SYSTEM. The macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the SYSTEM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,mask*).

End-of-command processing follows abend processing; therefore, if you want interrupt handlers to survive abend processing and end-of-command processing, specify SYSTEM=YES **and** KEEP=YES.

PERSIST=

specifies whether the handler survives machine check processing for DETACH/DEFINE of the device. Acceptable values are:

NO

specifies that the handler does not survive. This is the default value.

YES

specifies that the handler does survive. If you issue PERSIST=YES and KEEP=YES, make sure that the handler routine itself survives end of command processing.

(reg)

specifies the register that contains the value for PERSIST. The macro checks the value of the specified register and, if it is 0, sets PERSIST to NO. If the register contains a nonzero value, the macro sets PERSIST to YES.

(addr,mask)

defines a single bit in storage that sets the value of the PERSIST parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then PERSIST is set to NO. If the bit is 1, then PERSIST is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the PERSIST parameter as

```
PERSIST=(APPFLAG,X'80')
```

To set the value of the PERSIST parameter at assembly time, specify PERSIST=YES or PERSIST=NO. To set the value at execution time, specify PERSIST=(*reg*) or PERSIST=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. I/O interrupt handler routines are called in the addressing mode of the program that issues the HNDIO macro.
2. In an XC virtual machine, your second-level interrupt handler always receives control in primary space address translation mode and always must return control to CMS in primary space mode.
3. You must provide the proper entry and exit linkage for your interrupt handling routine. With two exceptions, handlers defined by HNDIO and HNDINT follow the same linkage conventions. For HNDIO, register 5 points to the INTBLOK and register 6 contains the UWORD, if specified. For interrupt handlers created by HNDINT, registers 5 and 6 contain zeros.

Register

Contents

R0-R1

I/O old PSW—This is a reconstructed PSW in basic control (BC) format. A program may extract the real I/O old PSW from the INTBLOK.

R2-R3

Channel status word (CSW)—This is a reconstructed CSW. A program may extract the real I/O status from the INTBLOK's IRB contents.

R4

The device number. A program may obtain more complete device information from the INTBLOK.

R5

A pointer to the INTBLOK specified on the HNDIO macro—This area contains the virtual machine interrupt information, such as the I/O old PSW, I/O interrupt code, and the IRB. If you do not specify the INTBLOK parameter, then register 5 is set to 0.

R6

The user word (UWORD)—If the UWORD parameter is not specified, then this register is set to 0.

R7-R11

Unspecified.

R12

Interrupt handling routine entry address.

R13

Pointer to a 24-word save area provided by the first-level handler.

R14

Return address.

R15

Interrupt handling routine entry address.

The routine must return control to the address in register 14, and indicate by return code in register 15 whether processing is complete. A zero (0) return code indicates that the second-level handler is complete; any nonzero return code indicates that the second-level handler expects another interrupt before processing is complete.

4. When your interrupt handler receives control, all I/O interrupts and external interrupts are disabled. Your interrupt handling routine should remain in this state through its processing. It should not perform any I/O operations, nor should it issue a HNDIO CLR or a HNDINT CLR for the device associated with the handling routine.
5. The first-level interrupt handler issues the TSCH (test subchannel) instruction to clear the interrupt. If the TSCH fails, the INTFAIL flag is set; the related IRB information is invalid for this interrupt.
6. The CLRLIST parameter clears interrupt handlers for a list of devices. Specify each device in the list as a 4-character symbolic name. To make sure that only the specified device interrupt handlers are cleared, end the list with a fence (8X'FF').

If a routine attempts to clear a handler for an unspecified device, the invalid device name is returned in register 1. Devices prior to the invalid device are cleared; devices following the invalid device are not.

The following example shows how to use the CLRLIST parameter to clear handling routines DSK1, DSK2, DSK3, and DSK4:

```

HNDIO CLR,CLRLIST=LISTADDR
.
.
.
LISTADDR DS 0H
          DC CL4'DSK1'
          DC CL4'DSK2'
          DC CL4'DSK3'
          DC CL4'DSK4'
          DC 8X'FF'
```

7. If you specify HNDIO with a different DEVNAME to create an additional handler for a specific device, the new handler processes subsequent interrupts. If you specify HNDINT with a different DEVNAME to create an additional handler for a specific device, the new handler is inactive until all existing handlers are deleted.
8. For I/O operations to a 3270-type device, use the CONSOLE macro rather than the HNDIO macro to handle interrupts. (For more information on interrupt handling, see the [z/VM: CMS Application Development Guide for Assembler](#).)
9. If a device for which an interrupt handler is defined is subsequently detached (using a CP DETACH command) or redefined at a new address (using a CP DEFINE or REDEFINE command), then the interrupt handler is cleared unless PERSIST=YES has been specified.
10. If PERSIST=YES is specified and the user DETACHes the current device and reDEFINEs the device address as a different type of device, then the exit routine must be able to handle the new device type or unexpected results may occur.
11. If PERSIST=YES and KEEP=YES are specified, then the user must reissue the HNDIO macro with a CLR parameter to clear the exit for the specified device when the exit is no longer wanted to receive interrupt control. PERSIST=YES and KEEP=YES disable the normal system actions to automatically clear the exit routine.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

1

An invalid device number (*vdev*) or interrupt handling routine address (*addr*) was specified.

- 2** Trap item replaces another of same device name.
- 3** An attempt was made to clear a nonexistent interrupt.
- 4** The device name specified is already in use by another device address.
- 11** The device name was specified as all blanks or 0.
- 15** INTBLOK specification error.
- 104** An out of storage condition occurred during processing.

HNDIUCV

Purpose

Use the HNDIUCV macro to start or end a program's IUCV (Inter-User Communications Vehicle) or APPC/VM (Advanced Program-to-Program Communication/VM) environment.

The basic functions of the HNDIUCV macro are:

HNDIUCV CLEAR

Removes an APPC/VM program name from the list of APPC/VM programs that are active in CMS.

HNDIUCV HOLD

Temporarily places private resource connection requests on a CMS queue.

HNDIUCV REPLACE

Replaces the exit address and UWORD for APPC/VM programs that have been declared to CMS.

HNDIUCV RESUME

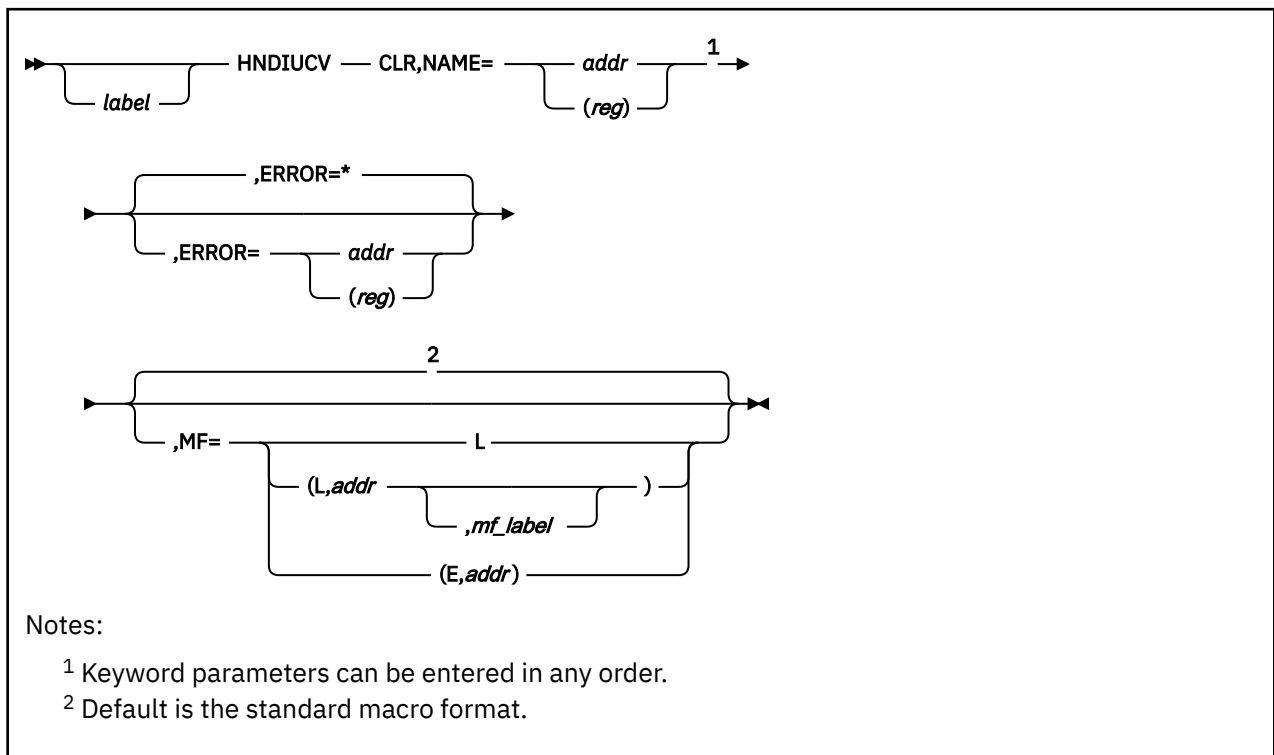
Releases previously-held private resource connection requests from a CMS queue.

HNDIUCV SET

Declares an APPC/VM program name to CMS.

For more information on how to use the HNDIUCV macro, see [*z/VM: CMS Application Development Guide for Assembler*](#).

HNDIUCV CLR (Clear)



Purpose

Use the CLR (Clear) function to remove an APPC/VM program from the list of active APPC/VM programs in CMS. This function should be issued when the program no longer wishes to do any more APPC/VM communications.

CLR severs any paths associated with this program. The IPUSER field of the APPCVM SEVER parameter list is set to binary ones to indicate the SEVER was done by CMS.

Parameters

Required Parameters:

CLR

Removes an APPC/VM program name from the list of APPC/VM programs that are active in CMS.

NAME=

specifies the name of the APPC/VM program in CMS. This name must be the same as a name previously specified on an HNDIUCV SET.

addr

specifies the address of an 8 character symbolic name.

(reg)

specifies a register that contains the address of an 8 character symbolic name.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr
passes control to the specified address.

(reg)
passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=
specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L
specifies the list format.

(L,addr,mf_label)
specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)
specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If protected (SYNCLVL=SYNCPT) conversations are deallocated as a result of executing HNDIUCV CLR, then the work units associated with the conversations should be rolled back before doing any other processing on those work units. See the *Synchronizing Updates to Multiple Resources* section in [z/VM: CMS Application Development Guide for Assembler](#) for a detailed discussion of when the work unit may need to be rolled back. For information about protected conversations and the Coordinated Resource Recovery (CRR) facility in CMS, see [z/VM: CMS Application Development Guide](#).

Return Codes

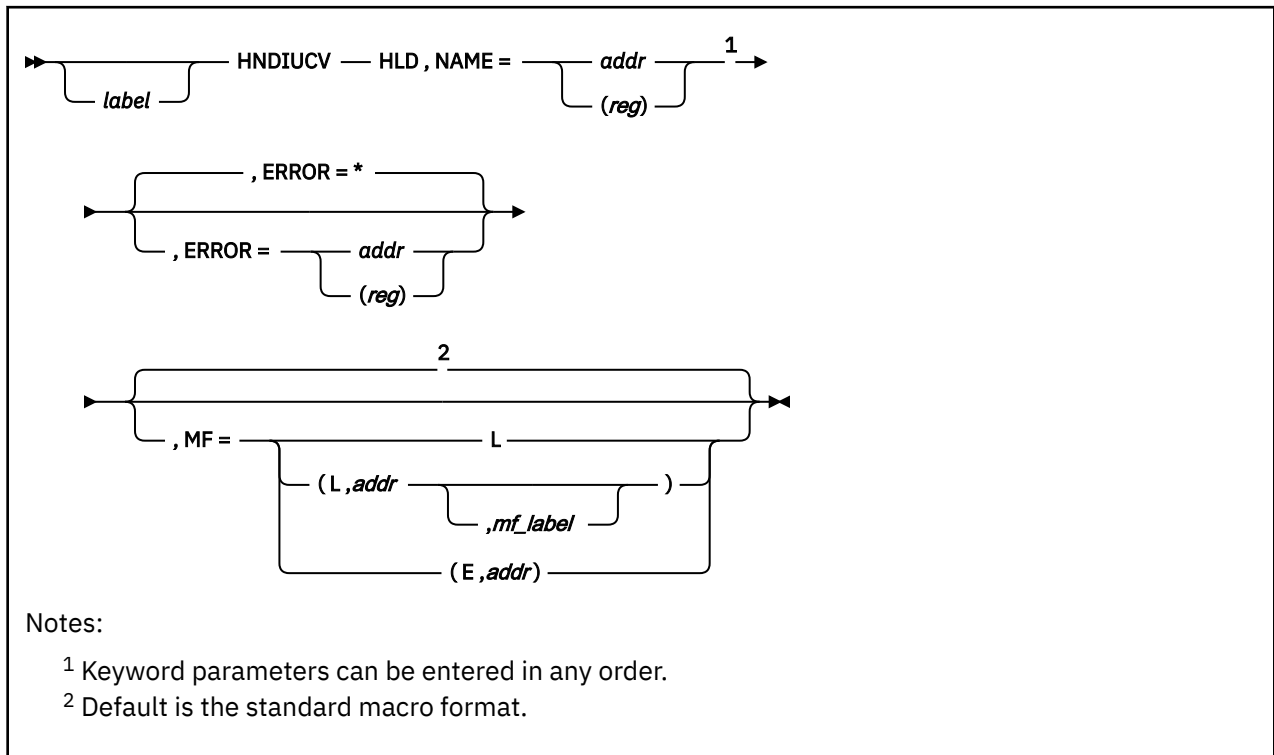
Upon completion of the HNDIUCV CLR function, register 15 contains either:

- A 5-digit reason code returned by a CSL routine that was called by HNDIUCV CLR processing. These are described in [z/VM: CMS and REXX/VM Messages and Codes](#), or
- One of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	HNDIUCV CLR completed successfully.
X'08'	8	No HNDIUCV SET has been issued for the specified program name.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'28'	40	An invalid HNDIUCV function was specified; must be SET, CLR, REP, HLD, or RES.
X'30'	48	The IUCV RTRVBFR failed, as indicated by CP.
X'3C'	60	!CMS cannot issue the HNDIUCV CLR function. (!CMS is a reserved name for CMS. CMS uses !CMS as a user ID so it can use its own APPC/VM support.)

Hex Code	Decimal Code	Meaning
X'68'	104	Out of Storage.
X'C8' + xx	2dd	An error was encountered in getting CMS free storage. The xx is the hexadecimal return code from CMSSTOR. The dd is the decimal equivalent of this return code.
X'3E8' + 1xxx	1ddd	While trying to SEVER all of the program's paths, an APPCVM SEVER error occurred. The xxx is the IPRCODE field that was returned by the APPCVM SEVER to aid in diagnosing the error. The ddd is the decimal equivalent of this IPRCODE. For more information on the APPCVM SEVER return codes, see z/VM: CP Programming Services .

HNDIUCV HLD (Hold)



Purpose

Use the HLD (Hold) function in a private resource manager program to queue private resource connection requests for the program without presenting them to the interrupt-processing exit routine.

After a HLD is issued for a program name, CMS severs any local or global resource connection requests for that program name.

HLD does not affect active paths, other APPC/VM or IUCV functions in CMS, or private resource connection requests that were previously queued.

Parameters

Required Parameters:

HLD

Temporarily places private resource connection requests on a CMS queue.

NAME=

specifies the name that identifies the program on this APPC/VM path. This name must have been previously specified on an HNDIUCV SET.

When this program issues the CMSIUCV macro to perform an APPC/VM function, the NAME parameter specified on the CMSIUCV macro must be the same as the one specified here.

addr

specifies the address of an eight-character symbolic name.

(reg)

specifies a register that contains the address of an eight-character symbolic name.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

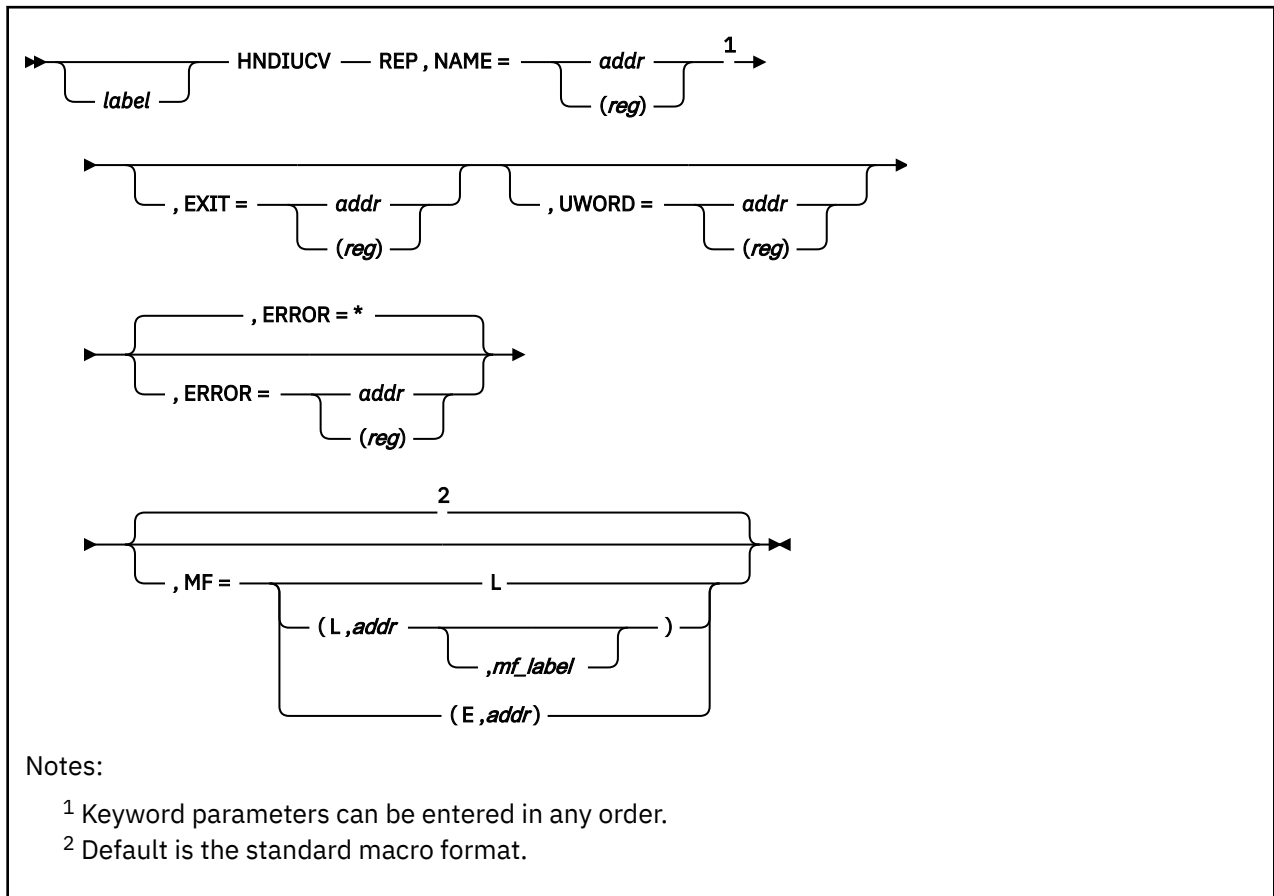
Return Codes

Upon completion of the HNDIUCV HLD function, register 15 contains either:

- A 5-digit reason code returned by a CSL routine that was called by HNDIUCV HLD processing. These are described in *z/VM: CMS and REXX/VM Messages and Codes*, or
- One of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	HNDIUCV HLD completed successfully.
X'08'	8	No HNDIUCV SET has been issued for this program.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'28'	40	An invalid HNDIUCV function was specified; must be SET, CLR, REP, HLD, or RES.
X'30'	48	The IUCV DCLBFR CONTROL=YES failed, as indicated by CP.
X'3C'	60	!CMS cannot issue the HNDIUCV HLD function. (!CMS is a reserved name for CMS. CMS uses !CMS as a user ID so it can use its own APPC/VM support.)
X'C8' + xx	2dd	An error was encountered in getting CMS free storage. The xx is the hexadecimal return code from CMSSTOR; the dd is the decimal equivalent of this return code.

HNDIUCV REP (Replace)



Purpose

Use the REP (Replace) function to replace the currently defined exit address or UWORD field for a specified program. Only the parameters specified are replaced.

The REP function replaces only the general exit address or UWORD set up by your program by the HNDIUCV SET function. If your program had previously issued any CMSIUCV CONNECTs and had the EXIT address or UWORD default to the HNDIUCV SET's EXIT and UWORD, the HNDIUCV REP function does not replace the path specific EXIT or UWORD set up by the CMSIUCV function. The EXIT and UWORD remain as established when the CMSIUCV function was issued.

Parameters

Required Parameters:

REP

Replaces the exit address and UWORD for APPC/VM programs that have been declared to CMS.

NAME=

specifies the name that identifies your program. This name must have been previously specified on an HNDIUCV SET.

When this program issues the CMSIUCV macro to perform an APPC/VM function, the NAME parameter specified on the CMSIUCV macro must be the same as the one specified here.

addr

specifies the address of an 8 character symbolic name.

(reg)

specifies a register that contains the address of an 8 character symbolic name.

Optional Parameters:

label

is an optional assembler label for the statement.

EXIT=

specifies the address of an exit routine that receives control when an APPC/VM connection pending interrupt occurs for this program. To activate this exit, the connecting program must specify the same name for RESID on the APPCVM CONNECT as the NAME parameter specified for this target program.

The exit address is the default address associated with any path owned by this program. This default address receives control if an APPC/VM external interrupt is presented to the program on a path that does not have a exit address specifically established. Two conditions could cause the default exit address to get control:

1. A connect pending interrupt had previously occurred on the path, but the program has not yet issued CMSIUCV ACCEPT.
2. A program established a path using CMSIUCV CONNECT or CMSIUCV ACCEPT, but did not specify the EXIT parameter.

The APPC/VM exit routine is called in the addressing mode of the program issuing this HNDIUCV REP function.

addr

specifies the address of the exit routine.

(reg)

specifies a register that contains the address of the exit routine.

When the program's APPC/VM external interrupt routine is given control, all interrupts are disabled. The exit routine is responsible for providing proper entry and exit linkage for its APPC/VM external interrupt handling routine. The exit routine:

- Should not enable itself for any type of interrupts
- Should not perform any I/O operations, because all interrupts are disabled
- Must save and restore the return address in register 14.

When the routine receives control, the significant registers contain:

Register	Contents																								
0	UWORD Field																								
1	<p>If the pending interrupt is for a private resource connection, register 1 contains a X'00'.</p> <p>If a connection to a global or local resource, register 1 points to a SAVEAREA in this format:</p> <table><tr><th>Label</th><th>Displacement Dec</th><th>Hex</th><th>Contents</th></tr><tr><td>GRS</td><td>0</td><td>0</td><td>General purpose registers 0-15 at the time of the interrupt.</td></tr><tr><td>FRS</td><td>64</td><td>40</td><td>Floating point registers 0-7 at the time of the interrupt.</td></tr><tr><td>PSW</td><td>96</td><td>60</td><td>External Old PSW at the time of the interrupt.</td></tr><tr><td>UAREA</td><td>104</td><td>68</td><td>Register save area for exit routine's use.</td></tr><tr><td>END</td><td>176</td><td>B0</td><td>End of save area.</td></tr></table>	Label	Displacement Dec	Hex	Contents	GRS	0	0	General purpose registers 0-15 at the time of the interrupt.	FRS	64	40	Floating point registers 0-7 at the time of the interrupt.	PSW	96	60	External Old PSW at the time of the interrupt.	UAREA	104	68	Register save area for exit routine's use.	END	176	B0	End of save area.
Label	Displacement Dec	Hex	Contents																						
GRS	0	0	General purpose registers 0-15 at the time of the interrupt.																						
FRS	64	40	Floating point registers 0-7 at the time of the interrupt.																						
PSW	96	60	External Old PSW at the time of the interrupt.																						
UAREA	104	68	Register save area for exit routine's use.																						
END	176	B0	End of save area.																						
2	Address of the APPC/VM External Interrupt Buffer																								
3	Address of the connection pending extended data (if the exit is driven by a connection pending interrupt), or the address of the connection complete extended data (if the exit is driven by a connection complete interrupt).																								
4	Address of the PIP variable (if the exit is driven by a connection pending interrupt).																								

Register	Contents
13	Points to the register save area at label UAREA for use by the exit routine. (If register 1 contains a X'00', register 13 points to a standard register save area.)
14	Return address
15	Entry point address

UWORD=

specifies a fullword (user word) containing information that the invoking program can specify. CMS passes this user word to the exit routine when an interrupt is presented for this APPC/VM path. The exit routine can use this information if it desires to do so. When the exit routine receives control, register 0 contains either an address where the user word is stored (if **UWORD=addr**) or the value of a register that contains the user word (if **UWORD=(reg)**).

If a **UWORD** value is not specified on a CMSIUCV ACCEPT or CMSIUCV CONNECT macro, it defaults to the **UWORD** value specified here. If you do not specify **UWORD** here, the user word value defaults to the value specified on the HNDIUCV SET macro for this program name.

addr

specifies the address where the user word value is stored.

(reg)

specifies a register that contains the user word value.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the **ERROR=** parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the **ERROR=** parameter with the list (**MF=L**) or complex list (**MF=(L,addr,mf_label)**) macro forms.

MF=

specifies the macro form. Omitting the **MF** parameter specifies the standard format. For more information about the **MF** parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

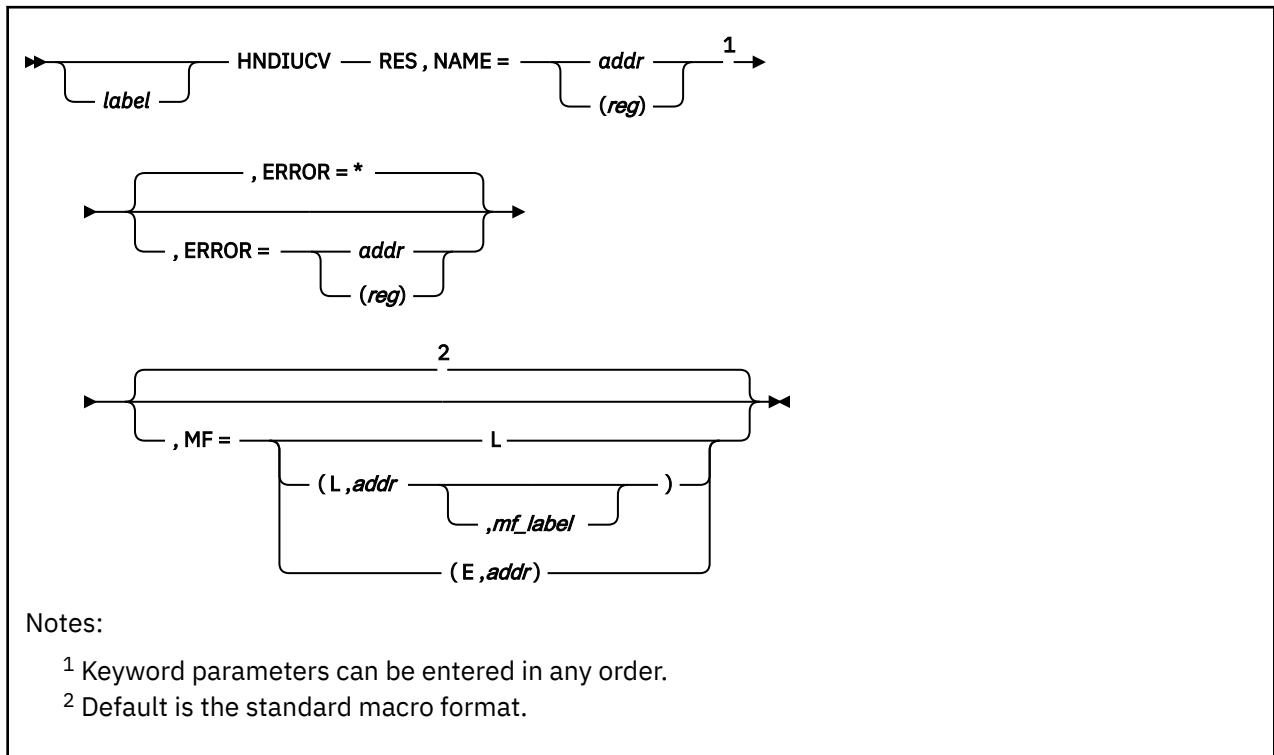
Upon completion of the HNDIUCV REP function, register 15 contains one of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	HNDIUCV REP completed successfully.

HNDIUCV REP (Replace)

Hex Code	Decimal Code	Meaning
X'08'	8	No HNDIUCV SET has been issued for this program.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'14'	20	The EXIT parameter specified an address equal to 0.
X'28'	40	An invalid HNDIUCV function was specified; must be SET, CLR, REP, HLD, or RES.
X'3C'	60	!CMS cannot issue the HNDIUCV REP function. (!CMS is a reserved name for CMS. CMS uses !CMS as a user ID so it can use its own APPC/VM support.)
X'C8' + xx	2dd	An error was encountered in getting CMS free storage. The xx is the hexadecimal return code from CMSSTOR; the dd is the decimal equivalent of this return code.

HNDIUCV RES (Resume)



Purpose

Use the RES (Resume) function in a private resource manager program to trigger the processing of queued private resource connection requests:

- Queued private resource connection requests for the specified program name are released.
- New private resource connection requests are presented to the interrupt-processing exit routine.

RES undoes the function of HOLD.

Note: HNDIUCV RES should not be issued from a user APPC/VM interrupt processing exit routine. If an APPC/VM interrupt event causes an HNDIUCV SET situation, the interrupt processing exit routine should set a flag (or post an ECB) to alert the calling program to issue the HNDIUCV RES.

Parameters

Required Parameters:

RES

Releases previously-held private resource connection requests from a CMS queue.

NAME=

specifies the name of the APPC/VM program in CMS. The program name specified on this function must have previously been specified on an HLD (hold) function.

When this program issues the CMSIUCV macro to perform an APPC/VM function, the NAME parameter specified on the CMSIUCV macro must be the same as the one specified here.

addr

specifies the address of an 8 character symbolic name.

(reg)

specifies a register that contains the address of an 8 character symbolic name.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

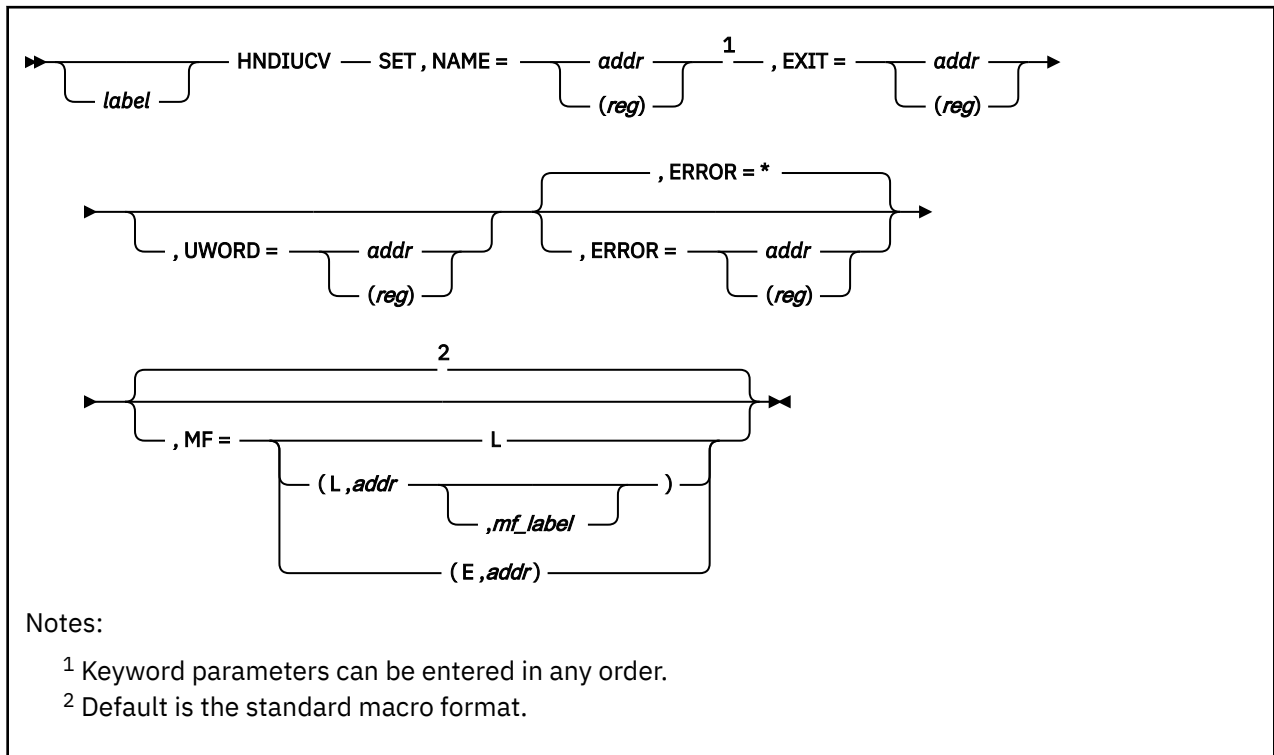
specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

Upon completion of the HNDIUCV RES function, register 15 contains one of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	HNDIUCV REP completed successfully.
X'08'	8	No HNDIUCV SET has been issued for this program.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'28'	40	An invalid HNDIUCV function was specified; must be SET, CLR, REP, HLD, or RES.
X'3C'	60	ICMS cannot issue the HNDIUCV RES function. (ICMS is a reserved name for CMS. CMS uses ICMS as a user ID so it can use its own APPC/VM support.)
X'C8' + xx	2dd	An error was encountered in getting CMS free storage. The xx is the hexadecimal return code from CMSSTOR. The dd is the decimal equivalent of this return code.

HNDIUCV SET



Purpose

Use the SET function to identify an APPC/VM program to CMS. A program must issue this SET function before issuing any functions with the CMSIUCV macro.

Here are some key values that are returned in registers:

Register

What It Contains

R0

The maximum number of possible connections for the virtual machine, upon error free completion of HNDIUCV SET.

R1

The size in bytes, rounded up to the nearest multiple of 8, of the interrupt buffer extension, upon error free completion of HNDIUCV SET.

Note: HNDIUCV SET should not be issued from a user APPC/VM interrupt processing exit routine. If an APPC/VM interrupt event causes an HNDIUCV SET situation, the interrupt processing exit routine should set a flag (or post an ECB) to alert the **calling** program to issue the HNDIUCV SET.

Parameters

Required Parameters:

SET

Declares an APPC/VM program name to CMS.

NAME=

specifies the name of the APPC/VM program to CMS. When this program issues subsequent HNDIUCV macros or CMSIUCV macros to perform APPC/VM functions, the NAME parameter specified on the CMSIUCV macro must be the same as the one specified here.

addr

specifies the address of an 8 character symbolic name.

(reg)

specifies a register that contains the address of an 8-character symbolic name.

See Usage Note [“1” on page 272](#) for more information.

EXIT=

specifies the address of an exit routine that receives control when an APPC/VM connection pending interrupt occurs for this program. To activate this exit, the connecting program must specify the same name for RESID on the APPCVM CONNECT as the NAME parameter specified for this target program.

The exit address is the default address associated with any path owned by this program. This default address receives control if an APPC/VM external interrupt is presented to the program on a path that does not have a exit address specifically established. Two conditions could cause the default exit address to get control:

- A connect pending interrupt had previously occurred on the path, but the program has not yet issued CMSIUCV ACCEPT.
- A program established a path using CMSIUCV CONNECT or CMSIUCV ACCEPT, but did not specify the EXIT parameter.

The APPC/VM exit routine is called in the addressing mode (24- or 31-bit) of the program issuing this HNDIUCV SET function.

addr

specifies the address of the exit routine.

(reg)

specifies a register that contains the address of the exit routine.

When the program's APPC/VM external interrupt routine is given control, all interrupts are disabled. The exit routine is responsible for providing proper entry and exit linkage for its APPC/VM external interrupt handling routine. The exit routine has the following requirements:

- The routine should not enable itself for any type of interrupts.
- The routine should not perform any I/O operations, because all interrupts are disabled.
- The routine must save and restore the return address in register 14.

When the routine receives control, the significant registers contain:

Register	Contents																												
0	UWORD Field																												
1	<p>If the pending interrupt is for a private resource connection, register 1 contains a X'00'.</p> <p>If a connection to a global or local resource, register 1 points to a SAVEAREA in this format:</p> <table><tr><th>Label</th><th colspan="2">Displacement</th><th>Contents</th></tr><tr><td></td><th>Dec</th><th>Hex</th><td></td></tr><tr><td>GRS</td><td>0</td><td>0</td><td>General purpose registers 0-15 at the time of the interrupt.</td></tr><tr><td>FRS</td><td>64</td><td>40</td><td>Floating point registers 0-7 at the time of the interrupt.</td></tr><tr><td>PSW</td><td>96</td><td>60</td><td>External Old PSW at the time of the interrupt.</td></tr><tr><td>UAREA</td><td>104</td><td>68</td><td>Register save area for exit routine's use.</td></tr><tr><td>END</td><td>176</td><td>B0</td><td>End of save area.</td></tr></table>	Label	Displacement		Contents		Dec	Hex		GRS	0	0	General purpose registers 0-15 at the time of the interrupt.	FRS	64	40	Floating point registers 0-7 at the time of the interrupt.	PSW	96	60	External Old PSW at the time of the interrupt.	UAREA	104	68	Register save area for exit routine's use.	END	176	B0	End of save area.
Label	Displacement		Contents																										
	Dec	Hex																											
GRS	0	0	General purpose registers 0-15 at the time of the interrupt.																										
FRS	64	40	Floating point registers 0-7 at the time of the interrupt.																										
PSW	96	60	External Old PSW at the time of the interrupt.																										
UAREA	104	68	Register save area for exit routine's use.																										
END	176	B0	End of save area.																										
2	Address of the APPC/VM External Interrupt Buffer																												
3	Address of the connection pending extended data (if the exit is driven by a connection pending interrupt), or the address of the connection complete extended data (if the exit is driven by a connection complete interrupt).																												
4	Address of the PIP variable (if the exit is driven by a connection pending interrupt).																												

Register	Contents
13	Points to the register save area at label UAREA for use by the exit routine. (If register 1 contains a X'00', register 13 points to a standard register save area.)
14	Return address
15	Entry point address

Optional Parameters:

label

is an optional assembler label for the statement.

UWORD=

specifies a fullword (user word) containing information that the invoking program can specify. CMS passes this user word to the exit routine when an interrupt is presented for this APPC/VM path. The exit routine can use this information if it desires to do so. When the exit routine receives control, register 0 contains either an address where the user word is stored (if UWORD=*addr*) or the value of a register that contains the user word (if UWORD=(*reg*)).

If the UWORD value is not specified on a CMSIUCV ACCEPT or CMSIUCV CONNECT macro, it defaults to the UWORD value specified here or on an HNDIUCV REP. If you do not specify UWORD here, the user word value is set to zero.

addr

specifies the address where the user word value is stored.

(*reg*)

specifies a register that contains the user word value.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. Because more than one HNDIUCV SET can be issued by an application, the name specified in the NAME= parameter generally has the following meaning for a target virtual machine:
 - For APPC/VM— the name of an APPC/VM resource being managed by the application.
 - For IUCV— the name should be specified in the IPUSER field of the source virtual machine's IUCV CONNECT parameter list.

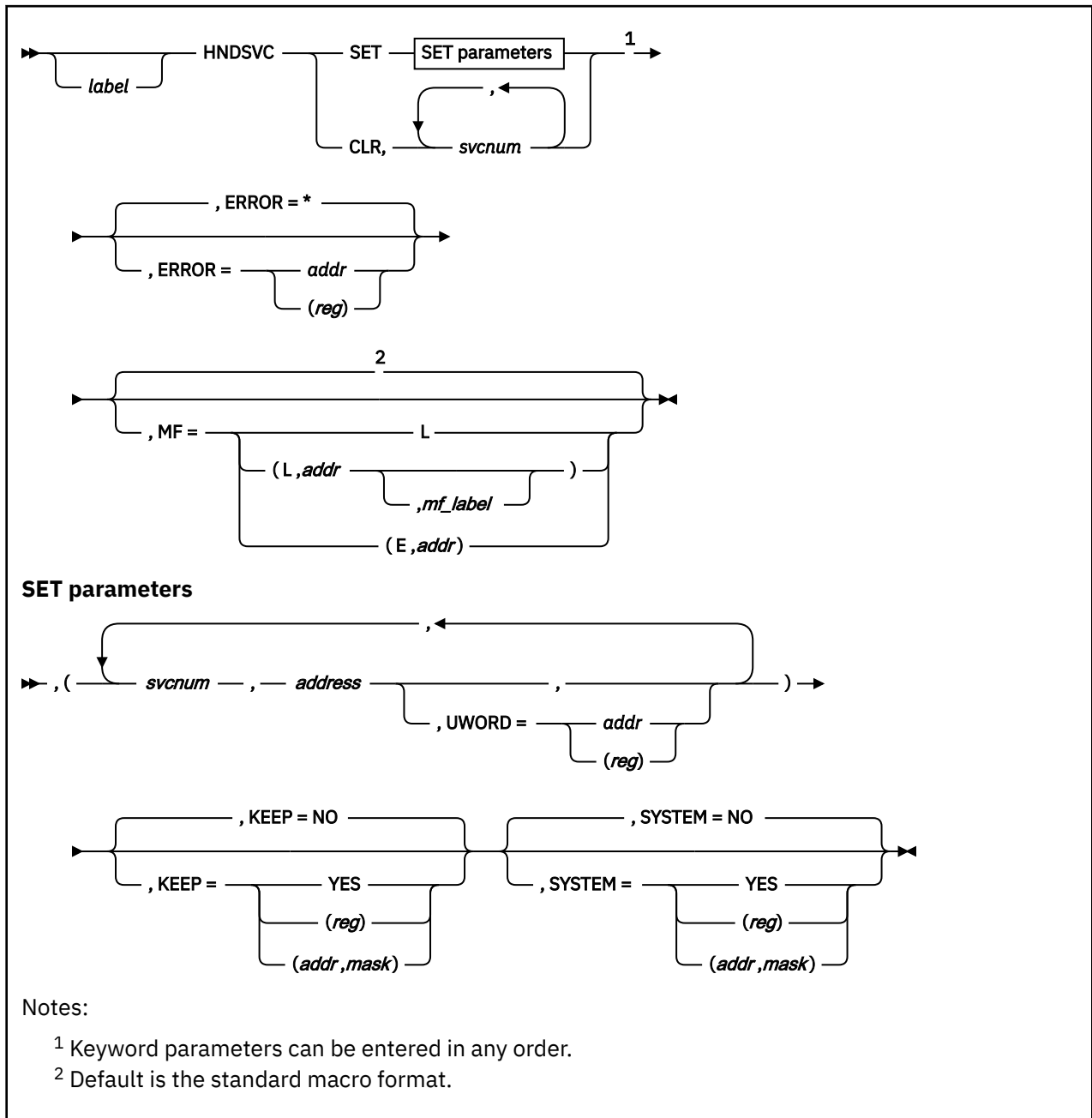
For more information regarding IUCV and APPC/VM communications, refer to [z/VM: CMS Application Development Guide for Assembler](#).

Return Codes

Upon completion of the HNDIUCV SET function, register 15 contains one of the following return codes:

Hex Code	Decimal Code	Meaning
X'00'	0	HNDIUCV SET completed successfully.
X'04'	4	A program with this name has previously issued an HNDIUCV SET.
X'10'	16	The NAME parameter was not specified or its address is equal to 0.
X'14'	20	The EXIT parameter was not specified or its address is equal to 0.
X'20'	32	An IUCV DCLBFR has already been issued by a non-CMS IUCV program. CMS IUCV support cannot be initialized.
X'24'	36	Errors were encountered reading the directory for the virtual machine during CMS IUCV initialization.
X'28'	40	An invalid HNDIUCV function was specified; it must be SET, CLR, REP, HLD, or RES.
X'30'	48	The IUCV DCLBFR CONTROL=YES failed, as indicated by CP.
X'48'	72	A zero value was found for the MAXCONN definition during CMS initialization, or the HNDIUCV SET cannot be performed because the SET for !CMS failed during IPL. (!CMS is a reserved name for CMS.
X'68'	104	Out of storage.
X'C8' + xx	2dd	<p>An error was encountered in getting CMS free storage. The xx is the hexadecimal return code from CMSSTOR; the dd is the decimal equivalent of this return code.</p> <p>If this error occurred because there was insufficient storage during CMS initialization, the storage calculation consisted of the MAXCONN value times 64, the size of a single path table entry (in bytes). Other storage is also included in the storage request, but it is not considered a major factor. The user can increase the virtual machine storage through the CP DEFINE STORAGE command. The user may have to contact the system administrator to increase the maximum storage size for the virtual machine or to reduce the MAXCONN value.</p>
X'7D0' + xxx	2ddd	<p>HNDIUCV was unable to create a second-level handler for IUCV external interrupts. The xxx is the hexadecimal return code from HNDEXT; the ddd is the decimal equivalent of this return code.</p>

HNDSVC



Purpose

Use the HNDSVC macro to set or clear routines that trap interrupts caused by specific supervisor call (SVC) instructions.

Parameters

Required Parameters:

SET

specifies that you want to trap SVCs of the specified number(s).

svcnum

specifies the number of the SVC you want to trap. SVC numbers 0 through 198 and 206 through 255 are valid.

address

specifies the address of the routine in your program that should receive control whenever the specified SVC is issued.

CLR

specifies that you no longer want to trap the specified SVC(s).

svcnum

specifies the number of the SVC you want to clear. SVC numbers 0 through 200 and 206 through 255 are valid.

Optional Parameters:

label

is an optional assembler label for the statement.

UWORD=

is an optional fullword parameter available to SVC trap routines. When the SVC trap routine gets control, UWORD is contained in the HSVUWORD field of the HSVCSAVE control block, which register 13 points to. Acceptable values are:

addr

specifies UWORD as an assembler expression.

(reg)

specifies a register that contains the address of the UWORD. Valid registers are 2-12 enclosed in parentheses.

KEEP=

specifies whether the SVC handler definitions are cleared at end-of-command. Acceptable values are:

NO

specifies that the SVC handler definitions are cleared. This is the default value.

YES

specifies that the SVC handler definitions are not cleared. If you issue KEEP=YES, make sure the SVC trap routines themselves survive end-of-command processing.

(reg)

specifies the register that contains the value for KEEP. The macro checks the value of the specified register and, if it is 0, sets KEEP to NO. If the register contains a nonzero value, the macro sets KEEP to YES.

(addr,mask)

defines a single bit in storage that sets the value of the KEEP parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then KEEP is set to NO. If the bit is 1, then KEEP is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the KEEP parameter as

```
KEEP=(APPFLAG,X'80')
```

Note that when you specify KEEP on an HNDSVC macro call, the KEEP attribute applies to all user SVC handler definitions you specify on that macro call.

To set the value of the KEEP parameter at assembly time, specify KEEP=YES or KEEP=NO. To set the value at execution time, specify KEEP=(reg) or KEEP=(addr,mask).

SYSTEM=

specifies whether the SVC handler definitions survive abend processing. Acceptable values are:

NO

specifies that the SVC handler does not survive. This is the default value.

YES

specifies that the SVC handler does survive. If you issue SYSTEM=YES, make sure the SVC trap routines themselves survive abend processing.

(reg)

specifies the register that contains the value for SYSTEM. The macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the SYSTEM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,mask*).

End-of-command processing follows abend processing; therefore, if you want SVC handlers to survive abend processing and end-of-command processing, specify SYSTEM=YES **and** KEEP=YES.

When you specify SYSTEM on an HNDSVC macro call, the SYSTEM attribute applies to all user SVC handler definitions you specify on that macro call.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. SVC trap routines receive control in the addressing mode of the program that issues the HNDSVC macro, not in the addressing mode of the program that issues the *trapped* SVC. To change or set the addressing mode of an SVC trap routine, use the AMODESW macro.

2. In an XC virtual machine, your second-level interrupt handler always receives control in primary space address translation mode and always must return control to CMS in primary space mode.
3. You must provide the proper entry and exit linkage for your SVC handling routine. When your program receives control, the register contents are as follows:

**Register
Contents**

R0-R11

Remain the same as when the SVC was issued.

R12

If the current addressing mode is AMODE 24, register 12 contains the SVC number in the high-order byte and a 3-byte address of the routine. If the addressing mode is AMODE 31, register 12 contains only the address of the SVC trap routine. For both addressing modes, the address of the SVC trap routine is in register 12 and the UWORD and SVC number can be found in the HSVCSAVE pointed to by register 13.

R13

The address of an HSVCSAVE save area.

R14

The return address to the SVC handler.

R15

Remains the same as when the SVC was issued.

When complete, your routine must return control to the address in register 14. You do not need to restore any registers. The registers are restored to the contents they held at the time the SVC was issued.

4. In multiprocessor mode CMS internally uses SVC 199, so this code should not be handled by multiprocessor applications. If the application uses the direct call CSL interface or takes advantage of multitasking, CMS will also use SVC 200.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
-------------	----------------

- | | |
|----------|--|
| 1 | Invalid SVC number or address. |
| 2 | SVC number set replaced previously set number. |
| 3 | SVC number cleared was not set. |

HSVCSAVE



Purpose

Use the HSVCSAVE macro to generate a DSECT for the HSVCSAVE control block.

Parameters

Optional Parameters:

label

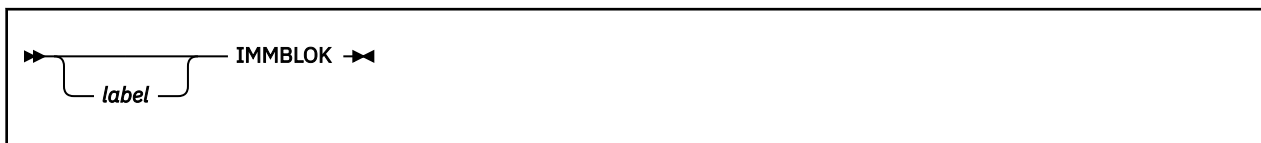
is an optional assembler label for the statement. The first statement in the HSVCSAVE macro expansion is labeled HSVCSAVE.

Usage Notes

- 1. The HSVCSAVE macroinstruction expands as follows:

HSVCSAVE			
HSVCSAVE	DSECT		
HSVUSAVE	DS	12D	USER REGISTER SAVE AREA
HSVUWORD	DS	1F	USER WORD
HSVCNUMB	DS	1X	SVC NUMBER CAUSING CONTROL
	DS	3X	RESERVED FOR IBM USE

IMMBLOK



Purpose

Use the IMMBLOK macro to generate a DSECT for the IMMBLOK control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the IMMBLOK macro expansion is labeled IMMBLOK.

Usage Notes

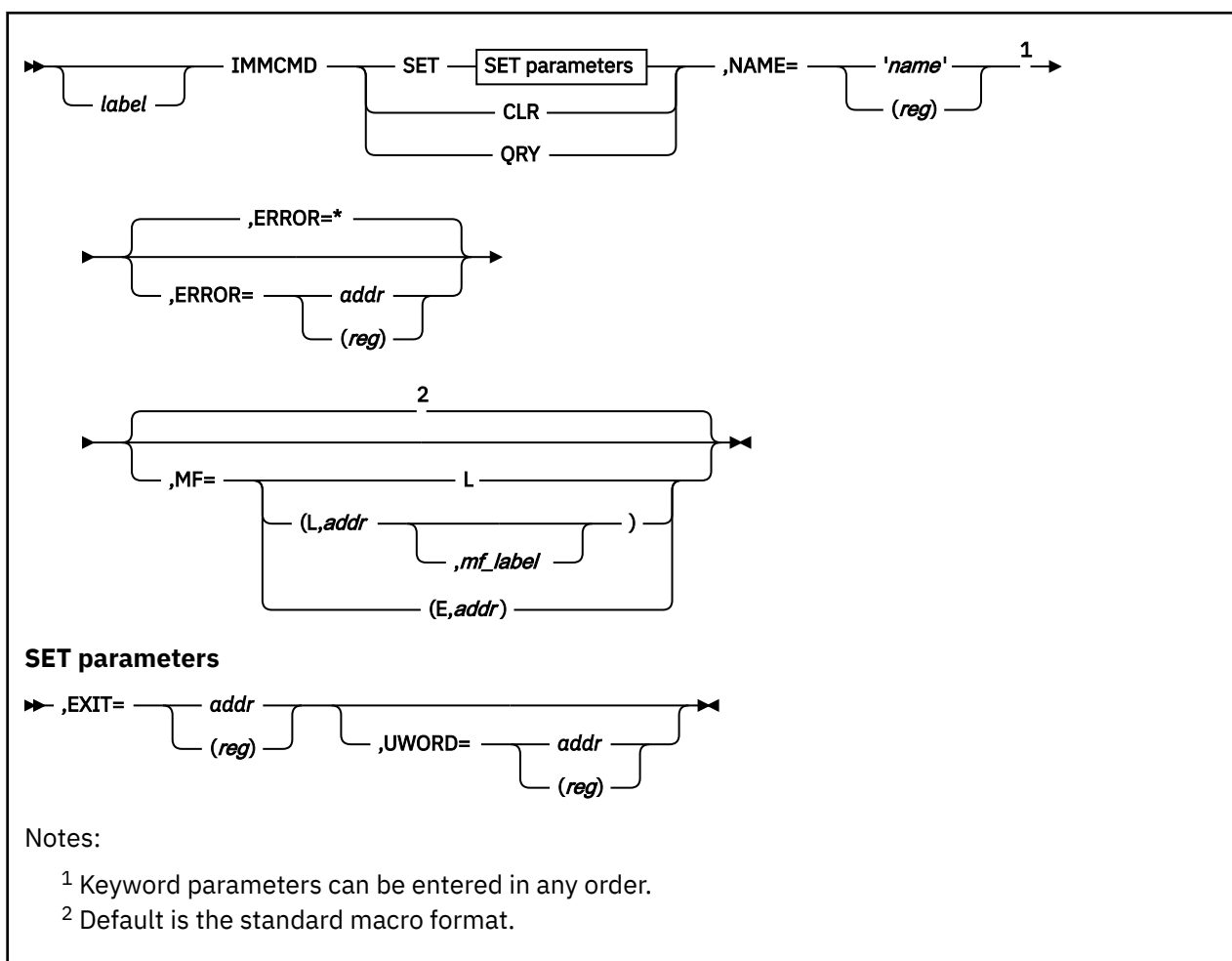
1. For more information on the IMMBLOK macroinstruction, see the macroinstruction [“IMMCMD”](#) on page 279.
2. The IMMBLOK macroinstruction expands as follows:

```

      IMMBLOK
*
*      IMMEDIATE COMMAND NAME BLOCK
*
*
IMMBLOK DSECT
IMMNEXTD EQU  *-IMMBLOK      IMMNEXT DISP INTO IMMBLOK
IMMNEXT  DS   A              POINTER TO NEXT IMMBLOK
IMMNEXTL EQU   4             LENGTH OF IMMNEXT FIELD
IMMUWORD DS   F              USER WORD
IMMNAME  DS   CL8            IMMEDIATE COMMAND NAME
IMMFLAG1 DS   X              FLAGS
IMMSYS   EQU  X'80'          IMMEDIATE COMMAND IS A NUCLEUS
*                               EXTENSION WITH SYSTEM
*                               ATTRIBUTE
IMMCOUNT EQU  X'40'          IMMEDIATE COMMAND ESTABLISHED
*                               VIA IMMCMD COMMAND
IMMNUCX  EQU  X'20'          IMMEDIATE COMMAND IS A NUCLEUS
*                               EXTENSION
IMMAM31  EQU  X'10'          AMODE OF EXIT IS AMODE 31
IMMFLAG2 DS   3X            FLAGS
IMMADDR  DS   A              ADDRESS OF EXIT ROUTINE
IMMHIDE  DS   F              NUMBER OF NUCLEUS EXTENSIONS
*                               THAT ARE HIDING THIS
*                               IMMEDIATE COMMAND
IMMEND   DS   0D
IMMDWDS  EQU  (*-IMMBLOK)/8  SIZE IN DOUBLEWORDS
IMMBYTES EQU  (*-IMMBLOK)    SIZE IN BYTES

```

IMMCMD



Purpose

Use the IMMCMD macroinstruction to declare, clear, and obtain information about immediate commands.

Parameters

Required Parameters:

SET

establishes an immediate command. If an immediate command with the same name already exists, it is overridden in a stack-like manner.

CLR

clears an immediate command. Any previously overridden immediate command with the same name is reinstated.

QRY

indicates that the caller is requesting information about an immediate command. A return code from QRY indicates whether the immediate command exists.

NAME=

is the name of the immediate command. This parameter is always required. Acceptable values are:

'name'

specifies a 1- to 8-character name enclosed within single quotation marks.

(reg)

specifies a register enclosed in parentheses that contains the address of the immediate command name.

EXIT=

is the address of the routine that receives control when the immediate command is entered from the terminal. Acceptable values are:

addr

specifies an assembler program label that is the address of the exit routine.

(reg)

specifies a general register. Its value is the address of the exit routine.

Optional Parameters:

label

is an optional assembler label for the statement.

UWORD=

is an optional fullword that can be specified for any purpose desired. When the exit routine gains control, UWORD is available to it in the IMMBLOK. Register 2 points to the IMMBLOK (see the Usage Notes for more details). Acceptable values are:

addr

specifies an assembler program label as the address that is stored as the UWORD.

(reg)

specifies a register that contains the UWORD.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. Immediate command routines receive control in the addressing mode of the program that issues the IMMCMD macro, not in the addressing mode of the program that issues the immediate command.

- The IMMCMD EXIT parameter lets you give control to an exit routine whenever a specific immediate command is invoked. These exit routines receive control as an extension of CMS I/O interrupt handling—with a PSW key of 0 and disabled for interrupts. The exit routine must not perform any I/O operations, issue any SVCs that result in I/O operations, or enable itself for interrupts. DIAGNOSE instructions can be used within the exit, but the exit routine must not enable itself for interrupts that may be caused by the DIAGNOSE (for example, DIAGNOSE code X'58'). On entry, the exit routine is passed the following information:

Register

Contents

R0

Address of immediate command line in extended parameter list format.

R1

Address of immediate command line in standard parameter list format. For a 31-bit mode program, register 1 contains only the address. For a program running in 24-bit mode, the high-order byte of register 1 is set to X'06' to indicate that this routine was invoked as a result of an immediate command.

R2

Address of the IMMBLOK. The IMMBLOK contains the user word and other relevant information. The format of the IMMBLOK is as follows:

Bytes

Information

0-3

Address of next IMMBLOK

4-7

User word

8-15

Command name

16-19

Reserved

20-23

Entry point address

R3-R11

Unspecified

R12

Entry address

R13

A thirteen doubleword save area mapped by the USERSAVE macro. The USECTYP field of USERSAVE is set to X'06' to indicate that the routine was invoked as an immediate command.

R14

Return address

R15

Entry address

- Immediate commands must use BR 14 rather than CMSRET to return control. Using CMSRET may cause the program that invoked the immediate command to end, rather than cause just the immediate command itself to end.
- Immediate commands created by the IMMCMD macro are automatically deleted when a program returns to the CMS command environment (except when in CMS subset mode), or when CMS performs abend recovery. To explicitly delete an immediate command that was created by IMMCMD SET, use IMMCMD CLR. Any previously overridden immediate command with the same name is reinstated by this action.

For example, to delete the immediate command named DOIT, code:

```
IMMCMD CLR,NAME='DOIT'
```

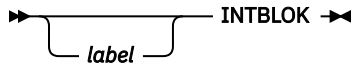
Note: To delete an immediate command that was created by the NUCXLOAD command, the NUCEXT function, or the NUCEXT macro, use the NUCXDROP command, the NUCEXT CANCEL function, or the NUCEXT CLR macro.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
------	---------

INTBLOK



Purpose

Use the INTBLOK macro to generate a DSECT for the INTBLOK control block. The INTBLOK control block contains device information returned by the HNDIO macro.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the INTBLOK macro expansion is labeled INTBLOK.

Usage Notes

1. The INTBLOK macroinstruction expands as follows:

Note: 370 fields are no longer used.

INTBLOK	DSECT		
INTXACOD	DS	0XL8	Interrupt information
INTPARM	DS	F	For XA/XC, interruption parameter; for 370, 0
INTIDENT	DS	0XL4	For XA/XC, subsystem ID word (SID); for 370, see individual components
	DS	H	For XA/XC, X'0001'; for 370, 0
INTSUBCH	DS	H	For XA/XC, subchannel number; for 370, use INTDEVAD
INTDEVAD	ORG DS	INTSUBCH H	For XA/XC, use INTSUBCH; for 370, device address
INTPSW	DS	D	I/O old PSW
INTXAIRB	DS	0XL64	For XA/XC, actual interruption response block (IRB); for 370, constructed IRB
INTSCSW	DS	0XL12	For XA/XC, actual subchannel status word (SCSW); for 370, constructed SCSW
INTSCCTL	DS	X	Key, S, L, and CC
	DS	X	For XA/XC, miscellaneous SCSW bits; for 370, 0
INTCCWFM	EQU	X'80'	For XA/XC, CCW format; for 370, 0
	DS	XL2	For XA/XC, more miscellaneous SCSW bits; for 370, 0
INTCCWAD	DS	A	For XA/XC, CCW address; for 370, X'00' and 3-byte CCW address
INTDEVST	DS	X	Device status
INTSCHST	DS	X	For XA/XC, subchannel status; for 370, channel status
INTRCNT	DS	H	Residual byte count
	DS	13F	For XA/XC, extended status word (ESW) and extended control word (ECW); for 370, 0
INTSTAT	DS	X	Status of INTBLOK information
INTFAIL	EQU	X'80'	For XA/XC, if 1, TSCH failed and IRB is not valid; for 370, always 0
INTPS370	EQU	X'40'	For XA/XC, always 0 (actual information given); for 370, always 1 (constructed, or "pseudo", information given)
	DS	1X	Reserved

INTBLOK

INTDEVNO	DS	1H	Saved device addr for user exit
INTBLKSZ	EQU	*-INTBLOK	INTBLOK length

LABSECT

►► LABSECT ◄◄

Purpose

Use the LABSECT macroinstruction to generate a DSECT for holding user tape label information.

Usage Notes

1. LABSECT is supported for use with the tape label processing routines. For more information, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The LABSECT control block is built from information supplied when:
 - A LABELDEF command is issued.
 - Either a user or the system issues a FILEDEF command for a tape.

For more information on FILEDEF and LABELDEF commands, see [z/VM: CMS Commands and Utilities Reference](#) and [z/VM: CMS Application Development Guide for Assembler](#).
3. A total of 16 volume IDs can be specified in this control block. If additional volumes are needed, a VOLSECT control block is used for each additional 24 volume IDs. See [“VOLSECT” on page 429](#) for more information.
4. Scratch tape processing is either specified by the user or as a result of system action as follows:
 - If a specific volume is not identified, then any scratch tape from the tape library free pool is selected.
 - By specifying *SCRATCH* in the LABELDEF command.
 - When no tape volume ID is specified in the LABELDEF command.
 - When the list of specified volume IDs has been exhausted and yet another volume is needed.
 - When the list of the specified volume IDs has *SCRATCH* entered.
5. The following control bits in the LABFLAG2 field have the following meaning:
 - LABSCRAT—bit on means that scratch processing is in effect.
 - LABSCRSP—user specifically requested scratch as the volume ID.
 - LABLBDSC—scratch processing set as a default because no volume IDs were entered.
 - LABFDEF—this LABSECT was created as a default block by the FILEDEF command.
6. The LABSECT macro expands as follows:

```

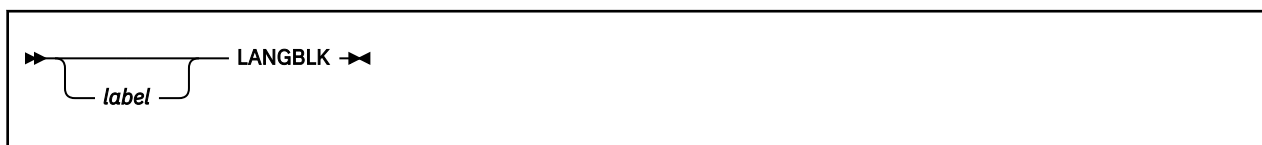
MACRO
LABSECT
*
* LABSECT - FOR HOLDING USER SUPPLIED TAPE LABEL INFORMATION
*
LABSECT DSECT
LABNEXT DS A RESERVED
LABFCBPT DS A POINTER TO FCBSECT OR ZERO
LABFILE DS CL8 NAME OF FILE (DDNAME) FOR BLOCK
LABFID DS CL17 FILE ID (RIGHTMOST 17 CHARACTERS)
LABSEC DS CL1 SECURITY
LABVOLID DS CL6 VOLUME SERIAL NUMBER (VALID)
LABVSEQ DS CL4 VOLUME SEQUENCE NUMBER
LABFSEQ DS CL4 FILE SEQUENCE NUMBER
LABGENN DS CL4 GENERATION NUMBER
LABGENV DS CL2 GENERATION VERSION
LABCRD DS CL6 CREATION DATE
LABEXD DS CL6 EXPIRATION DATE
SPACE
LABFLAG1 DS 1X THIS BYTE HAS DEFAULT FLAGS:
LABDFID EQU X'80' DEFAULT FILE ID

```

LABSECT

LABDSEC	EQU	X'40'	DEFAULT SECURITY
LABDVID	EQU	X'20'	DEFAULT VOLUME SERIAL NUMBER
LABDVSEQ	EQU	X'10'	DEFAULT VOLUME SEQUENCE NUMBER
LABDFSEQ	EQU	X'08'	DEFAULT FILE SEQUENCE NUMBER
LABDGENN	EQU	X'04'	DEFAULT GENERATION NUMBER
LABDGENV	EQU	X'02'	DEFAULT GENERATION VERSION
LABDCRD	EQU	X'01'	DEFAULT CREATION DATE
SPACE			
LABFLAG2	DS	1X	MISCELLANEOUS FLAGS BYTE:
LABDEXD	EQU	X'80'	DEFAULT EXPIRATION DATE
LABSCRAT	EQU	X'40'	DO 'SCRATCH' VOLID PROCESSING
LABSCRSP	EQU	X'20'	SCRATCH SPECIFIED; NOT DEFAULT
LABLBDSC	EQU	X'10'	LABSCRAT set by LABELDEF
LABFDEF	EQU	X'04'	LABSECT GOTTEN BY FILEDEF
LABPERM	EQU	X'02'	PERMANENT SPECIFIED
LABNOCHG	EQU	X'01'	NOCHANGE SPECIFIED
SPACE			
LABCUVOL	DS	A	POINTER TO CURRENT VOLID MOUNTED
LABNXVOL	DS	A	POINTER TO NEXT VOLID TO MOUNT
LABVSECT	DS	A	FORWARD CHAIN POINTER TO VOLSECT
LABCSECT	DS	A	VOLSECT ADDR OF CURRENT VOLID
LABVOLS	DS	CL120	SPACE FOR 15 ADDITIONAL VOLIDS
LABEND	DS	XL4'FF'	FENCE FOR END OF VOLIDS
LABFILID	DS	CL44	FILE IDENTIFIER
LABSIZE	EQU	(*-LABSECT+7)/8	SIZE OF LABSECT IN DOUBLE WORDS

LANGBLK



Purpose

Use the LANGBLK macro to generate a DSECT for the LANGBLK control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the LANGBLK macro expansion is labeled LANGBLK.

Usage Notes

1. The LANGBLK macroinstruction expands as follows:

```

LANGBLK      LANGBLK
LANGBLK DSECT
LANGNEXT DC AL4(0)      Pointer to next LANGBLK
LANGAPID DC CL3'DMS'    Application ID
LANGFLG1 DC X'78'       Flag byte
LANGET EQU X'80'        On indicates DBCS language
LANGUSSY EQU X'40'      User synonyms wanted
LANGUSTR EQU X'20'      User translations wanted
LANGSYSY EQU X'10'      System synonyms wanted
LANGSYTR EQU X'08'      System translations wanted
LANGLANG DC CL5' '      Language identifier
              DC X'00'    Reserved for IBM use
LANGDISK DC XL2'00'     HELP (or application) disk address
LANGMSG DC AL4(0)       Message repository
LANGSPA DC AL4(0)       System parser table
LANGUPA DC AL4(0)       User parser table
LANGSSY DC AL4(0)       System Synonym and Abbreviation table
LANGUSY DC AL4(0)       User Synonym and Abbreviation table
LANGTRTS DC AL4(0)      NLS translation tables
LANGUSER DC AL4(0)      Reserved for application's use
LANGUME DC AL4(0)       User additions to message repository
LANGBLKB EQU *-LANGBLK  Bytes of storage for LANGBLK
LANGBLKD EQU (LANGBLKB+7)/8 Double words of storage for LANGBLK

```

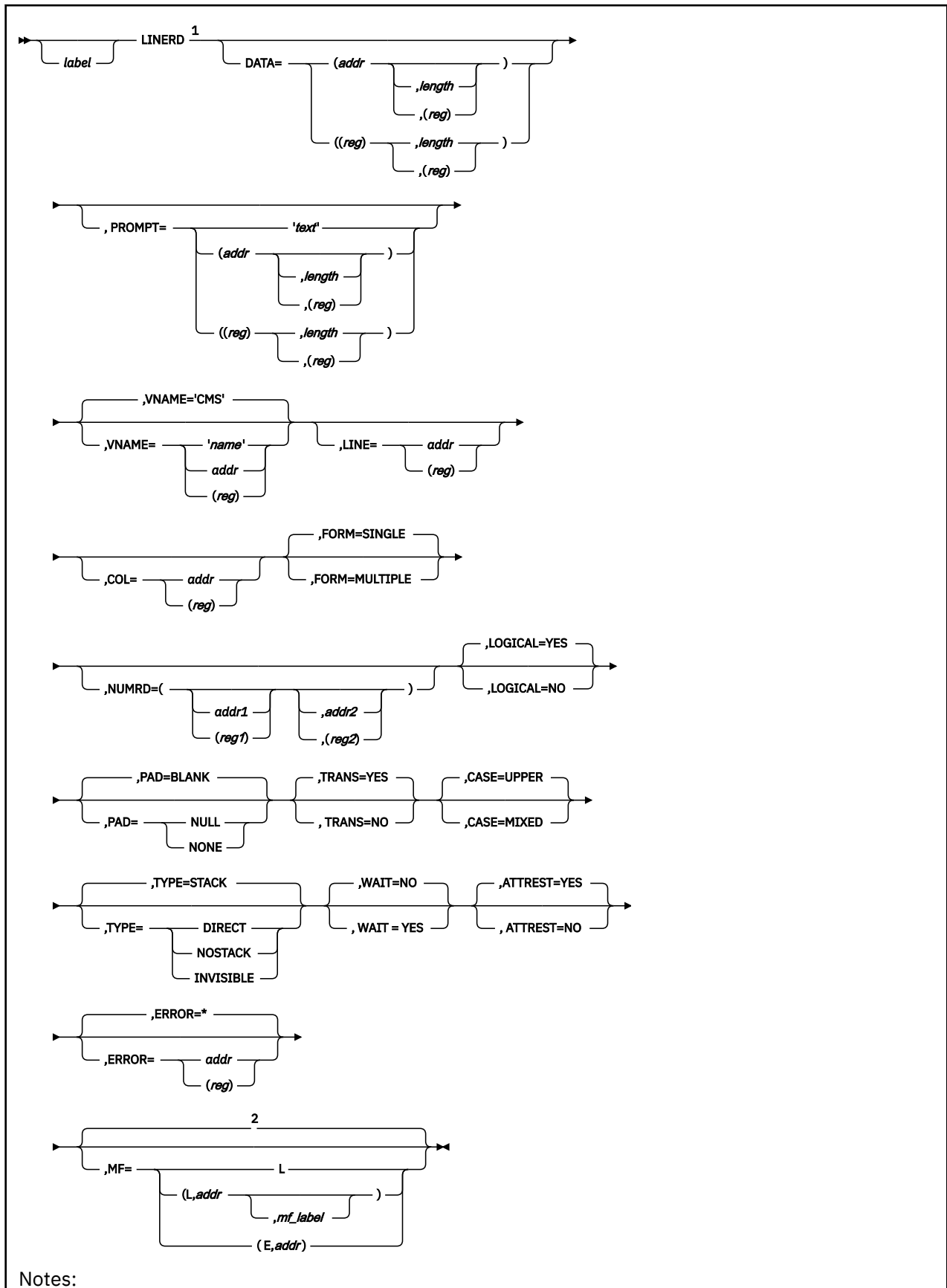
2. The LANGBLK fields are used as follow:

- LANGBLK fields are modified indirectly, with the exception of LANGUSER, using one or a combination of:
 - SET LANGUAGE command, for more information see, [z/VM: CMS Commands and Utilities Reference](#).
 - LANGADD function, refer to “LANGADD” on page 447 for more information.
 - LANGFIND function, refer to “LANGFIND” on page 449 for more information.
- Most fields are used indirectly by using system facilities such as APPLMSG or PARSECMD. LANGFLG1, LANGDISK, and LANGTRTS are used by CMS only if the LANGAPID='DMS'.
- LANGAPID and LANGLANG are set by an application in LANGBLKs used as input to the LANGFIND and LANGADD functions.
- LANGMSG, LANGSPA, LANGSSY, LANGDISK, and LANGTRTS are set by an application using the LANGADD function when the NLS information does not reside in a system language segment.

LANGBLK

- LANGDISK and LANGTRTS may be referenced by an application after using the LANGFIND function.
- LANGUSER is an application only usage field. It may be directly modified using the LANGFIND function to locate the LANGBLK.

LINERD



¹ Keyword parameters can be entered in any order.

² Default is the standard macro format.

Purpose

Use the LINERD macroinstruction to read one or more lines of input from the terminal. The LINERD macro can be used for single or multiple reads in full screen mode (SET FULLSCREEN ON) and can be used for single reads in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND).

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

DATA=

specifies the address and length of the text to be read when the FORM parameter is omitted or specified as SINGLE. When FORM=MULTIPLE is specified, the address designates the beginning of the chain of input descriptors and the *length* field specifies in bytes the length of the buffer for the chain of descriptors. Acceptable values are:

(*addr,length*)

specifies the address as an assembler expression and, optionally, the length as an absolute expression. If a label specifies the address and the length is not specified, the length associated with the label will be used.

(*addr, (reg)*)

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses. If a label specifies the address and the length is not specified, the length associated with the label will be used.

((*reg*) , *length*)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((*reg*) , (*reg*))

specifies a register that contains the address and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Note: The DATA parameter is required with the standard format of the LINERD macro.

PROMPT=

specifies the address and length of the prompt information written when a line is read. If you omit PROMPT, no prompt information is displayed. Acceptable values are:

'*text*'

specifies the prompt text as a literal string enclosed in quotation marks.

(*addr,length*)

specifies the address of the text as an assembler expression and, optionally, specifies the length as an absolute expression. If a label specifies the address and the length is not specified, the length associated with the label will be used.

(*addr, (reg)*)

specifies the address of the text as an assembler expression and, optionally, specifies the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses. If a label specifies the address and the length is not specified, the length associated with the label will be used.

((reg), length)

specifies a register that contains the address of the text and specifies the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the text and specifies a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

If you specify length but not an address, CMS assumes the prompt information is in the read buffer.

VNAME=

specifies the name of the virtual screen to be read. If you omit VNAME, the default vscreen name is CMS. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in quotation marks.

addr

specifies the name as an assembler label.

(reg)

specifies a register that contains the address of an 8-byte name.

LINE=

specifies the virtual screen line from which the data was read. This information is not available if VNAME='CMS' (or default) and CMS is in line mode. Acceptable values are:

addr

specifies the address of a fullword in storage where LINERD stores the virtual screen line of the data read.

(reg)

specifies a general register (2-12) enclosed in parentheses that contains the address of a fullword in storage containing the virtual screen line of the data read.

COL=

specifies the virtual screen column from which the data was read. This information is not available if VNAME='CMS' (or default) and CMS is in line mode. Acceptable values are:

addr

specifies the address of a fullword in storage where LINERD stores the virtual screen column of the data read.

(reg)

specifies a general register (2-12) enclosed in parentheses that contains the address of a fullword in storage containing the virtual screen column of the data read.

FORM=

specifies whether more than one input is requested in the application buffer. Acceptable values are:

SINGLE

means that only one input is requested. This is the default.

MULTIPLE

means that a chain of input descriptors is requested in the application buffer. The first input descriptor in the chain only returns information on the cursor position and the key pressed. The other input descriptors return the information for each modified field.

NUMRD=

returns the number of inputs read (number of modified fields plus one for the descriptor returning the cursor and key information) and the length of the next input if there are more inputs to be read (see Usage Note "15" on page 299). This parameter should be used when FORM=MULTIPLE is specified. Acceptable values are:

addr1

specifies a fullword of storage to return the number of inputs read.

(reg1)

specifies a register (2-12) which contains the address of a fullword of storage to return the number of inputs read.

addr2

specifies a fullword of storage in which to return the length of the next input if there are more inputs to be read.

(reg2)

specifies a register (2-12) which contains the address of a fullword of storage to return the length of the next input if there are more inputs to be read.

LOGICAL=

specifies whether new-line characters in the input data are interpreted as a logical end-of-line. Processing of the line-end symbol only occurs for data entered in the CMS virtual screen. Acceptable values are:

YES

only the logical line is returned. This is the default value.

NO

the new-line characters are ignored and the entire line is returned.

PAD=

specifies whether the input data is padded with nulls or blanks to the length of the input buffer. Acceptable values are:

BLANK

pads with blanks. This is the default value.

NULL

pads with nulls.

NONE

specifies that no padding is requested. If the data you read does not fill the input buffer, the remainder of the input buffer contains its previous contents.

PAD=NONE is invalid when CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND).

TRANS=

specifies whether the input data is translated according to the user input translate table, if any, defined by the SET INPUT command. Acceptable values are:

YES

translates input data according to the user input table. This is the default value.

NO

specifies the macro does not translate input data.

CASE=

specifies whether the input data is translated to upper case. Acceptable values are:

UPPER

translates input data to upper case. This is the default value.

MIXED

specifies that the macro does not translate data to upper case; the data is left as is.

TYPE=

specifies the type of read request. Acceptable values are:

STACK

reads a line (a) from the program stack if a line is available, (b) from the input queue of the specified virtual screen if the queue is not empty, or (c) directly from the console. LINERD does not perform user input translation or logical line editing for lines read from the program stack. This is the default value.

DIRECT

reads the input line directly from the virtual machine console, bypassing the program stack and the input queue associated with the virtual screen. If TYPE=DIRECT, LINERD redisplay the input data on the virtual machine console.

NOSTACK

bypasses the program stack and reads a line from the virtual screen input queue or directly from the virtual machine console.

INVISIBLE

like TYPE=DIRECT, TYPE=INVISIBLE reads the input line directly from the virtual machine console, bypassing the input queue associated with the virtual screen and the program stack. Unlike TYPE=DIRECT, TYPE=INVISIBLE does not redisplay the input data on the virtual machine console.

WAIT=

specifies the status area message. This can help you to distinguish between system read requests and program read requests during program execution. Acceptable values are:

NO

specifies that there is no distinction between system reads and program reads. If FULLSCREEN is ON, the status area message is: Enter a command or press a PF or PA key. If FULLSCREEN is OFF or SUSPEND and VNAME is CMS (or default), the status area message is RUNNING. The default is NO.

YES

specifies the status area message (when FULLSCREEN is ON) as: Enter your response in vscreen VNAME. This shows that your program is requesting input (a program read). If FULLSCREEN is OFF or SUSPEND and VNAME is CMS (or default), the status area message is VM READ.

If you specify TYPE=DIRECT or INVISIBLE, or if you specify the PROMPT parameter and the read is satisfied from the virtual console, the WAIT parameter, if specified, is ignored and the status area message is Enter your response in vscreen VNAME or VM READ.

Note: If FULLSCREEN is OFF or SUSPEND and VNAME is not CMS (or default), then no status information is available.

ATTREST=

specifies whether an attention interrupt during a read request restarts the read operation. If CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND) then ATTREST=NO can be used only when reading physical lines (LOGICAL=NO).

The ATTREST operand is ignored when reading from a virtual screen. Acceptable values are:

YES

specifies that an attention interrupt during a read operation restarts the read operation. This is the default value.

NO

specifies that an attention interrupt during a read operation signals the end of the line. ATTREST=NO is valid only when CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND) and you specify LOGICAL=NO.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

label

is an optional assembler label for the statement.

DATA=

specifies the address and length of the text to be read when the FORM parameter is omitted or specified as SINGLE. When FORM=MULTIPLE is specified, the address designates the beginning of the chain of input descriptors and the *length* field specifies in bytes the length of the buffer for the chain of descriptors. Acceptable values are:

(addr,length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression. If a label specifies the address and the length is not specified, the length associated with the label will be used.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses. If a label specifies the address and the length is not specified, the length associated with the label will be used.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Note: The DATA parameter is required with the standard format of the LINERD macro.

PROMPT=

specifies the address and length of the prompt information written when a line is read. If you omit PROMPT, no prompt information is displayed. Acceptable values are:

'text'

specifies the prompt text as a literal string enclosed in quotation marks.

(addr,length)

specifies the address of the text as an assembler expression and, optionally, specifies the length as an absolute expression. If a label specifies the address and the length is not specified, the length associated with the label will be used.

(addr, (reg))

specifies the address of the text as an assembler expression and, optionally, specifies the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses. If a label specifies the address and the length is not specified, the length associated with the label will be used.

((reg), length)

specifies a register that contains the address of the text and specifies the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address of the text and specifies a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

If you specify length but not an address, CMS assumes the prompt information is in the read buffer.

VNAME=

specifies the name of the virtual screen to be read. If you omit VNAME, the default vscreen name is CMS. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in quotation marks.

addr

specifies the name as an assembler label.

(reg)

specifies a register that contains the address of an 8-byte name.

LINE=

specifies the virtual screen line from which the data was read. This information is not available if VNAME='CMS' (or default) and CMS is in line mode. Acceptable values are:

addr

specifies the address of a fullword in storage where LINERD stores the virtual screen line of the data read.

(reg)

specifies a general register (2-12) enclosed in parentheses that contains the address of a fullword in storage containing the virtual screen line of the data read.

COL=

specifies the virtual screen column from which the data was read. This information is not available if VNAME='CMS' (or default) and CMS is in line mode. Acceptable values are:

addr

specifies the address of a fullword in storage where LINERD stores the virtual screen column of the data read.

(reg)

specifies a general register (2-12) enclosed in parentheses that contains the address of a fullword in storage containing the virtual screen column of the data read.

FORM=

specifies whether more than one input is requested in the application buffer. Acceptable values are:

SINGLE

means that only one input is requested. This is the default.

MULTIPLE

means that a chain of input descriptors is requested in the application buffer. The first input descriptor in the chain only returns information on the cursor position and the key pressed. The other input descriptors return the information for each modified field.

NUMRD=

returns the number of inputs read (number of modified fields plus one for the descriptor returning the cursor and key information) and the length of the next input if there are more inputs to be read (see Usage Note "15" on page 299). This parameter should be used when FORM=MULTIPLE is specified. Acceptable values are:

addr1

specifies a fullword of storage to return the number of inputs read.

(reg1)

specifies a register (2-12) which contains the address of a fullword of storage to return the number of inputs read.

addr2

specifies a fullword of storage in which to return the length of the next input if there are more inputs to be read.

(reg2)

specifies a register (2-12) which contains the address of a fullword of storage to return the length of the next input if there are more inputs to be read.

LOGICAL=

specifies whether new-line characters in the input data are interpreted as a logical end-of-line. Processing of the line-end symbol only occurs for data entered in the CMS virtual screen. Acceptable values are:

YES

only the logical line is returned. This is the default value.

NO

the new-line characters are ignored and the entire line is returned.

PAD=

specifies whether the input data is padded with nulls or blanks to the length of the input buffer. Acceptable values are:

BLANK

pads with blanks. This is the default value.

NULL

pads with nulls.

NONE

specifies that no padding is requested. If the data you read does not fill the input buffer, the remainder of the input buffer contains its previous contents.

PAD=NONE is invalid when CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND).

TRANS=

specifies whether the input data is translated according to the user input translate table, if any, defined by the SET INPUT command. Acceptable values are:

YES

translates input data according to the user input table. This is the default value.

NO

specifies the macro does not translate input data.

CASE=

specifies whether the input data is translated to upper case. Acceptable values are:

UPPER

translates input data to upper case. This is the default value.

MIXED

specifies that the macro does not translate data to upper case; the data is left as is.

TYPE=

specifies the type of read request. Acceptable values are:

STACK

reads a line (a) from the program stack if a line is available, (b) from the input queue of the specified virtual screen if the queue is not empty, or (c) directly from the console. LINERD does not perform user input translation or logical line editing for lines read from the program stack. This is the default value.

DIRECT

reads the input line directly from the virtual machine console, bypassing the program stack and the input queue associated with the virtual screen. If TYPE=DIRECT, LINERD redisplay the input data on the virtual machine console.

NOSTACK

bypasses the program stack and reads a line from the virtual screen input queue or directly from the virtual machine console.

INVISIBLE

like TYPE=DIRECT, TYPE=INVISIBLE reads the input line directly from the virtual machine console, bypassing the input queue associated with the virtual screen and the program stack.

Unlike TYPE=DIRECT, TYPE=INVISIBLE does not redisplay the input data on the virtual machine console.

WAIT=

specifies the status area message. This can help you to distinguish between system read requests and program read requests during program execution. Acceptable values are:

NO

specifies that there is no distinction between system reads and program reads. If FULLSCREEN is ON, the status area message is: Enter a command or press a PF or PA key. If FULLSCREEN is OFF or SUSPEND and VNAME is CMS (or default), the status area message is RUNNING. The default is NO.

YES

specifies the status area message (when FULLSCREEN is ON) as: Enter your response in vscreen VNAME. This shows that your program is requesting input (a program read). If FULLSCREEN is OFF or SUSPEND and VNAME is CMS (or default), the status area message is VM READ.

If you specify TYPE=DIRECT or INVISIBLE, or if you specify the PROMPT parameter and the read is satisfied from the virtual console, the WAIT parameter, if specified, is ignored and the status area message is Enter your response in vscreen VNAME or VM READ.

Note: If FULLSCREEN is OFF or SUSPEND and VNAME is not CMS (or default), then no status information is available.

ATTREST=

specifies whether an attention interrupt during a read request restarts the read operation. If CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND) then ATTREST=NO can be used only when reading physical lines (LOGICAL=NO).

The ATTREST operand is ignored when reading from a virtual screen. Acceptable values are:

YES

specifies that an attention interrupt during a read operation restarts the read operation. This is the default value.

NO

specifies that an attention interrupt during a read operation signals the end of the line. ATTREST=NO is valid only when CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND) and you specify LOGICAL=NO.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L, *addr*, *mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E, *addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If the length of the data buffer is specified as 0, then the length will be assumed to be 130 bytes.
2. When LINERD for a single input completes, register 0 contains the number of characters read. Register 0 is unchanged when multiple input format is used.
3. When the virtual screen name is CMS (which is also the default virtual screen name), the action taken by LINERD depends on the setting of full-screen CMS. If SET FULLSCREEN is ON, LINERD waits for input into the CMS virtual screen. When the LINERD function is executed with CMS in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND), LINERD calls the RDTERM function to do the read. To support the length parameter, RDTERM must be called with the EDIT=PHYS option. In this case, data padding is restricted to BLANK or NULL because the RDTERM function clears the entire data buffer before doing the read.
4. If LOGICAL=YES, the maximum length for a read request is 240 bytes. If LOGICAL=NO, the maximum length is 2030 bytes.
5. If you specify LOGICAL=NO, do not store prompt data in the read buffer because the read buffer may be cleared prior to the execution of the function.
6. When a part of a field from a virtual screen is modified, the entire field is returned as a modified field. The LINE and COL parameters are the virtual screen line and column of the field. For more information on display of windows and modified fields in a fullscreen environment, see the VSCREEN WAITREAD command documented in the [z/VM: CMS Commands and Utilities Reference](#).
7. In fullscreen CMS or when VNAME is specified, the PROMPT parameter defines two fields in the virtual screen: one field for the prompt (which continues on the first line from the end of the prompt text to the end of the line) and one field for the user response (which starts on the second line). If the response to be entered cannot fit in the first field (up to the end of the first line), then place the cursor on the second line and enter the response on the second line (which can accept a response longer than one line). If TYPE is not specified as INVISIBLE, the cursor is positioned at the beginning of the response field. Specifying TYPE=INVISIBLE causes the cursor to be positioned on the line following the prompt.
8. The lines in a window that are not reserved lines or data lines are called pad lines. When TYPE=INVISIBLE is specified, the pad lines will be invisible. In addition, if VNAME='CMS' (the default) is specified, the command line will also be invisible. All other lines in the window will be protected.
9. The LRDD mapping macro can be used to map the fields of the LINERD input descriptors.
10. Because of the similar structure of the LINERD and LINEWRT descriptors, the virtual screen can be updated with the inputs by using the input descriptors (LRDD) as output descriptors (LWRD) with the LINEWRT macro.
11. More details of the physical screen display of windows and wait functions are documented in the [z/VM: CMS Commands and Utilities Reference](#) under the WAITREAD command.
12. Modified fields are returned in either of two different formats:
 - Multiple input format, in which a chain of input descriptors (LRDDs) is used
 - Single input format, in which the input is placed at the location specified by the DATA operand.

If the value at *addr1* of NUMRD is greater than 0, the multiple input format is used. The value returned at *addr1* is the number of modified fields plus one (for the cursor and key descriptor). If no fields were modified, the value at *addr1* is 1, and the multiple input format returns the cursor and key descriptor in the buffer. A value of 0 at *addr1* indicates that the input is returned in single input format.

13. Information regarding the key pressed and cursor position is returned to the application only when multiple inputs are returned.
14. FORM=MULTIPLE is ignored and only one input is returned in the single input format when:
 - a. TYPE=STACK is specified (or defaulted to), and the read is satisfied by the program stack.
 - b. The vscreen name (VNAME parameter) is not specified or the vname is CMS and CMS is running in line mode.
15. If there are more inputs to be read (RC=13), the value stored at the fullword location specified by *addr2* will include the:
 - Length of a cursor and key descriptor (LRDD)
 - Length of the next input descriptor (LRDD)
 - Text length of the modified field.

If the application buffer was not large enough to hold the cursor and key descriptor, the value stored at the fullword location specified by *addr2* will represent the length of the cursor and key descriptor only.
16. If the buffer specified by the DATA parameter is not large enough to hold all modified fields, those remaining are left on the queue, and are available for successive LINERDs from this vscreen.
17. When a LINERD is done in a virtual machine that is IPLed as a batch machine, a buffer length of 130 must be specified.
18. Any translation done on the input buffer that contains both SBCS and DBCS data will only occur on the SBCS portions of the data provided that the display is capable of supporting mixed DBCS.
19. If truncation occurs because the data being read in is longer than the input buffer, and the truncation occurs within a mixed DBCS string, then adjustments will be made to validate the truncated string.
20. CMS signals the VMCONINPUT event whenever it receives unsolicited input from the virtual machine console. It is a broadcast event with session scope and does not synchronize the handling process. It contains no event data. The monitoring application should perform a read operation to the console to obtain the input data. It has a loose signal limit of zero, so when an event monitor is created for this event, previously signaled console input notifications will not be seen by the corresponding event handler. See [z/VM: CMS Application Multitasking](#) for more information.
21. In linemode CMS, WAIT=YES should be used when an EXIT routine created by the STAX macro exists. WAIT=YES differs from WAIT=NO in that WAIT=NO causes an attention interrupt to be generated by the user when the enter key is pressed.

Return Codes

When LINERD completes, register 15 contains one of the following return codes:

Code

Meaning

0

The function executed successfully.

4

If VNAME='CMS' (the default) is specified and SET FULLSCREEN is OFF or SUSPEND, then an attention interrupt ended the read operation. Otherwise, no windows are showing the virtual screen specified.

12

The function is not valid for the virtual screen specified.

13

Application buffer is full.

24

An incorrect parameter list was specified.

28

The virtual screen does not exist.

88

The virtual device does not support full-screen I/O.

89

The console is a 2741 typewriter terminal.

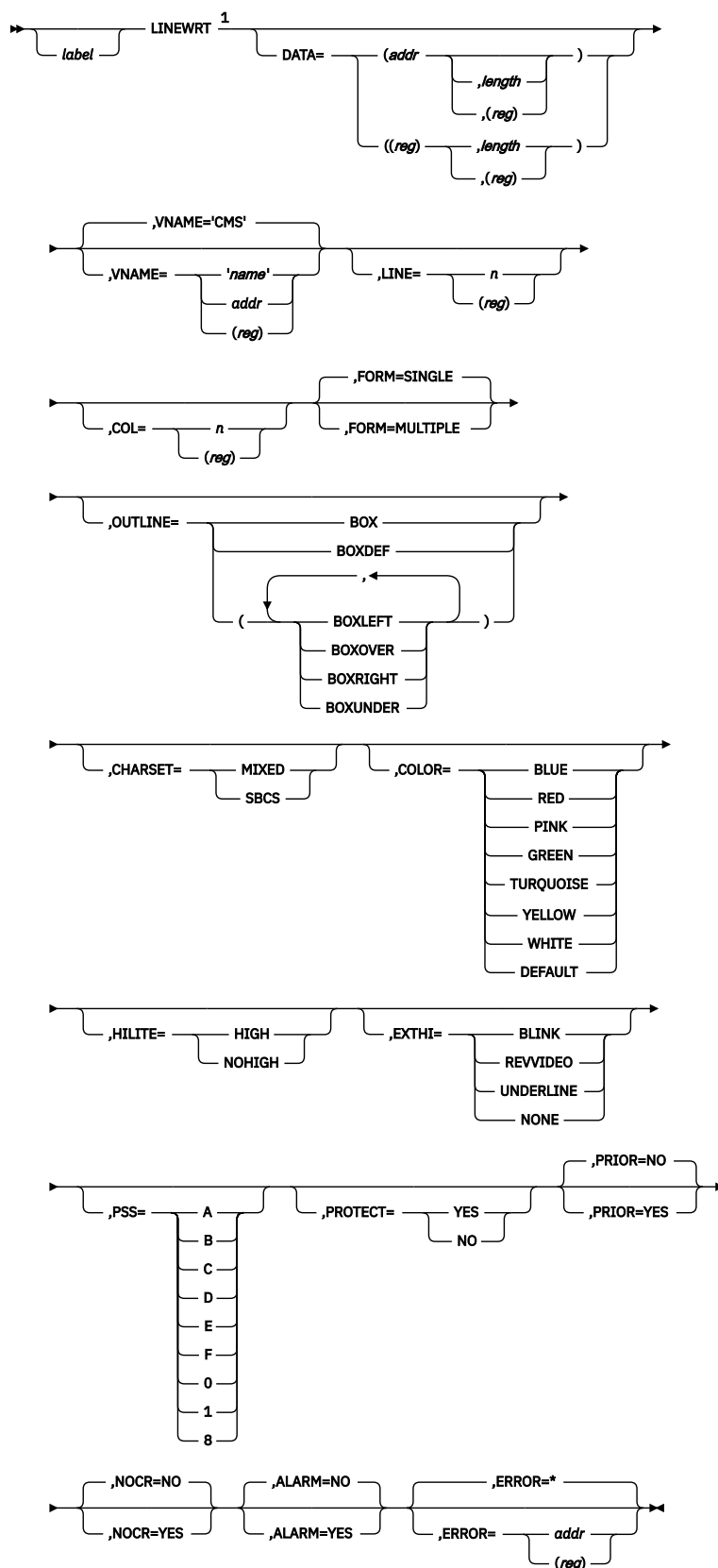
100

I/O error on screen.

104

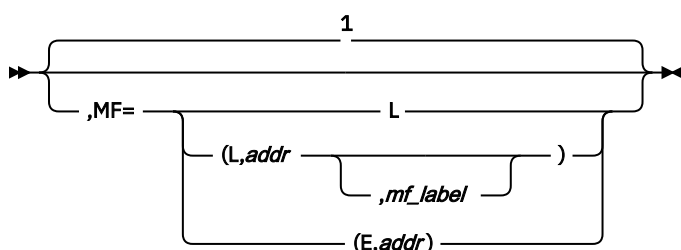
Insufficient storage was available to execute the requested function.

LINEWRT



Notes:

¹ Keyword parameters can be entered in any order.



Notes:

¹ Default is the standard macro format.

Purpose

Use the LINEWRT macroinstruction to display one or more lines of output at the terminal. You can use the LINEWRT macroinstruction for single or multiple writes in full-screen mode (SET FULLSCREEN ON) and for a single output in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND).

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

DATA=

specifies the address and length of the text to be written when the FORM parameter is omitted or specified as SINGLE. The DATA parameter is required with the standard format of the LINEWRT macro. When FORM=MULTIPLE is specified, the address designates the beginning of the chain of output descriptors and the *length* operand is ignored. Acceptable values are:

(addr, length)

specifies the address as an assembler expression and, optionally, the length as an absolute expression.

(addr, (reg))

specifies the address as an assembler expression and, optionally, the length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

((reg), length)

specifies a register that contains the address and the length as an absolute expression. If you use a register to specify the address, you must specify a length.

((reg), (reg))

specifies a register that contains the address and a register that contains the length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

VNAME=

specifies the name of a previously defined virtual screen where CMS writes the data. You can define a virtual screen using the VSCREEN command, which is described in the *z/VM: CMS Commands and Utilities Reference*. If you omit the VNAME parameter, CMS directs the output to the CMS message class, which is displayed in the CMS virtual screen by default. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in quotation marks.

addr

specifies the name as an assembler label.

(reg)

specifies a register that contains the address of an 8-byte name.

LINE=

specifies the line on the virtual screen where CMS writes the data. If you omit the LINE parameter, or specify a value of 0, CMS writes the data on the line after the last line it wrote. LINE must be a nonnegative integer value. Acceptable values are:

n

specifies the line number as an absolute expression.

(reg)

specifies a register (2-12) enclosed in parentheses that contains the line number.

COL=

specifies the column on the virtual screen where CMS writes the data. If you omit the COL parameter, CMS writes the data in the first column of the virtual screen. COL must be a nonnegative integer value. Acceptable values are:

n

specifies the column number as an absolute expression.

(reg)

specifies a register (2-12) enclosed in parentheses that contains the column number.

FORM=

specifies whether more than one output is in the application buffer. Acceptable values are:

SINGLE

means that only one output is specified. This is the default.

MULTIPLE

means that a chain of output descriptors contains the outputs. The first output descriptor only contains cursor information. Each of the other output descriptors represents all of the information required for one output.

OUTLINE

specifies the field outlining for a PS/55-family device. It may be specified as BOXDEF (the default outlining for the device), BOX (a full box), or any combination of the following, to obtain the remaining 14 possible valid values:

BOXLEFT

A vertical line on the left of the field

BOXOVER

Overline

BOXRIGHT

A vertical line on the right of the field

BOXUNDER

Underline

Note: Currently on PS/55-family displays, for fields ending at column 80 of the physical screen outlined on the right (with BOX or BOXRIGHT), the outlining on the right actually appears to the left of column 1 on the next line.

CHARSET=

specifies the attribute for defining mixed DBCS or SBCS fields.

MIXED

specifies a mixed DBCS (with SO/SI positions) field.

SBCS

specifies a single-byte character set field.

COLOR=

specifies the color of the data. Only one of the following keywords for color may be specified:

- BLUE
- RED
- PINK
- GREEN
- TURQUOISE
- YELLOW
- WHITE
- DEFAULT

The DEFAULT keyword specifies the default color of the physical device. If you do not specify the COLOR parameter, CMS uses the default color for the virtual screen.

HILITE=

specifies the highlighting attribute for the data. If HILITE is not specified, the default highlighting for the virtual screen is used. Acceptable values are:

HIGH

indicates bright (or high intensity).

NOHIGH

indicates normal intensity.

For more information on defining default values for virtual screens, see the VSCREEN DEFINE command in the [z/VM: CMS Commands and Utilities Reference](#).

EXTHI=

specifies the extended highlighting attribute for the data. You can specify the EXTHI parameter as one of the following:

- BLINK
- REVVIDEO
- UNDERLINE
- NONE

If you do not specify the EXTHI parameter, CMS uses the default extended highlighting for the virtual screen.

For more information on defining default values for virtual screens, see the VSCREEN DEFINE command in the [z/VM: CMS Commands and Utilities Reference](#).

PSS=

specifies the programmed symbol set CMS uses to write the data. Specify symbol sets as A, B, C, D, E, F, 0, 1, or 8. Only one PSS can be specified. If you specify PSS=1, you must set CHARMODE to ON to display the text. PSS=8 specifies a pure DBCS field. After you specify PSS=8, you cannot change to another PSS value without redefining the field. If you do not specify the PSS parameter, CMS uses the default character set for the virtual screen.

For more information on defining default values for virtual screens, see the VSCREEN DEFINE command in the [z/VM: CMS Commands and Utilities Reference](#).

PROTECT=

specifies whether the data can be typed over. If you do not specify the PROTECT parameter, CMS uses the default for the virtual screen. Acceptable values are:

YES

means the data cannot be typed over.

NO

means the data can be typed over.

For more information on defining default values for virtual screens, see the VSCREEN DEFINE command in the [z/VM: CMS Commands and Utilities Reference](#).

PRIOR=

specifies whether CMS writes the data if CMS halt typing is in effect. Acceptable values are:

NO

indicates that the data is not written if halt typing is in effect. This only has an effect on a virtual screen which is handling message class CMS. (See the ROUTE command in the [z/VM: CMS Commands and Utilities Reference](#) for more information on routing message classes.) This is the default value.

YES

indicates a priority write. The data is written even if halt typing is in effect.

NOCR=

specifies whether CMS sets the cursor in the column following the data written. Acceptable values are:

NO

CMS sets the cursor in the line following the data written. This is the default value.

YES

indicates no carriage return—CMS sets the cursor in the column following the data written.

Note: If CMS is in linemode (SET FULLSCREEN is OFF or SUSPEND) and VNAME='CMS', then trailing blanks are removed when NOCR=NO.

ALARM=

specifies whether the alarm sounds the next time I/O is performed. Acceptable values are:

NO

specifies the alarm does not sound the next time I/O is performed. This is the default value.

YES

specifies the alarm sounds the next time I/O is performed.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The LWRD mapping macro can be used to map the fields of the LINEWRT output descriptors. The *z/VM: CMS Application Development Guide for Assembler* contains an example showing how to use these two macroinstructions to write multiple lines on a single call.
2. Similarities between the LINEWRT and LINERD descriptors allow you to update the vscreen by using the input descriptors (LRDDs) as output descriptors (LWRDs) with the LINEWRT macro.
3. You cannot specify text on the DATA parameter itself; you must specify a buffer that contains the text. To specify text on a macro call, use the APPLMSG macro.
4. The buffer whose address is specified on the DATA parameter should contain only data to be displayed. Any data which is not displayable (such as control characters) is translated according to the SET NONDISP setting.
5. Using LINEWRT with FORM=SINGLE specified to write data into a virtual screen is equivalent to issuing a VSCREEN WRITE command with the FIELD option. For more information on the effects and interactions of parameters when creating fields for a virtual screen, see the VSCREEN WRITE command in the *z/VM: CMS Commands and Utilities Reference*.
6. OUTLINE, CHARSET, COLOR, HILITE, EXTHI, and PSS work only on devices that support them; otherwise they are ignored.
7. When the virtual screen name is CMS (the default virtual screen name), the action taken by LINEWRT depends on the setting of full-screen CMS. If SET FULLSCREEN is ON, LINEWRT writes the data into the CMS virtual screen. If CMS is in line mode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND), LINEWRT calls the WRTERM function to display the output and the LINE, ALARM, COL, OUTLINE, CHARSET, COLOR, HILITE, EXTHI, PSS, and PROTECT parameters are ignored.

When the virtual screen name is not CMS, the setting of full-screen CMS has no effect on the LINEWRT macro.
8. If FULLSCREEN is ON and NOCR=YES is specified, the cursor is positioned in the field defined immediately following the data.
9. Some programs embed hexadecimal code X'1D' to affect the highlighting and color attributes of output in line mode. In full-screen CMS, however, X'1D' is a nondisplayable character and does not affect the attributes of data following it. The LINEWRT macro (as well as the SET VSCREEN, VSCREEN DEFINE, and VSCREEN WRITE commands) lets programs specify attributes for data in full-screen CMS.
10. If the length of the data buffer is specified as 0, then a single blank will be written.
11. The maximum length of data that can be written is the size of the virtual screen you write to.
12. If you write to line zero, any character X'15' is treated as a line-end character; text following an X'15' is written in the next line.
13. The detection of loaded programmed symbol sets occurs when full-screen CMS is initialized (SET FULLSCREEN ON), resumed (SET FULLSCREEN RESUME), or when XEDIT is invoked. Therefore, programmed symbol sets should be loaded prior to invoking these commands. They will then be available for use by full-screen CMS or XEDIT in displaying the screen.

In line mode CMS, programmed symbol sets are detected the first time a window is displayed or when XEDIT is invoked. If XEDIT has not been invoked and Session Services commands are used to display a window, the check to determine if programmed symbol sets are loaded is only done the first time a window is displayed. If programmed symbol sets are loaded after the initial display of a window or after XEDIT has been invoked, you must invoke XEDIT to detect the new programmed symbol sets.
14. PSS=8 lets you define a pure DBCS field in a vscreen when the device is a PS/55-family display with PS8 capability. You can define a pure DBCS field only when you define a field (with the LINEWRT macro or the VSCREEN WRITE command with the FIELD option). After you specify PSS=8, you cannot change to another *psset* value without redefining the field. If you specify both PSS=8 and CHARSET=MIXED or SBCS, PSS=8 will override the other two options; they are ignored.

15. When MIXED is specified, both DBCS and SBCS characters can be displayed in the field. The SBCS and DBCS strings are separated in the field by 1-byte SO/SI control codes.

When SBCS is specified, both SBCS and mixed DBCS data may be displayed in the field. However, you cannot enter SO/SI codes directly from the keyboard.

After you specify MIXED or SBCS, you cannot change the field attributes without redefining the field.

The MIXED and SBCS options are ignored if the PSS=8 option is in effect. If you are writing a MIXED or SBCS field on a PS8 vscreen, be sure to specify PS0 or the PSS value you want. Otherwise, PS8 is assumed, and MIXED or SBCS is ignored.

16. When the LINEWRT macro is invoked and logging is in effect (SET LOGFILE command), CMS converts pure DBCS to mixed DBCS text (by adding SO/SI control codes) before putting the data into a file. This is done so that you can use XEDIT to view or change the file. Logging does not take place until a refresh occurs.

For more information about DBCS adjustments when writing to the virtual screen, see the VSCREEN WRITE command, in the *z/VM: CMS Commands and Utilities Reference*.

17. Any translation done on the input buffer that contains both SBCS and DBCS data will only occur on the SBCS portions of the data provided that the display is capable of supporting mixed DBCS.
18. If FORM=MULTIPLE is specified and CMS is in linemode (SET FULLSCREEN is OFF or SUSPEND) and VNAME='CMS', then descriptors that write color codes, extended highlighting codes, or pss codes (values of LWRDCLRT, LWRDEXHT, or LWRDPSST for LWRDTEXT) are ignored. Also, descriptors that write to the reserved area of a virtual screen are ignored.
19. CMS signals the VMCONINPUT event whenever it receives unsolicited input from the virtual machine console. It is a broadcast event with session scope and does not synchronize the handling process. It contains no event data. The monitoring application should perform a read operation to the console to obtain the input data. It has a loose signal limit of zero, so when an event monitor is created for this event, previously signaled console input notifications will not be seen by the corresponding event handler. See *z/VM: CMS Application Multitasking* for more information.

Return Codes

When LINEWRT completes, register 15 contains one of the following return codes:

Code

Meaning

0

The function executed successfully.

12

The function is invalid for the virtual screen specified.

24

The parameter list is invalid.

28

The virtual screen is not defined.

32

The specified line or column is outside the virtual screen.

104

Insufficient storage was available to execute the requested function.

LRDD

► LRDD ◄

Purpose

Use the LRDD macroinstruction in conjunction with the LINERD macro to map the LINERD descriptors for multiple inputs.

Usage Notes

1. For more information on the LRDD macro, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The LRDD mapping macro expands as follows:

```

LRDD      DSECT
LRDDNEXT  DS      A           Pointer to next LINERD descriptor (LRDD)
          DS      CL8         Reserved
LRDDLINE  DS      F           Line number
LRDDCOL   DS      F           Column number
LRDDTXTA  DS      A           Text address                for input LRDD
LRDDXTL   DS      F           Length of text following this LRDD
*                                     for input LRDD
          DS      CL4         Reserved
*
LRDDFLG1  DS      XL1         Flag byte #1                for input LRDD
LRDDRESI  EQU    X'01'        .... ...X Reserved area of vscreen
LRDDOUTL  DS      XL1         Field outlining byte         for input LRDD
LRDDCSET  DS      XL1         MIXED/SBCS field attribute  for input LRDD
          DS      CL6         Reserved
LRDDATTR  DS      XL1         Attribute byte              for input LRDD
LRDDCOLR  DS      XL1         Color byte                   for input LRDD
LRDDEXHI  DS      XL1         Extended highlighting byte  for input LRDD
LRDDPSS   DS      XL1         PSS byte                     for input LRDD
*
LRDDFLG2  DS      XL1         Flag byte #2                for cursor LRDD
LRDDRESC  EQU    X'01'        .... ...X Reserved area of vscreen
*
          DS      XL1         Reserved
*
LRDDKEY   DS      XL1         Holds key pressed           for cursor LRDD
LRDDLEN   EQU    *-LRDD       Length of LRDD in bytes
LRDDBSZ   EQU    ((LRDDLEN+7)/8) Length of LRDD in doublewords
*

```

Note: The next three usage notes pertain to the cursor & key descriptor only.

3. If the cursor is not located in the vscreen, the value in both LRDDLINE and LRDDCOL is -1. If the cursor is located between the top and bottom of the vscreen, the line and column are returned. If the cursor is located on the line following the bottom line, the column number is returned in LRDDCOL and LRDDLINE is set to 0. If the cursor is located below the line following the bottom line, LRDDLINE is set to 0 and LRDDCOL is set to 2.
4. If the LRDDRESC flag of the LRDDFLG2 field is not set, the cursor is in the scrollable area of the vscreen. If this flag is set, the cursor is in the top reserved area of the vscreen if the line number is a positive value, and if the line number is negative, the cursor is in the bottom reserved area.
5. The LRDDKEY field returns the hexadecimal value of the key pressed. These values are documented in the *IBM 3270 Information Display System Data Stream Programmer's Reference*. The valid key values are:

Code	Key	Code	Key
X'7D'	ENTER	X'C3'	PF15
X'F1'	PF1	X'C4'	PF16

Code	Key	Code	Key
X'F2'	PF2	X'C5'	PF17
X'F3'	PF3	X'C6'	PF18
X'F4'	PF4	X'C7'	PF19
X'F5'	PF5	X'C8'	PF20
X'F6'	PF6	X'C9'	PF21
X'F7'	PF7	X'4A'	PF22
X'F8'	PF8	X'4B'	PF23
X'F9'	PF9	X'4C'	PF24
X'7A'	PF10	X'6C'	PA1
X'7B'	PF11	X'6E'	PA2
X'7C'	PF12	X'6B'	PA3
X'C1'	PF13	X'6D'	CLEAR
X'C2'	PF14		

Note: The following usage notes pertain to the input descriptors only.

- The text always immediately follows the LRDD input descriptor, and LRDDTXTA points to that location.
- The input length returned in LRDDTXTL may be shorter than the field displayed for user modification in the vscreen (trailing nulls are removed). If only part of a field from a virtual screen is displayed on the physical screen and the field is modified, the entire field is returned. If LRDDTXTL is zero, no text follows the descriptor.
- If the LRDDRESI flag of the LRDDFLG1 field is not set, the input was read from the scrollable area of the vscreen. If the flag is set, the input was read from the top reserved area of the vscreen if the line number is a positive number, and if the line number is a negative number, the input was read from the bottom reserved area.
- LRDDCSET returns the character set of the input. X'00' indicates a mixed DBCS field, and X'01' indicates a SBCS field, or a pure DBCS (PSS 8) field.
- Valid values for the LRDDOUTL, LRDDATTR, LRDDCOLR, LRDDXHI, and LRDDPSS fields are documented in the description of the LWRD macroinstruction.

LWRD

► LWRD ◄

Purpose

Use the LWRD macroinstruction in conjunction with the LINEWRT macro to map the LINEWRT descriptors for multiple outputs.

Usage Notes

1. For more information on the LWRD macro, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The LWRD mapping macro expands as follows:

```

LWRD      DSECT
LWRDNEXT  DS      A      Pointer to next LINEWRT descriptor (LWRD)
          DS      CL8     Reserved
LWRDLINE  DS      F      Line number
LWRDCOL   DS      F      Column number
LWRDXTXTA DS      A      Text address          for output LWRD
LWRDXTL   DS      F      Length of text        for output LWRD
LWRDFLDL  DS      F      Output length in vscreen for output LWRD
*
LWRDFLG1  DS      XL1     Flag byte #1          for output LWRD
LWRDNTRF  EQU     X'80'   X... .. No nulls translation
LWRDNOTR  EQU     X'40'   .X... .. No user translation
*         EQU     X'20'   ..X... .. Reserved
*         EQU     X'10'   ...X... .. Reserved
LWRDCSEF  EQU     X'08'   .... X... MIXED/SBCS attribute is
                        specified
LWRDOUTF  EQU     X'04'   .... .X.. Field outlining is specified
LWRDPRTY  EQU     X'02'   .... ..X. Priority write
LWRDRESO  EQU     X'01'   .... ...X Reserved area of vscreen
LWRDOUTL  DS      XL1     Field outlining byte   for output LWRD
LWRDCSET  DS      XL1     MIXED/SBCS attribute   for output LWRD
          DS      CL6     Reserved
LWRDATTR  DS      XL1     Attribute byte        for output LWRD
LWRDCOLR  DS      XL1     Color byte            for output LWRD
LWRDEXHI  DS      XL1     Extended highlighting byte for output LWRD
LWRDPSS   DS      XL1     PSS byte              for output LWRD
*
LWRDFLG2  DS      XL1     Flag byte #2          for output LWRD
LWRDPSSF  EQU     X'80'   X... .. PSS is specified
LWRDEXHF  EQU     X'40'   .X... .. Extended highlight is specified
LWRDCLRF  EQU     X'20'   ..X... .. Color is specified
LWRDDATF  EQU     X'10'   ...X... .. Update data buffer
*         EQU     X'08'   .... X... Reserved

LWRDCRSF  EQU     X'04'   .... .X.. Position cursor within field
*         EQU     X'02'   .... ..X. Reserved
LWRDPADF  EQU     X'01'   .... ...X Padding with blanks
*
          ORG      LWRDFLG2  Redefine flag byte #2      for cursor LWRD
LWRDLCUR  DS      X
LWRDSETC  EQU     X'02'   .... ..X. Position cursor using curs LWRD
LWRDRESC  EQU     X'01'   .... ...X Reserved area of vscreen
*
LWRDTEXT  DS      X      Text writes a field,data,color,exthi,pss
***** Valid text codes *****
LWRDFLDV  EQU     X'00'   Define a field with default vscreen attr.
LWRDFLDD  EQU     X'01'   Define a field and use descriptor attr.
LWRDDATT  EQU     X'02'   Text is data to write in predefined field
LWRDCLRT  EQU     X'03'   Text is color codes
LWRDEXHT  EQU     X'04'   Text is extended highlighting codes
LWRDPSST  EQU     X'05'   Text is PSS codes
*
LWRDRC    DS      X      Individual return code

```

```

*****
***      Valid return codes      **
*****
*
LWRDOK   EQU    0           Function executed successfully
LWRDINVP EQU    24          User did not specify descriptor correctly
LWRDINVL EQU    32          Specified line/column is outside vscreen
LWRDNOST EQU   104          Insufficient storage was available
*
LWRDLEN  EQU    *-LWRD      Length of LWRD
LWRDDBSZ EQU    (LWRDLEN+7)/8 Length in doublewords

```

3. When you place the cursor in the scrollable area of the vscreen (LWRDRESC flag of LWRDFLG2 field is not set), the value of LWRDLIN must be greater than or equal to zero. When you specify a line number of zero, the cursor is positioned at the line following the current bottom of the vscreen. If you are positioning the cursor in the reserved area of the vscreen, a positive line number places the cursor in the top reserved area. The lines are numbered from the top down, with the top line being line 1. Specifying a negative line number for the cursor in the reserved area of the vscreen places the cursor in the bottom reserved area, where the lines are numbered from the bottom of the screen up. For example, the bottom line is -1, the second line up is -2, and so on. The line number cannot be zero when positioning the cursor in the reserved area. If the line number is out of the range of the vscreen area, a return code of 32 is set in the LWRDRC field.
4. The value of LWRDCOL must be greater than or equal to zero. Specifying a column number of zero is valid only when positioning the cursor in the scrollable area (in which case it is equivalent to specifying column number 2). If the column number is out of the range of the vscreen area, a return code of 32 is set in the LWRDRC field. A negative value is invalid, and the LWRDRC field is set to 24.
5. Cursor positioning is determined by the two flags of the LWRDFLG2 field in the cursor descriptor as follows:

LWRDSETC indicates whether the cursor is to be positioned as specified in the cursor descriptor. If the flag is set, the cursor is positioned on the specified line and column in the vscreen. If the flag is not set, the vscreen coordinates specified by the LWRD cursor descriptor are ignored, and the cursor is placed according to the VSCREEN CURSOR command or the setting of the LWRDCRSF flag of the LWRDFLG2 field of the last LWRD output descriptor defining a field in the data area. This flag is described in Usage Note “17” on page 314.

LWRDRESC indicates whether the cursor is to be placed in the reserved area of the vscreen. If the flag is set, the cursor is placed in the reserved area.
6. When you write in the scrollable area of the vscreen (LWRDRESO flag of the LWRDFLG1 field is not set), the value of LWRDLIN must be greater than or equal to zero. If an invalid line number is specified, return code 24 is set in the LWRDRC field.
7. The value specified in the LWRDCOL field must be greater than or equal to zero. A negative column number is invalid and results in return code 24 being set in the LWRDRC field.
8. The value specified in the LWRDXTL field must be greater than or equal to zero. If it is zero, the output for the length of LWRDFLDL is filled with the pad character specified by the LWRDPADF flag of the LWRDFLG2 field. The details of padding for each vscreen buffer are explained in Usage Note “18” on page 315. A negative value is invalid and causes a return code of 24 to be set in the LWRDRC field.
9. Use the LWRDFLDL field to specify the length of the output to be written in the vscreen. It must be greater than or equal to zero. If it is zero, the length of the text is used (this is the value in LWRDXTL, plus one byte for the start field if defining a field). If LWRDFLDL is less than the text length, the text is truncated. If LWRDFLDL is greater than the text length, the text is padded. When the output defines a field (text code of LWRDFLDD or LWRDFLDV is specified in the LWRDXTL field), LWRDFLDL is the length of the field. Note that when you are not writing a field, the text is written for the length specified in LWRDFLDL or until the next field is encountered. The length of a field can range from one to the size of the vscreen area (the number of lines times the number of columns). A negative value is invalid and causes a return code of 24 to be set in the LWRDRC field.
10. The flags of the LWRDFLG1 field serve the following purposes:

LWRDNTRF indicates whether nulls are translated when translation of nondisplayable characters is performed. If the flag is **not** set, nulls are translated to the character defined by SET NONDISP.

Setting the flag on indicates that nulls are not to be translated. Nulls used for padding the data buffer (when LWRDPADF is not set) are never translated.

LWRDCSEF indicates whether the MIXED or SBCS attribute is specified in the LWRDCSET field.

LWRDOUTF indicates, when set, that the field outlining is specified in the LWRDOUTL field.

LWRDPRTY indicates whether CMS halt typing (HT) setting is to be respected or ignored by the output. When this flag is set, the output is written to the CMS message class vscreen in CMS fullscreen or is displayed on the terminal in CMS linemode regardless of the HT setting. Output to be written to a reserved area is displayed regardless of the HT setting. For other virtual screens, the flag and HT are ignored.

LWRDRESO indicates whether the output is to be written to the reserved area of the vscreen. If the flag is set, writing will occur in the top reserved area if the line number is a positive value, and if the line number is negative, the output is written to the bottom reserved area. If the flag is not set, the output is written to the scrollable area of the vscreen.

11. Valid values for the LWRDOUTL field, which specifies the field outlining of the output field, are:

X'00'

Device default

X'01'

Underline

X'02'

Vertical line on the right

X'03'

Underline and vertical line on the right

X'04'

Overline

X'05'

Overline and underline

X'06'

Overline and vertical line on the right

X'07'

Overline, underline, and vertical line on the right

X'08'

Vertical line on the left

X'09'

Underline and vertical line on the left

X'0A'

Vertical lines on the left and right

X'0B'

Underline and vertical lines on the left and right

X'0C'

Overline and vertical line on the left

X'0D'

Underline, overline, and vertical line on the left

X'0E'

Overline and vertical lines on the right and left

X'0F'

Complete box

12. LWRDCSET specifies the character set of the field, either MIXED or SBCS. X'00' defines a mixed DBCS field, and X'01' defines a SBCS field. Any other value is accepted as X'00'.

13. LWRDATTR specifies the field attributes of the output. These can be any valid field attributes as documented in the *IBM 3270 Information Display System Data Stream Programmer's Reference*. In order to change the field attribute to the value specified in LWRDATTR, you must redefine the field with the LWRDFLDL text code specified in the LWRDTEXT field of the output descriptor. The bit definitions for 3270 field attributes are as follows:

Bit	Description
0, 1	Make the field attribute an EBCDIC/ASCII translatable graphic character
2	Has the following meanings: <ul style="list-style-type: none"> • 0 Unprotected field • 1 Protected field
3	Has the following meanings: <ul style="list-style-type: none"> • 0 Alphanumeric • 1 Numeric (causes an automatic upshift of data entry keyboard)
Note: Bits 2 and 3 equal to B'11' cause an automatic skip of a protected field.	
4, 5	Have the following meanings: <ul style="list-style-type: none"> • 00 Display/not selector-pen-detectable • 01 Display/selector-pen-detectable • 10 Intensified display/selector-pen-detectable • 11 Nondisplay, nondetectable (not printable)
6	Reserved; must always be 0
7	Modified data tag (MDT); identifies modified fields during Read Modified command operations

14. Valid values for the LWRDCOLR field, which specifies the color of the output, are:

Value	Description
X'00'	Device or field default. See note below.
X'F1'	Blue
X'F2'	Red
X'F3'	Pink (Magenta)
X'F4'	Green
X'F5'	Turquoise (Cyan)
X'F6'	Yellow
X'F7'	White

15. LWRDEXHI specifies the extended highlighting of the output. Valid values are:

Value	Description
-------	-------------

X'00'	Device or field default. See note below.
--------------	--

X'F1'	Blink
--------------	-------

X'F2'	Reverse Video
--------------	---------------

X'F4'	Underscore
--------------	------------

16. LWRDPSS specifies the programmed symbol set of the output. Valid values are:

Value	Description
-------	-------------

X'00'	Device or field default. See note below.
--------------	--

X'C1' through X'C6'	Loadable symbol sets (PSA-PSF)
----------------------------	--------------------------------

X'F1'	Nonloadable symbol set (CHARMODE=ON) (PS1)
--------------	--

X'F8'	Pure DBCS field (also nonloadable) (PS8)
--------------	--

If you specify LWRDPSS of X'F1', you must set CHARMODE to ON to display the text.

Note: When defining a field (text code of LWRDFLDD or LWRDFLDV), the default color, extended highlighting, or PSS is the vscreen default. When writing to a predefined field (text code of LWRDDATT, LWRDCLRT, LWRDEXHT, or LWRDPSSST is specified in the LWRDTEXT field), the default is the color, extended highlighting, or PSS of the field.

17. LWRDFLG2 uses flags to indicate the function to be performed. The details for each flag are:

LWRDPSSF indicates the programmed symbol set code when padding or updating is required for the PSS buffer. When the flag is set, the programmed symbol set specified in LWRDPSS is used; otherwise, the default PSS of the virtual screen is used.

LWRDEXHF indicates the extended highlighting code when padding or updating is required for the extended highlighting buffer. When the flag is set, the extended highlighting specified in LWRDEXHI is used; otherwise, the default extended highlighting of the virtual screen is used.

LWRDCLRF indicates the color code when padding or updating is required for the color buffer. When the flag is set, the color specified in LWRDCOLR is used; otherwise, the default color of the virtual screen is used.

LWRDDATF indicates whether the data buffer of the field is to be updated with a pad character. The pad character is either a blank or a null (X'00') character, based on the LWRDPADF setting. When this flag is set, the pad character indicated by LWRDPADF updates the data buffer of the field. This flag is used only when the output is to write color, extended highlighting, or PSS.

LWRDCRSF indicates whether the cursor is placed within this field. If the flag is set and the text is shorter than the field, then the cursor is placed in the column following the last character of the text. If this flag is not set or the text fills the field, then the cursor is placed in the column following the field. The cursor positioning within a field is ignored when the cursor is set by the cursor descriptor or the VSCREEN CURSOR command or when the output is to write data, color, extended highlighting, or PSS (text code of LWRDDATT, LWRDCLRT, LWRDEXHT, or LWRDPSSST specified in the LWRDTEXT field). If a WRTERM is issued, LWRDCRSF adds a line end character to the end of the line when the flag is set.

LWRDPADF indicates the padding character for the vscreen data buffer when the length of the output (LWRDFLDL) is greater than the length of the text (LWRDXTL). If the flag is set, the data buffer is padded with blanks. Otherwise, it is padded with nulls (X'00').

18. LWRDXT defines a field or writes text containing data, color, highlighting, or PSS codes for each character of a predefined field. If an invalid text code is specified, the output is ignored (NO-OP). When you specify the text code of LWRDCLRT, LWRDEXHT, or LWRDPSSST, CHARMODE must be ON to see the results on the screen. Valid text codes and their definitions are as follows:

LWRDFLDV (code X'00') defines a field in the vscreen at the row and column and for the length specified. All buffers (attribute, data, color, extended highlighting, and PSS) are updated with the new information. The attribute buffer is updated with the default vscreen field attribute. The data buffer is updated with the text associated with the output descriptor. The outlining, color, extended highlighting, and PSS buffers are updated with the attributes specified in the output descriptor or with the default vscreen attributes. If the attribute flags (LWRDOUTF, LWRDCSEF, LWRDCLRF, LWRDEXHF, or LWRDPSSF of the LWRDFLG2 field) are set, the attribute values in LWRDOUTL, LWRDCSET, LWRDCOLR, LWRDEXHI, or LWRDPSS are used, and if the flags are not set, the default attributes of the vscreen are used. When a field is defined, the first character contains the start field. The start field is a one-byte character identifying the attribute for the field. The start field character is protected and cannot be written to. For more information on fields, see the *IBM 3270 Information Display System Data Stream Programmer's Reference*. After a field is defined, you cannot change the field attributes without redefining the field.

LWRDFLDD (code X'01') defines a field in the vscreen in the same way that LWRDFLDV does, except it uses the attribute byte of the output descriptor (LWRDATTR).

LWRDDATT (code X'02') indicates that the text associated with the output descriptor consists of the new character codes for each position within the predefined field in the data buffer. The color, extended highlighting, or PSS buffers can be updated by setting the attribute flags (LWRDCLRF, LWRDEXHF, or LWRDPSSF). In this case, these buffers are updated with LWRDCOLR, LWRDEXHI, or LWRDPSS.

LWRDCLRT (code X'03') indicates that the text associated with the output descriptor consists of the new character codes for each character position within the predefined field. The text is written to the color buffer, and a color code is used for padding if it is required. The padding color code is selected according to the color flag (LWRDCLRF). If LWRDCLRF is set, LWRDCOLR is used for padding, otherwise the default vscreen color is used. The data, extended highlighting, and PSS buffers can be updated for the length of the output. If the LWRDDATF flag is set, the data buffer is updated with blanks or nulls, based on the LWRDPADF setting. The highlighting and PSS buffers are updated with the LWRDCOLR and LWRDPSS codes if the LWRDCLRF and LWRDPSSF flags are set.

LWRDEXHT (code X'04') indicates that the text associated with the output descriptor consists of the new highlighting codes for each character position within the predefined field. The text is written to the extended highlighting buffer, and the padding and updating of the other buffers are performed in the same manner as described in LWRDCLRT.

LWRDPSSST (code X'05') indicates that the text associated with the output descriptor consists of the new symbol set identifier codes for each character position within the predefined field. The text is written to the PSS buffer, and the padding and updating of the other buffers are performed in the same manner as described in LWRDCLRT.

19. LWRDRC is the return code. The return code is set in the LWRDRC field of the output descriptor. The following return codes are possible for each output processed by LINEWRT.

Code

Meaning

0

Function executed successfully

24

User did not specify the descriptor correctly

32

Specified line or column is outside vscreen

104

Insufficient storage was available to execute

If more than one error occurs, the highest return code is put in register 15.

NUCEXT

Purpose

Use the NUCEXT macroinstruction to access the NUCEXT function. The NUCEXT macro provides all the functions available with the NUCEXT function; it also lets you specify the addressing mode of the nucleus extension entry point.

The five basic functions of the NUCEXT macro are:

NUCEXT ANCHOR

Obtains the anchor pointer for the list of SCBLOCKs that describe the current list of nucleus extensions.

NUCEXT CLR

Deletes a nucleus extension from the chain of SCBLOCKs that describe the current list of nucleus extensions.

NUCEXT QUERY

Determines if a nucleus extension is defined.

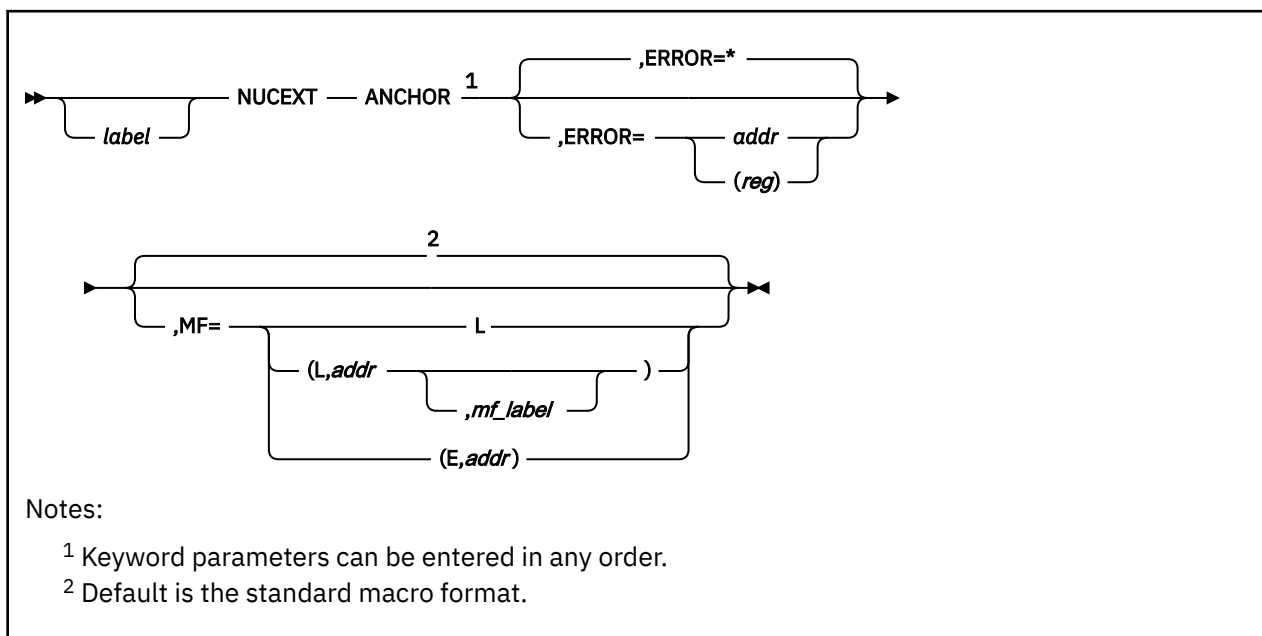
NUCEXT RENAME

Changes the nucleus extension command name field of an SCBLOCK.

NUCEXT SET

Declares a nucleus extension.

NUCXT ANCHOR



Purpose

Use NUCXT ANCHOR to obtain the anchor pointer for the chain of SCBLOCKs that describe the current list of nucleus extension programs.

Parameters

Required Parameter:

ANCHOR

returns in register 1 the pointer to the first entry in the NUCXT chain of SCBLOCKs.

The ANCHOR option requires a read and write parameter list. If you require reentrant code, use the execute form (MF=(E,addr)) of the macro.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

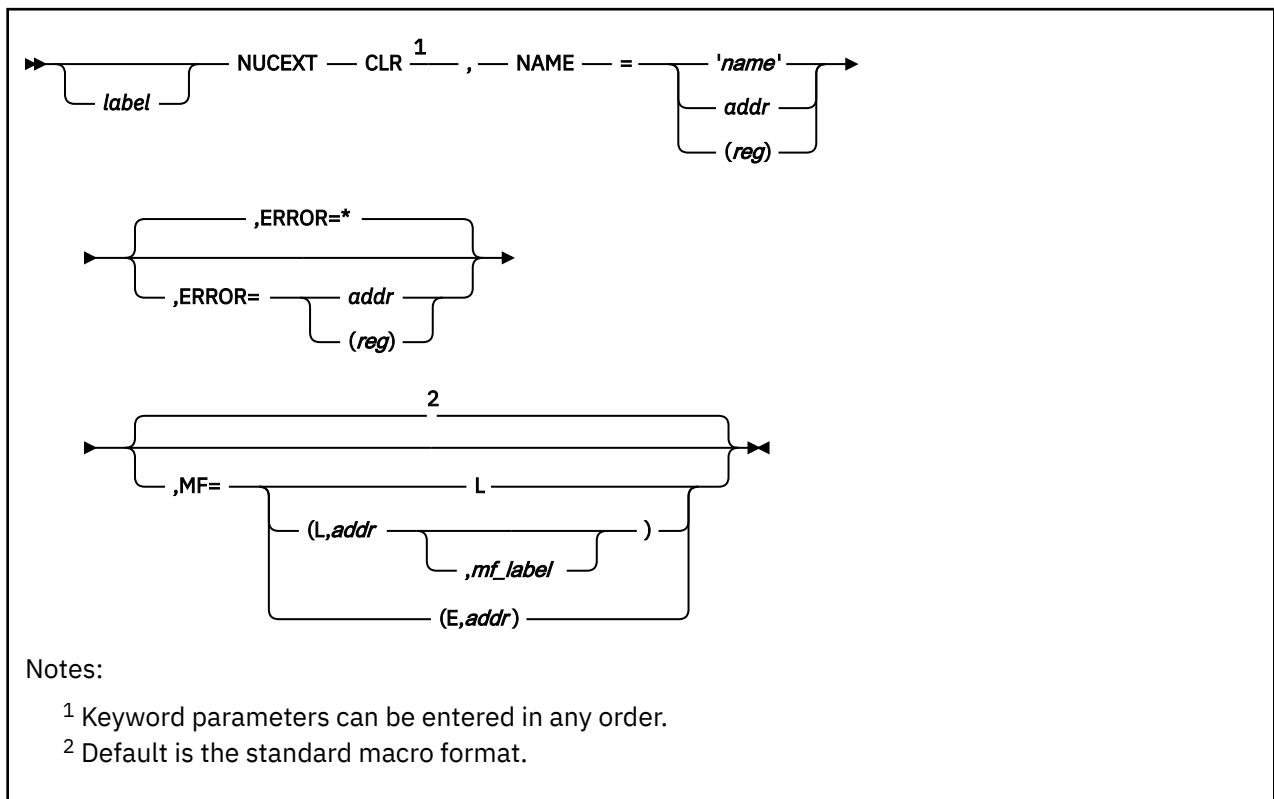
(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

There are no error return codes.

NUCEXT CLR



Purpose

Use NUCEXT CLR to delete a nucleus extension from the chain of SCBLOCKs that describe the current list of nucleus extension programs.

Parameters

Required Parameters:

CLR

deletes the named nucleus extension from the list of nucleus extensions.

NAME=

names the nucleus extension to be cleared. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the name of the nucleus extension. It can be any valid assembler expression.

(reg)

specifies a register that contains the address of the 8-byte storage area containing the name of the nucleus extension. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

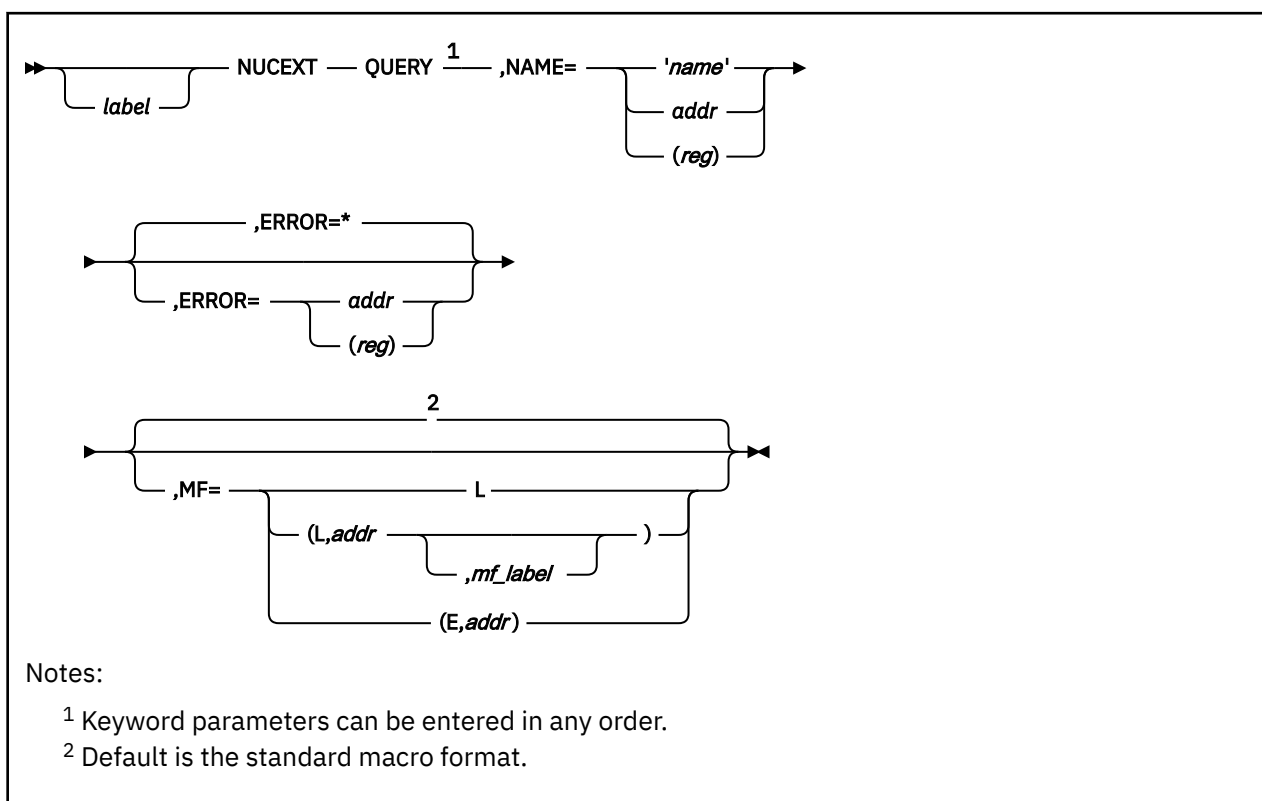
If an error occurs, register 15 contains the following return code:

Code**Meaning**

1

'*name*' is not found.

NUCXT QUERY



Purpose

Use `NUCXT QUERY` to determine whether a specific nucleus extension is currently defined.

Parameters

Required Parameters:

QUERY

returns in register 1 the pointer to the SCBLOCK that describes the named nucleus extension.

The `QUERY` option requires a read and write parameter list. If you require reentrant code, use the execute form (`MF=(E,addr)`) of the macro.

NAME=

names the nucleus extension to be queried. Acceptable values are:

'name'

specifies the name of the nucleus extension as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the name. It can be any valid assembler expression.

(reg)

specifies a register that contains the address of the 8-byte storage area containing the name. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. At abend cleanup time NUCEXT SCBLOCKs are moved.

Return Codes

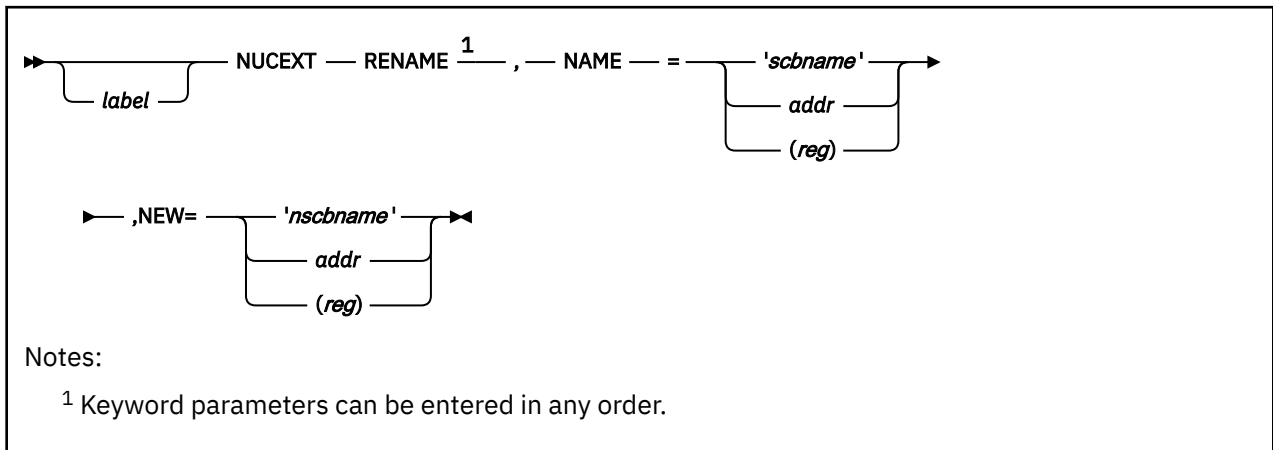
If an error occurs, register 15 contains the following return code:

Code**Meaning**

1

'*name*' is not defined.

NUCEXT RENAME



Purpose

Use NUCEXT RENAME to change the name field of an SCBLOCK for a nucleus extension.

Parameters

Required Parameters:

RENAME

indicates that the RENAME function is desired.

NAME=

specifies the old name of the nucleus extension that is being renamed. Acceptable values are:

'scbname'

specifies the current name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the current name. It can be any valid assembler expression.

(reg)

specifies the register containing the address of the storage area containing the new current name. Valid registers are 2-12 enclosed in parentheses.

NEW=

specifies the new name of the nucleus extension. Acceptable values are:

'nscbname'

specifies the new name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the new name. It can be any valid assembler expression.

(reg)

specifies the register containing the address of the storage area containing the new name. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is any valid assembler language label.

Usage Notes

1. The NUCEXT RENAME function (called from a program through the NUCEXT macro) requires the following PLIST:

```
label    DC CL8'NUCEXT'
          DC CL8'oldname'
          DC CL4                                ignored
          DC AL4(2)                            identifies the rename function
          DC CL8'newname'
```

This changes the name field of the 'oldname' nucleus extension to 'newname'.

Return Codes

If an error occurs, register 15 contains the following return code:

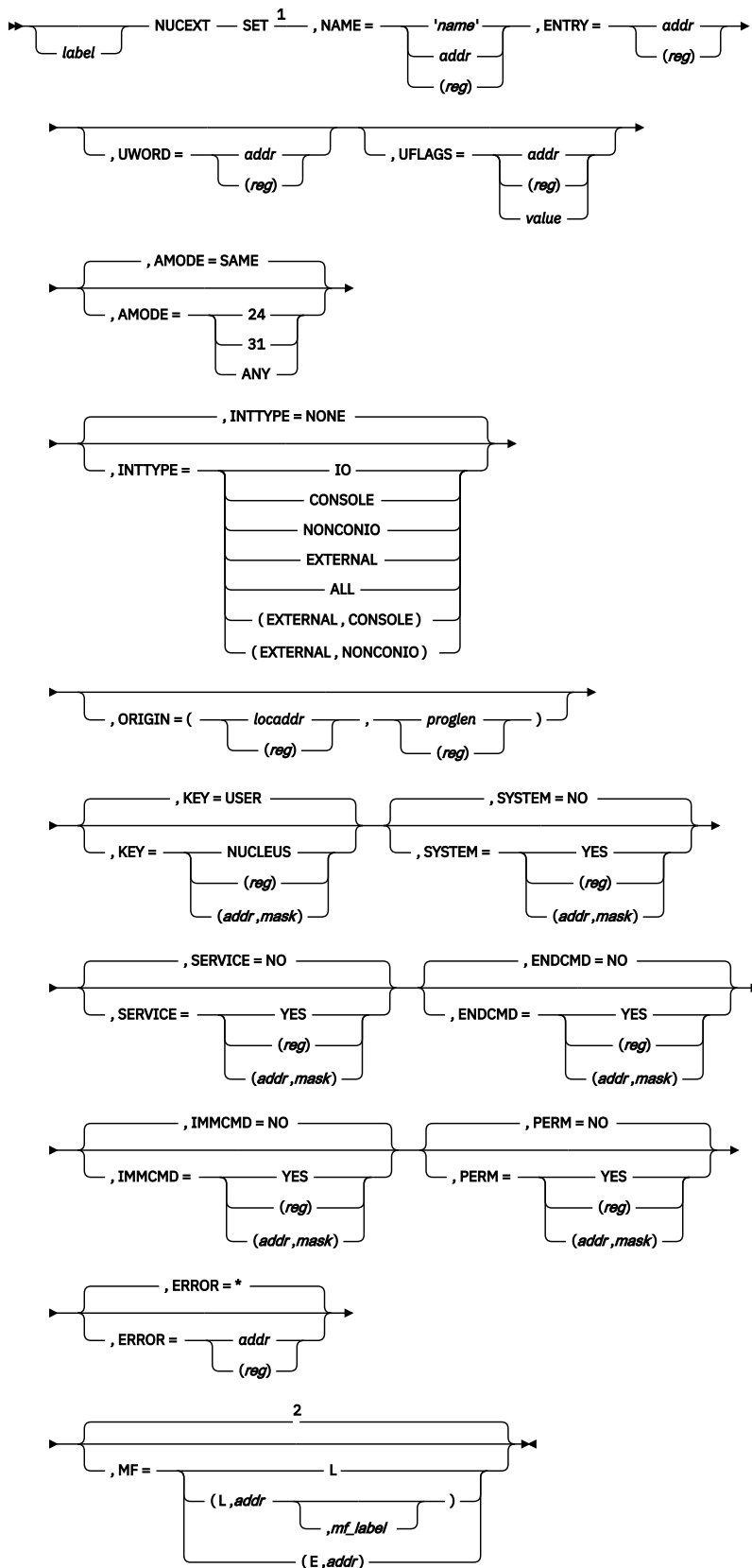
Code

Meaning

1

'oldname' is not found.

NUCEXT SET



Notes:

¹ Keyword parameters be entered in any order.

² Default is the standard macro format.

Purpose

Use the NUCEXT SET macro to declare a nucleus extension.

Parameters

Required Parameters:

SET

declares the named entry point as a nucleus extension.

NAME=

names the nucleus extension to be defined. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the name. This can be any valid assembler expression.

(reg)

specifies a register that contains the address of the storage area holding the name. Valid registers are 2-12 enclosed in parentheses.

ENTRY=

defines the entry point of the nucleus extension. Acceptable values are:

addr

specifies the entry point at the 8-byte storage location defined by *addr*. This can be any valid assembler expression.

(reg)

specifies the entry point at the address contained in the register. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

UWORD=

specifies an optional fullword available in the SCBWKWRD field of the SCBLOCK, which register 2 points to when the nucleus extension is invoked. Acceptable values are:

addr

specifies the *addr* of the UWORD. This can be any valid assembler expression.

(reg)

defines the contents of *(reg)* as the UWORD. Valid registers are 2-12 enclosed in parentheses.

UFLAGS=

specifies an optional 1-byte field available in the SCBUFLAG field of the SCBLOCK, which register 2 points to when the nucleus extension is invoked. Acceptable values are:

addr

specifies the address of the 1-byte UFLAGS field. This can be any valid assembler expression.

(reg)

defines UFLAGS as the contents of low-order byte of *(reg)*. Valid registers are 2-12 enclosed in parentheses.

value

defines UFLAGS as a self-defining 1-byte constant (such as X'01' or C'F').

AMODE=

specifies the addressing mode in which the nucleus extension is entered. Acceptable values are:

SAME

enters the nucleus extension in the same addressing mode as the program that issues the NUCXT macroinstruction. This is the default value.

24

enters the nucleus extension in 24-bit addressing mode.

31

enters the nucleus extension in 31-bit addressing mode.

ANY

enters the nucleus extension in the same addressing mode as the calling routine.

INTTYPE=

specifies the PSW interrupt mask the CMS SVC interrupt handler is to use when invoking the nucleus extension. Acceptable values are:

NONE

disables all interrupts. This is the default value.

ALL

enables all interrupts.

IO

enables all I/O interrupts.

CONSOLE

enables only I/O interrupts from the virtual machine console. The interrupt subclass (ISC) for the console is enabled.

NONCONIO

enables only nonconsole I/O interrupts. All ISCs except for the console ISC are enabled.

EXTERNAL

enables external interrupts.

(EXTERNAL, CONSOLE)

enables external interrupts and I/O interrupts from the virtual machine console. The interrupt subclass (ISC) for the console is enabled.

(EXTERNAL, NONCONIO)

enable for external interrupts and nonconsole I/O interrupts. All ISCs except for the console ISC are enabled.

See [“ENABLE” on page 178](#) for more information on the INTTYPE parameter.

ORIGIN=

specifies the location and length (in bytes) of the program in virtual storage. NUCXDROP and CMS abend processing use this value to remove the nucleus extension program from storage. If the length of the program is specified as zero, NUCXDROP and CMS abend processing will not attempt to free the module's storage. Acceptable values are:

locaddr

specifies the origin location as an assembler expression.

(reg)

specifies a general register (2-12) in parentheses that contains the location.

proglen

specifies the program's length.

(reg)

specifies a general register (2-12) in parentheses that contains the length of the program in virtual storage.

KEY=

specifies the storage key in which the routine executes (either NUCLEUS or USER key). Acceptable values are:

USER

sets storage key to USER. This is the default value.

NUCLEUS

sets storage key to NUCLEUS.

(reg)

the macro checks the value of the specified register and, if it is 0, sets KEY to USER. If the register contains a nonzero value, the macro sets KEY to NUCLEUS.

(addr,mask)

defines a single bit in storage that sets the value of the KEY parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then KEY is set to USER. If the bit is 1, then KEY is set to NUCLEUS. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the KEY parameter as

```
KEY=(APPFLAG,X'80')
```

To set the value of the KEY parameter at assembly time, specify KEY=NUCLEUS or KEY=USER. To set the value at execution time, specify KEY=(*reg*) or KEY=(*addr,mask*).

SYSTEM=

indicates whether the nucleus extension survives CMS abend processing. Acceptable values are:

NO

specifies not to save the nucleus extension. This is the default value.

YES

specifies to save the nucleus extension. You should specify SYSTEM=YES if the nucleus extension must reside in storage that is not reclaimed during abend processing or which CMS is unable to reclaim during abend processing without receiving errors.

(reg)

the macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the SYSTEM parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,mask*).

SERVICE=

indicates whether this entry point receives control during CMS abend processing or NUCXDROP. Acceptable values are:

NO

specifies that this entry point does not receive control. This is the default value.

YES

specifies that this entry point does receive control.

(reg)

the macro checks the value of the specified register and, if it is 0, sets SERVICE to NO. If the register contains a nonzero value, the macro sets SERVICE to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the SERVICE parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SERVICE is set to NO. If the bit is 1, then SERVICE is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SERVICE parameter as

```
SERVICE=(APPFLAG,X'80')
```

To set the value of the SERVICE parameter at assembly time, specify SERVICE=YES or SERVICE=NO. To set the value at execution time, specify SERVICE=(*reg*) or SERVICE=(*addr,mask*).

ENDCMD=

indicates whether the nucleus extension receives control at normal end-of-command processing. Acceptable values are:

NO

indicates that the nucleus extension does not receive control. This is the default value.

YES

indicates that the nucleus extension receives control.

(*reg*)

the macro checks the value of the specified register and, if it is 0, sets ENDCMD to NO. If the register contains a nonzero value, the macro sets ENDCMD to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the ENDCMD parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then ENDCMD is set to NO. If the bit is 1, then ENDCMD is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the ENDCMD parameter as

```
ENDCMD=(APPFLAG,X'80')
```

To set the value of the ENDCMD parameter at assembly time, specify ENDCMD=YES or ENDCMD=NO. To set the value at execution time, specify ENDCMD=(*reg*) or ENDCMD=(*addr,mask*).

IMMCMD=

indicates whether the nucleus extension can be invoked as an immediate command. Acceptable values are:

NO

indicates that the nucleus extension cannot be invoked as an immediate command. This is the default.

YES

indicates that the nucleus extension can be invoked as an immediate command.

(*reg*)

the macro checks the value of the specified register and, if it is 0, sets IMMCMD to NO. If the register contains a nonzero value, the macro sets IMMCMD to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the IMMCMD parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then IMMCMD is set to NO. If the bit is 1, then IMMCMD is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the IMMCMD parameter as

```
IMMCMD=(APPFLAG,X'80')
```

To set the value of the IMMCMD parameter at assembly time, specify IMMCMD=YES or IMMCMD=NO. To set the value at execution time, specify IMMCMD=(*reg*) or IMMCMD=(*addr,mask*).

PERM=

indicates whether the nucleus extension is to be loaded *permanently*, that is, whether it can be dropped by NUCXDROP *. Acceptable values are:

NO

indicates that the nucleus extension can be dropped by NUCXDROP *. This is the default.

YES

indicates that the nucleus extension must be named explicitly on NUCXDROP.

(reg)

the macro checks the value of the specified register and, if it is 0, sets PERM to NO. If the register contains a nonzero value, the macro sets PERM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the PERM parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then PERM is set to NO. If the bit is 1, then PERM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the PERM parameter as

```
PERM=(APPFLAG,X'80')
```

To set the value of the PERM parameter at assembly time, specify PERM=YES or PERM=NO. To set the value at execution time, specify PERM=(*reg*) or PERM=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. On entry to a nucleus extension, the register contents are:

Register**Contents**

R0

Address of extended parameter list (if one was provided by the caller)

R1

Address of the command name (and the tokenized parameter list)

R2

Address of SCBLOCK (If the nucleus extension is invoked as an immediate command, R2 contains the address of an IMMBLOCK, not an SCBLOCK.)

R12

Entry point address

R13

User save area. Note that the USECTYP field of the user save area contains call type information. For 24-bit applications, this information is also found in the high-order byte of register 1. If the nucleus extension is called during end-of-command processing (ENDCMD=YES), the call type is X'FE'. If the nucleus extension is called during abend processing (SERVICE=YES), the call type is X'FF'.

R14

Return address

R15

Entry point address

This is the standard entry point convention except that register 2 points to the SCBLOCK.

2. Nucleus extensions invoked as immediate commands must use BR 14 rather than CMSRET to return control. Using CMSRET may cause the program that invoked the immediate command to end, rather than causing just the immediate command itself to end.
3. When a nucleus extension is established by a multitasking application, it becomes associated with the process that created it, while also being known throughout the session. If a thread in another process invokes the nucleus extension, a thread is created in the process that established it to run the nucleus extension. In this way, it runs in the language environment of its process and if it abends, VMERROR event handlers established in that process can attempt recovery. See [z/VM: CMS Application Multitasking](#) for more information.
4. A program that is to be a nucleus extension must not be built with the multitasking initialization routine VMSTART. While a nucleus extension can perform multitasking operations, it cannot be the starting point for a new process. See [z/VM: CMS Application Multitasking](#) for more information.

Return Codes

If an error occurs, register 15 contains the following return code:

Code

Meaning

25

Insufficient storage to allocate SCBLOCK.

NUCON



Purpose

Use the NUCON macroinstruction to generate a mapping of the fields of the nucleus constant area (NUCON) control block that are supported as programming interfaces.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the NUCON macro expansion is labeled NUCON.

Usage Notes

1. The NUCON macroinstruction expands the six fields of NUCON that are supported as a programming interface as follows:

NUCON	DSECT		
ACMSCVT	DS	1F	Address of simulated OS CVT
ADEVTAB	DC	V(addr)	CMS device information
AEXEC	DC	V(addr)	Address of CMS EXEC interface
NUCXFRES	DC	F'0'	Amount of NUCLEUS free storage to survive abend
*			
USERLVL	DS	F	User area; contents returned in reg 0
AUSER	DC	V(USERSECT)	Address of USERSECT

2. The ACMSCVT field contains the address of CMSCVT, the simulated OS CVT.
3. The ADEVTAB field contains the address used to find CON1ECB for the WAITECB macro. It points to device information for CMS. CON1ECB is located at offset X'C' into this device information. CON1ECB is the only field of the device information that is supported as a programming interface.
4. The AEXEC field contains the address of the CMS module that serves as the interface to the CMS exec processors. This address can be used by programs requiring fast-path subcommand processing. For more information, see the [z/VM: REXX/VM Reference](#).
5. The NUCXFRES field is maintained in NUCON for compatibility, however, it is no longer used to contain the amount of free storage to survive an abend.
6. The USERLVL field is reserved for use by the user. Its contents are returned in register 0 after QUERY CMSLEVEL is issued.



- ### 334 z/VM: 7.3 CMS Macros and Functions Reference

Purpose

Use the PARSECMD macroinstruction from an assembler program to parse (and translate) the arguments of a command.

Parameters

Required Parameters:

UNIQID=

specifies the unique identifier of the syntax definition used for parsing. It has a maximum length of 16 characters and is always required. Acceptable values are:

'uniqueid'

specifies the unique identifier within single quotation marks.

(reg)

specifies a register that contains the address of the unique identifier.

addr

specifies the name as an assembler label.

CALLTYP=

is the call type information passed to the parsing facility. (In previous releases of CMS, this was the information that was passed in the high-order byte of register 1.) The call type information for a command invocation is the value found at label USECTYP in the USERSAVE control block. On entry to a program invoked using SVC 202 or CMSCALL, register 13 points to a USERSAVE control block.

To use the call type your program was invoked with, establish addressability to the USERSAVE control block and specify CALLTYP=USECTYP. Acceptable values are:

addr

specifies the address containing the CALLTYP information.

(reg)

specifies the register that contains the address of the CALLTYP information.

Optional Parameters:

label

is an optional assembler label for the statement.

APPLID=

specifies an application identifier such as DMS or OFS. It must be 3 alphanumeric characters enclosed in single quotation marks, and the first character must be alphabetic. The default is DMS, which is the application identifier for CMS.

PLIST=

specifies the address of the tokenized parameter list for the command. Acceptable values are:

(1)

specifies that register 1 contains the address of the tokenized parameter list. This is the default value.

addr

specifies the address of the tokenized parameter list.

(reg)

specifies a general register (2-12) enclosed in parentheses which contains the address of the tokenized parameter list.

Note: In 24-bit addressing mode, as in previous releases of CMS, the high-order byte of the address indicates if an extended parameter list is available at execution time. In 31-bit addressing mode, you must use the CALLTYP parameter to specify this information. (You can use the CALLTYP parameter in 24-bit or 31-bit mode; if you do not specify CALLTYP, PARSECMD assumes that register 1 contains a 24-bit address.) See the description of the CALLTYP parameter for more information.

EPLIST=

specifies the address of the extended parameter list for the command. Acceptable values are:

(0)

specifies that register 0 contains the address of the extended parameter list. This is the default value.

addr

specifies the address of the extended parameter list.

(reg)

specifies a general register (2-12) in parentheses which contains the address of the extended parameter list.

UPPER=

specifies whether the parsing facility translates lowercase alphabetic characters in the tokenized parameter list to uppercase. Acceptable values are:

CMS

specifies that CMS determines whether to perform uppercase translation according to how the module issuing PARSECMD was invoked. If the command is invoked using the same name by which the command is specified, then tokens in the tokenized parameter list are translated to uppercase by the parsing facility (UPPER=YES). If the command is not invoked using the same name by which the command is specified, the parsing facility does not translate the tokens to uppercase (UPPER=NO). UPPER=CMS is the default.

YES

specifies that tokens in the tokenized parameter list should be translated to uppercase by the parsing facility.

NO

specifies that tokens in the tokenized parameter list should not be translated to uppercase by the parsing facility.

MSGDISP=

specifies how CMS handles parsing facility error messages. Acceptable values are:

ERRMSG

writes parser error messages to the terminal according to the current setting of CP SET EMSG. This is the default value.

NONE

specifies that no output occurs and is most useful when used with the MSGBUFF option.

EXECCOMM

returns the message to a variable in the exec that called this module. The complete message is copied into the variable 'MESSAGE', with the first line in 'MESSAGE.1', the second in 'MESSAGE.2', and so on. The number of lines in the message is copied into 'MESSAGE.0'. This can only be used when the module issuing PARSECMD is called from an exec.

var

specifies a variable that defines the message display format to be used.

The variable must be 1 byte long, and the low-order 3 bits of the byte must be set as follows:

```
000 (for ERRMSG)
010 (for NONE)
100 (for EXECCOMM)
```

MSGBUFF=

specifies the buffer for error message text. When the text is copied into the buffer, the length of the message occupies the first byte of the buffer, preceding the text. Place the length of the buffer, not including the length byte, in the first byte of the buffer before the call to PARSECMD is made. Acceptable values are:

0

specifies that there is no buffer. This is the default value.

addr

specifies an assembler program label that is the address of the buffer.

(reg)

specifies a register that contains the address of the buffer.

TRANSL=

specifies whether the parsing facility translates keywords found in the parameter list. Acceptable values are:

CMS

specifies that CMS determines translation status according to how the module issuing PARSECMD was invoked. TRANSL=CMS is the default; use this unless your program performs its own command resolution.

When CMS determines translation status, it uses:

- TRANSL=YES if the specified command name is a translation (or a synonym or abbreviation of a translation) of the command invoked.
- TRANSL=NO if the specified command is a synonym (or an abbreviation of a synonym) set with the SYNONYM command of the command invoked.
- TRANSL=SAME if the command is invoked using the same name by which the command is specified.

For more information on how and when CMS translates or creates a synonym of a command name, see [z/VM: CMS Commands and Utilities Reference](#).

YES

specifies that all keywords should be translated by the parsing facility. In other words, only keywords defined as nl-names in the Definition Language for Command Syntax (DLCS) syntax definition are recognized.

NO

specifies keywords should not be translated by the parsing facility. In other words, only keywords defined as sl-names in the DLCS syntax definition are recognized.

SAME

specifies the parsing facility should determine translation status from the first keyword found whose nl-name and sl-name DLCS definitions are different. This status is then used for any remaining keywords.

TYPICAL=

specifies how the parsing facility is to be called. Acceptable values are:

SVC

indicates the parsing facility should be called by a CMSCALL macroinstruction. This is the default value.

BALR

indicates the parsing facility should be called by a BALR 14,15. Register 13 must point to an 18-fullword save area.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Note:

1. The MF=L and MF=(L,addr,label) formats of the PARSECMD macro generate a data area which is mapped by the PARSERCB macro.
2. The MF=L format of the PARSECMD macro does not substitute default values for the UNIQID, PLIST, or EPLIST parameters.
3. The MF=(E,addr) format of the PARSECMD does not substitute default values for any options **except** for TYPCALL, which defaults to TYPCALL=SVC.

Some IBM-supplied commands also use the PARSFLG parameter for special purposes. Do not use this parameter yourself.

Usage Notes

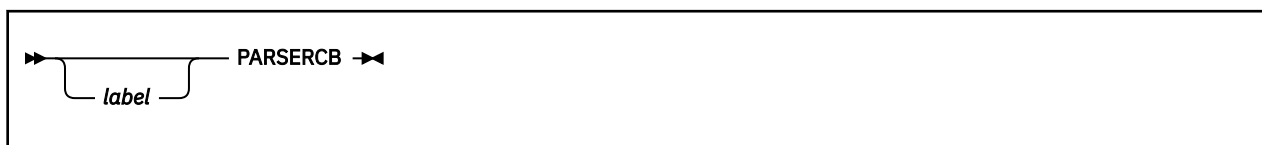
1. The *uniqueid* you specify in the PARSECMD macro is matched up with the *uniqueid* specified in the Command Syntax Definition Language file. For more information on *uniqueids*, see [z/VM: CMS Application Development Guide](#).
2. If you have not issued the standard or MF=L formats of this macro and you are not using a message buffer, code MSGBUFF=0.
3. On exit from the PARSECMD function, general register 1 contains the address of PARSERCB control block. Refer to the PARSERCB macro and the PVCENTRY macro for details on how to obtain the parsed and translated arguments.
4. When you call PARSECMD (standard format and execute format), the parsing facility automatically obtains storage for the parsed (and translated) tokenized and extended parameter lists and the PVCENTRY table. Do not try to free this storage yourself; it is automatically released at SVC 202/CMSCALL termination when the module that invoked PARSECMD returns to its caller. For more information on end-of-SVC, see [z/VM: CMS Application Development Guide for Assembler](#).
5. The parser will do translation when TRANSL=CMS only when called from a program that was invoked from the command line. The translation is assumed to have been done by the caller when the parser is invoked from a program that was invoked by another program.
6. The PARSERCB control block contains several bits for controlling translation and uppercasing. The default setting of these bits requires that the parser set the specific bits describing the actions taken to process a command. (The actions taken will differ depending on the environment.) If these bit settings remain in the PARSERCB, they can then be used by subsequent PARSECMD calls to force the parser to process strings in the same manner it handled the command line. If you are using the execute or complex list forms of PARSECMD, you need to ensure that these bits (and any reserved bits that may be defined in the future) are reset before invoking the parser within a new command environment. This can be done by clearing the PARSERCB storage to binary zeros or by refreshing the working copy of the PARSERCB with a new copy generated by the simple list form.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
24	Syntax error found.
26	Application not active.
28	Syntax definition not found in the command table or user function not found.
30	CALLTYP parameter is required.
104	Insufficient free storage.

PARSERCB



Purpose

Use the PARSERCB macroinstruction to generate a DSECT for the PARSECMD control block.

Parameters

Optional Parameter:

label

is an optional assembler label for the statement. The first statement in the PARSERCB macro expansion is labeled PARSERCB.

Usage Notes

1. For more information on the PARSERCB macro, see the macro [“PARSECMD”](#) on page 334.
2. The PARSERCB macroinstruction expands as follows:

PARSERCB	DSECT				
PARNAME	DS	CL8		Parser entry point	
PARTOKIN	DS	AL4		Input tokenized plist address	
PARTOKPT	DS	AL4		Parsed (translated) tokenized plist address	
*					
PAREPLIN	DS	AL4		Input extended plist address	
PAREPLPT	DS	AL4		Parsed (translated) extended plist address	
*					
PARPTYPE	DS	XL1	F*1	Plist Type-High order byte of R1	
PARTRANS	DS	XL1	F*2	Translation flag	
PARTRYES	EQU	X'80'		Translation = YES (national lang)	
PARTRNO	EQU	X'40'		Translation = NO (system lang)	
PARTRSAM	EQU	X'20'		Translation = SAME (system=national)	
PARSFLG	EQU	X'10'		Parsflg specified	
PARUPYES	EQU	X'08'		Uppercase tokenized plist	
PARUPNO	EQU	X'04'		Copy tokenized plist from eplist	
PARCALT	EQU	X'02'		CALLTYP specified	
PARMSG	DS	XL1	F*3	Message disposition	
PARMSGER	EQU	X'00'		Message disposition is ERRMSG	
PARMSGNO	EQU	X'02'		Message disposition is NONE	
PARMSGXC	EQU	X'04'		Message disposition is EXECOMM	
	DS	XL1	F*4	Reserved	
PARPVCAD	DS	AL4		PVC table address	
PARPVCNM	DS	F		Number of entries in PVC table	
PARMSGAD	DS	AL4		Message buffer address	
PARUNQID	DS	CL16		Syntax definition unique id	
PARAPLID	DS	CL3		Application identifier	
	DS	XL5		Reserved	
PARLENBY	EQU	*-PARSERCB			Length of PARSERCB in bytes
PARLENDW	EQU	(PARLENBY+7)/8			Length of PARSERCB in dwords

3. The PARPVCAD field contains the address of the Parser Validation Code Table. Each entry in this table contains the address, length and validation code for a token in the parsed (and translated) extended parameter list (PAREPLPT). PARPVCNM gives the number of entries in this table; the entries are contiguous. Refer to the PVCENTRY macro for the mapping of each entry.
4. If neither the PARUPYES nor the PARUPNO bit has been set on when the parser is invoked, CMS will determine how to build the tokenized plist based on how the command was invoked and will set the bit appropriate to its choice.
5. A PARSERCB is created by the standard and list formats of the PARSECMD macro and should be filled in with the other formats of the macro.

6. If none of the translation bits (PARTRYES, PARTRNO or PARTRSAM) of the PARTRANS flag has been set on when the parser is invoked, CMS will determine how to build the tokenized PLIST based on how the command was invoked and will set the bit appropriate to its choice.

PARSERUF



Purpose

Use the PARSERUF macroinstruction to generate a mapping to the parser interface for user token validation functions.

Parameters

Optional Parameter:

label

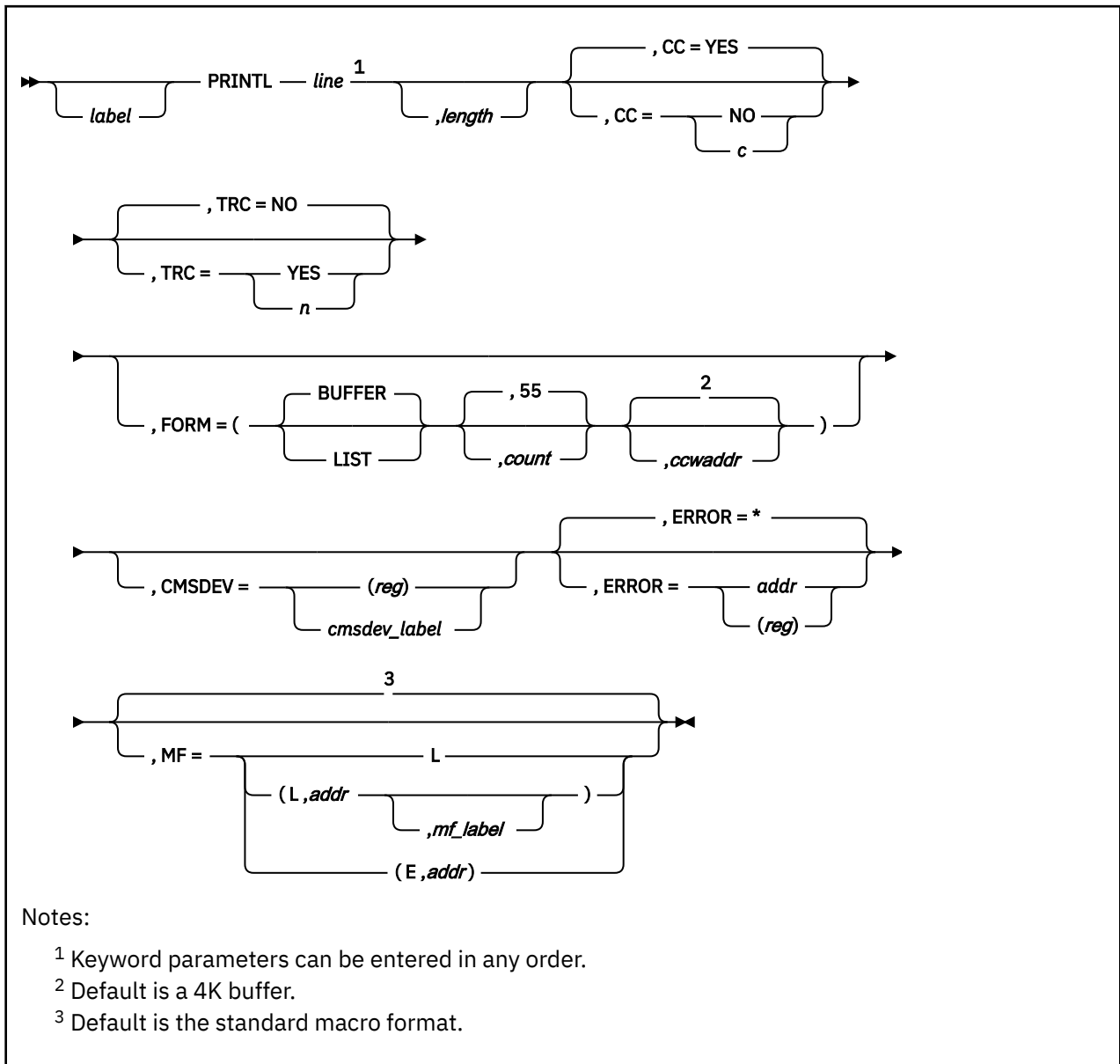
is an optional assembler label for the statement. The first statement in the PARSERUF macro expansion is labeled PARSERUF.

Usage Notes

- 1. For more information on the PARSERUF macro, see [z/VM: CMS Application Development Guide](#).
- 2. The PARSERUF macroinstruction expands as follows:

PARSERUF	DSECT		
PARUNAME	DS	CL8	Name of function
PARUTKAD	DS	A	Address of token
PARUTKLG	DS	F	Length of token
PARUPVC	DS	XL1	User Function Validation Code
	DS	CL7	** RESERVED **
PARUFNCE	DS	CL8'FF'	** RESERVED **
PARUSZBY	EQU	*-PARSERUF	Length in bytes of this block
PARUSZDW	EQU	(PARUSZBY+7)/8	Length in DWORDS of this block

PRINTL



Purpose

Use the PRINTL macroinstruction to write a line or multiple lines to a virtual printer.

Parameters

Required Parameters:

line

specifies one of the following:

- Lines to be printed
- Addresses of the buffer containing the fixed-length records to be printed
- Addresses of the list of variable-length records to be printed.

If you write one line to a virtual printer with each PRINTL instruction and you do not specify FORM=,

line

specifies the line to be printed. Acceptable values are:

'linetext'

text enclosed in quotation marks.

lineaddr

the symbolic address of the line.

(reg)

a register (2-12) containing the address of the line.

If you write multiple lines to a virtual printer with each PRINTL instruction and you specify FORM=BUFFER,

line

specifies the address of the buffer containing the fixed-length records. Acceptable values are:

lineaddr

the symbolic address of the BUFFER.

(reg)

a register (2-12) containing the address of the BUFFER.

If you write multiple lines to a virtual printer with each PRINTL instruction and you specify FORM=LIST,

line

specifies the address of the list of variable-length records to be printed. Acceptable values are:

lineaddr

the symbolic address of the LIST.

(reg)

a register (2-12) containing the address of the LIST.

Optional Parameters:

label

is an optional assembler label for the statement.

length

specifies one of the following two conditions:

- The length of the line to be printed
- The length of the records in the buffer.

If you write one line to a virtual printer with each PRINTL instruction and you do not specify FORM=,

length

specifies the length of the line to be printed (see Usage Note [“1” on page 347](#) for information about line lengths). Acceptable values are:

(reg)

a register (2-12) containing the length.

n

a self-defining term indicating the length.

If you write multiple lines to a virtual printer with each PRINTL instruction and you specify FORM=BUFFER,

length

specifies the length of the records in the BUFFER. Acceptable values are:

(reg)

a register (2-12) containing the length.

n

a self-defining term indicating the length.

If you write multiple lines to a virtual printer with each PRINTL instruction and you specify FORM=LIST, the length of each record is specified in the list and the *length* parameter is ignored.

CC=

specifies whether the records to be printed contain a carriage control character in the first byte. The carriage control character specifies how many lines to skip before the next line prints. Acceptable values are:

YES

specifies that each line to be printed contains a carriage control character. (See Usage Note “2” on [page 347](#) for information on carriage control characters.) This is the default value.

NO

specifies that no carriage control characters are present in the lines to be printed. If you specify CC=NO, the system uses the ASA carriage control character (X'40') to space 1 line before printing.

c

specifies an ASA carriage control character to be used for all lines. CMS assumes the lines to be printed do not contain carriage control characters. See Usage Note “2” on [page 347](#) for valid ASA carriage control characters.

TRC=

specifies whether the current print line includes a TRC (table reference character) byte. The TRC byte indicates which 3800 translate table is selected.

NO

specifies that the line to be printed does not have a TRC byte. This is the default value.

YES

specifies that the line to be printed has a TRC byte. The TRC byte is the second byte when a carriage control byte is present; otherwise, the TRC byte is the first byte. The value of the TRC byte determines which 3800 translate table is selected. If an invalid value is found, translate table 0 is selected.

n

specifies a value for TRC to indicate which 3800 translate table should be selected. The line to be printed does not contain a TRC byte. If an invalid value is specified, translate table 0 is selected.

The value of the TRC byte corresponds to the order in which you have loaded WCGMs (through the CHARS keyword on the SETPRT and SPOOL commands). Valid values for TRC are 0, 1, 2, and 3.

FORM=

specifies that each PRINTL instruction prints multiple records.

BUFFER

specifies that fixed length records are in a buffer. The address of the buffer is specified by the *line* parameter and the number of records in the buffer is specified by *count*. The length of the records is specified by the *length* parameter. If you specify TRC, it applies to all records in the buffer. The *linetext* parameter cannot be used. This is the default value.

LIST

specifies that the addresses of variable length records are in a list. The address of the list is specified by the *line* parameter and the number of entries in the list is specified by *count*. The length of each record is specified in the list and the *length* parameter is ignored. If you specify TRC, it applies to all records in the list. The *linetext* parameter cannot be used.

Each entry in the list is on a fullword boundary and contains 8 bytes:

Bytes**Information****0-3**

Record address

4-5

Reserved

6-7

Record length

count

specifies the number of records to be printed. When FORM=BUFFER, it specifies the number of records in the BUFFER. When FORM=LIST, it specifies the number of entries in the LIST. The maximum number of records a single PRINTL instruction can print is 32,767. Acceptable values are:

n

a self-defining term indicating the number. The default is 55.

(reg)

a register (2-12) containing the number.

countaddr

the address of a halfword containing the number.

ccwaddr

specifies the address of a 4 KB buffer that contains the CCW chains required to perform the requested I/O. If you do not specify this parameter, the system allocates a 4 KB buffer for you. To achieve optimum performance, specify this parameter. Acceptable values are:

label

a label containing the symbolic address of the buffer.

(reg)

a register (2-12) containing the address of the buffer.

CMSDEV=

specifies the 12-byte storage area containing the device characteristics provided by the CMSDEV macro. If not supplied, or if the contents of the area are 0, CMS will perform a DIAGNOSE code X'24' to determine the device type. Acceptable values are:

(reg)

specifies a register (2-12) containing the address of the 12-byte area provided by the CMSDEV macro.

cmsdev_label

specifies the symbolic address of the 12-byte storage area provided by the CMSDEV macro.

Note: Do not specify the CMSDEV= parameter with the list (MF=L) macro form.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The maximum number of data bytes allowed depends on the type of virtual printer:

Table 21. Virtual Printer Maximum Data Bytes	
Virtual Printer Type	Maximum Data Bytes
1403	132
3203	132
3211	150
3800	204
4248	168
VAFP	32767

To determine the line length, add the following to your bytes of data:

- 1 byte for the carriage control character if CC=YES is specified
- 1 byte for the TRC byte if TRC=YES is specified.

If you do not specify the length, it defaults to 133 characters, unless you specify *linetext*. In this case, the length is taken from the length of *linetext*.

Lines greater than the carriage size are not printed and a return code of 1 is issued. However, lines with a carriage control character of X'5A' may have lengths up to 32767 bytes. If you use quoted data with a X'5A' carriage control, the line length must not be greater than 256 bytes.

Note: The record length written to a virtual printer spool file is the maximum data bytes of the spooled device as defined by CP (see Table 21 on page 347). This length is not affected by the *line,length* parameter and cannot be changed by the CMS user.

2. When CC=YES, the first character of the line is interpreted as a carriage control character, which may be either ASA (ANSI) or machine code. The valid ASA control characters are:

Character	Hex Code	Meaning
blank	40	Space 1 line before printing
0	F0	Space 2 lines before printing
-	60	Space 3 lines before printing
+	4E	Suppress space before printing
1	F1	Skip to channel 1
2	F2	Skip to channel 2
3	F3	Skip to channel 3
4	F4	Skip to channel 4
5	F5	Skip to channel 5
6	F6	Skip to channel 6
7	F7	Skip to channel 7
8	F8	Skip to channel 8

Character	Hex Code	Meaning
9	F9	Skip to channel 9
A	C1	Skip to channel 10
B	C2	Skip to channel 11
C	C3	Skip to channel 12

Hex codes X'C1' and X'C3' are used in both machine code and ASA code. CMS recognizes these codes as ASA control characters, not as machine control characters.

Hex code X'5A' is recognized as only a machine code character. This code is used with a composed page data stream record.

When CC=NO or when the line does not begin with a valid carriage control, the line is printed with an ASA carriage control character to space 1 line before printing (ASA X'40').

3. If you specify the TRC= parameter and the virtual printer is not a 3800, the TRC byte is stripped off before the line is printed. If the TRC byte is invalid, PRINTL issues the following MNOTE:

```
MNOTE 8, 'INVALID TRC SPECIFICATION'
```

Translate table 0 is selected if the TRC byte is invalid.

4. For the CMSDEV= parameter, use the CMSDEV macro to obtain printer characteristics and status.
5. All output from the PRINTL macro is directed to device X'00E' regardless of the device type contained in the 12-byte storage area provided by the CMSDEV macro.
6. When PRINTL completes, register 15 contains a 2 if channel 12 was sensed, or a 3 if channel 9 was sensed. If you specify the FORM parameter, channels 9 and 12 are ignored. When channel 9 or channel 12 is sensed, the write operation terminates after carriage spacing but before writing the line. If you want to write the line without additional space, you must modify the carriage control character in the buffer to a code that writes without spacing (ASA code + or machine code 01).

The location on the page being printed and the corresponding channel is defined by the current forms control buffer image being used. For information on how to specify the forms control buffer image for a virtual spooled printer, see the LOADVFCB and SPOOL commands in the [z/VM: CP Commands and Utilities Reference](#) and if you are using a virtual 3800, also see the CMS SETPRT command in the [z/VM: CMS Commands and Utilities Reference](#).

7. You must issue the CP CLOSE command to close the virtual printer file. Issue the CLOSE command either from your program (using CMSCALL) or from the CMS environment after your program completes execution. The printer is automatically closed when you log off or when you use the CMS PRINT command.
8. If the virtual printer is a 4248 with an extended FCB and the duplication option specified, you should check to be sure that the duplication offset contained in the extended FCB declaration is valid for the line length and that the line length is short enough to be duplicated.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

- 1
Line too long.
- 2
Channel 12 punch detected.
- 3
Channel 9 punch detected.

4

Intervention required.

5

Unknown error.

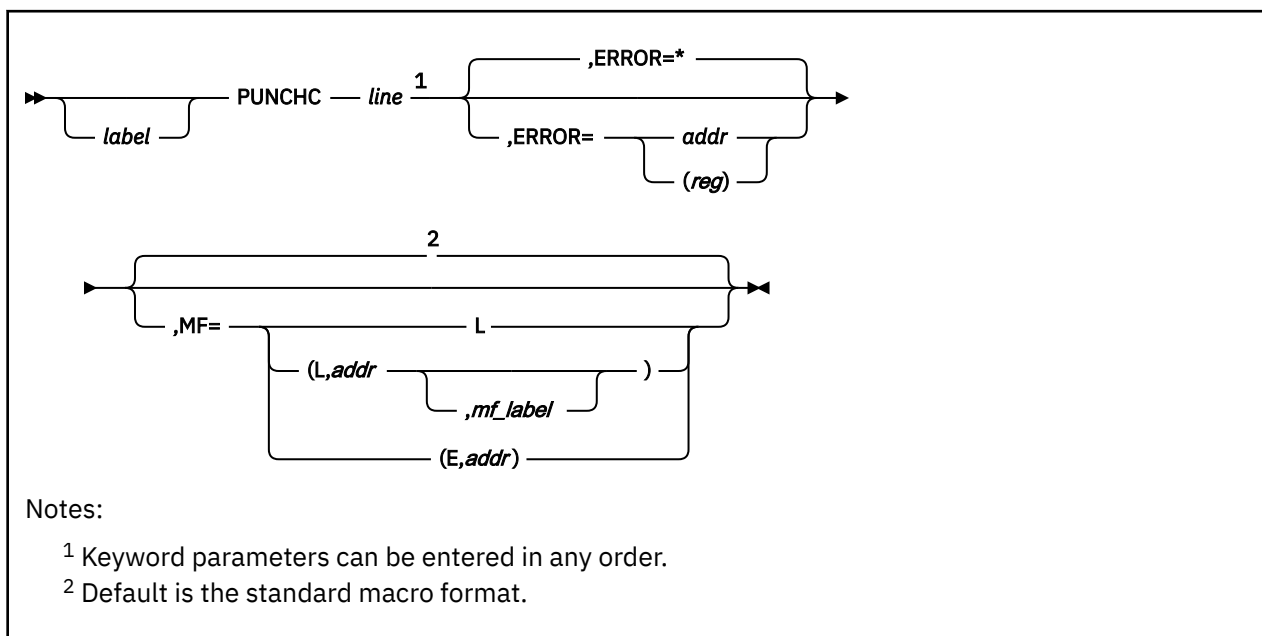
100

Printer not attached.

104

Not enough storage available to successfully complete the program.

PUNCHC



Purpose

Use the PUNCHC macroinstruction to write a line to a virtual punch.

Parameters

Required Parameters:

line

specifies the line to be punched. It may be:

'linetext'

text enclosed in single quotation marks.

lineaddr

the symbolic address of the line.

(reg)

a register containing the address of the line.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. No stacker selecting is allowed. The line length must be 80 characters.
2. You must issue the CP CLOSE command to close the virtual punch file. Issue the CLOSE command either from your program (using the CMSCALL macro) or from the CMS environment when your program completes execution. The punch is closed automatically when you log off or when you use the CMS PUNCH command.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

2

Unit check.

3

Unknown error.

100

Punch not attached.

PVCENTRY



Purpose

Use the PVCENTRY macroinstruction to generate a DSECT for the parser validation code table entry. Each parser validation code table entry contains the address, length, and validation code for a token in the parsed (and translated) extended parameter list.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the PVCENTRY macro expansion is labeled PVCENTRY.

Usage Notes

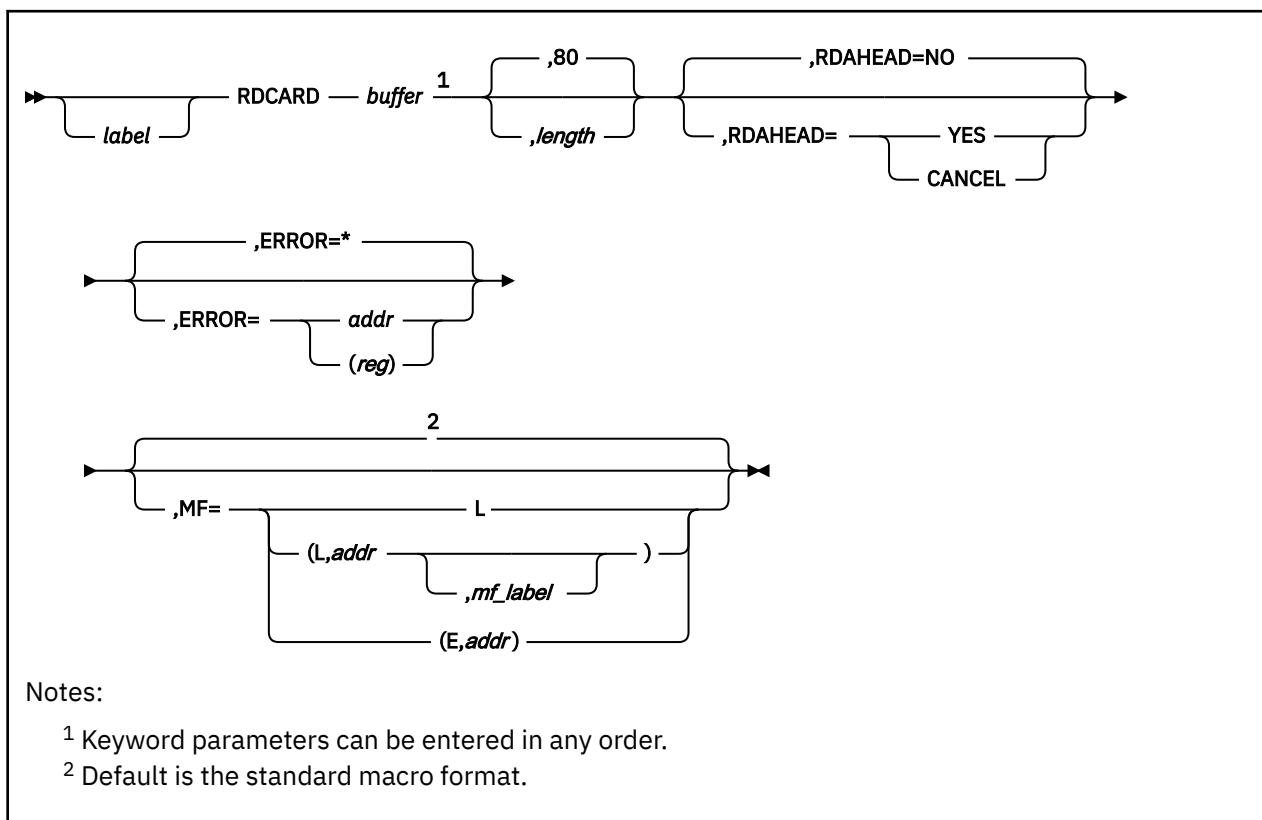
1. For more information on the PVCENTRY macro, see [z/VM: CMS Application Development Guide](#).
2. The PVCENTRY macroinstruction expands as follows:

PVCENTRY	DSECT		Parser Validation Code Entry
PVCNEXTA	DS	A	Next PVC entry address, or 0 if last
PVCCODE	DS	XL1	Parser validation code
	DS	XL3	Reserved
PVCTTOKA	DS	A	Tokenized token address
PVCETOKA	DS	A	Extended token address
PVCETOKL	DS	F	Extended token length
*	EQU	X'00'	Reserved for IBM use
PVCCNAME	EQU	X'01'	Command Name
PVCKWORD	EQU	X'02'	Keyword
PVCOPTST	EQU	X'03'	Option start (
PVCOPTEN	EQU	X'04'	Option end)
PVCCOMMT	EQU	X'05'	Comment
PVCALNUM	EQU	X'06'	Alphanumeric string
PVCCHAR	EQU	X'07'	A single character
PVCCUU	EQU	X'08'	Device address: X'001',X'002',...,X'FFF'
PVCFN	EQU	X'09'	File name
PVCFT	EQU	X'0A'	File type
PVCFN	EQU	X'0B'	File name with '*'
PVCEFT	EQU	X'0C'	File type with '*'
PVCEXECN	EQU	X'0D'	Exec name
PVCEXECT	EQU	X'0E'	Exec type
PVCFM	EQU	X'0F'	File mode
PVCHX	EQU	X'10'	Hexadecimal number
PVCINT	EQU	X'11'	Integer: ..., -2, -1, 0, 1, 2, ...
PVCNINT	EQU	X'12'	Negative integer: ..., -2, -1
PVCPINT	EQU	X'13'	Positive integer: 1, 2, ...
PVCMODE	EQU	X'14'	Alphabetic character
PVCSTRIN	EQU	X'15'	Any character string(no blanks)
PVCTEXT	EQU	X'16'	Any string
PVCDIGIT	EQU	X'17'	Any unsigned integer
PVCAPPID	EQU	X'18'	Application identifier
PVCARBMD	EQU	X'19'	Arbitrary modifier
PVCVDEV	EQU	X'1A'	4 digit device addr
PVCFPOOL	EQU	X'1B'	File pool ID
PVCNMEF	EQU	X'1C'	Namedef
PVCDIRID	EQU	X'1D'	Full dirid
PVCDPOOL	EQU	X'1E'	Dirid, w/o user ID
PVCDUSER	EQU	X'1F'	Dirid, w/o file pool ID
PVCDSUB	EQU	X'20'	Subdir only dirid
PVCDMIN	EQU	X'21'	-fm dirid
PVCDPLUS	EQU	X'22'	+fm dirid
PVCDPATH	EQU	X'23'	path1.path2

PVCDRIDN	EQU	X'24'	Full dirid w/nickname
PVCDUSEN	EQU	X'25'	Dirid w/nickname/userid
*	EQU	X'26'-X'7C'	Reserved for IBM use
PVCINVPD	EQU	X'7D'	Invalid fm, fp, or dir
PVCINVFD	EQU	X'7E'	Invalid fm or dirid
PVCINVLD	EQU	X'7F'	Unconditionally invalid
*	EQU	X'80'-X'FF'	Reserved for customer use

3. The parsing facility creates a table containing contiguous PVCENTRY entries addressed by PARSERCB. See the PARSERCB macroinstruction for details.

RDCARD



Purpose

Use the RDCARD macroinstruction to read a line from a virtual reader.

Parameters

Required Parameters:

buffer

specifies the buffer address where the line is read. Acceptable values are:

bufaddr

the symbolic address of the buffer.

(reg)

a register (2-12) containing the address of the buffer.

Optional Parameters:

label

is an optional assembler label for the statement.

length

specifies the length of card to be read. The minimum length and default value is 80. The maximum length is 204. Specify the length as:

n

a number indicating the length.

(reg)

a register (2-12) containing the length.

RDAHEAD=

specifies whether CMS reads as many lines as possible into an internal I/O buffer before it (CMS) reads each line into the user-specified buffer. Acceptable values are:

NO

does not read multiple lines into an internal I/O buffer. This is the default value.

YES

reads multiple lines into an internal I/O buffer. See Usage Notes 5 and 6.

CANCEL

releases the internal I/O buffer used for RDAHEAD=YES. Any lines in the buffer are lost.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. No stacker selecting is allowed.
2. When the RDCARD macro completes, register 0 contains the length of the card that was read.
3. Do not use the RDCARD macro in jobs that run under the CMS batch machine.
4. If the reader file being processed contains carriage control characters, the RDCARD macro returns the records with the carriage control characters stripped off.
5. If you specify RDAHEAD=YES and the virtual card reader is closed before an error condition is detected (other than wrong-length record, return code=5), lines may still remain in the buffer. Subsequent RDCARD calls return the next available lines from the internal buffer until it is empty. Changes in the status of the virtual card reader are not recognized until the buffer is empty and the next physical read is performed. For most applications that read to end-of-file, RDAHEAD=YES should be specified.

To make sure that the internal I/O buffer is released and that the next RDCARD request reads from the virtual reader, not the internal buffer, issue RDCARD with RDAHEAD=CANCEL and a length of 0.

6. RDAHEAD=NO is forced if the logical record length is greater than 2028, or if there is insufficient storage to allocate the internal I/O buffer.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code**Meaning****1**

End of file.

2

Unit check.

3

Unknown error.

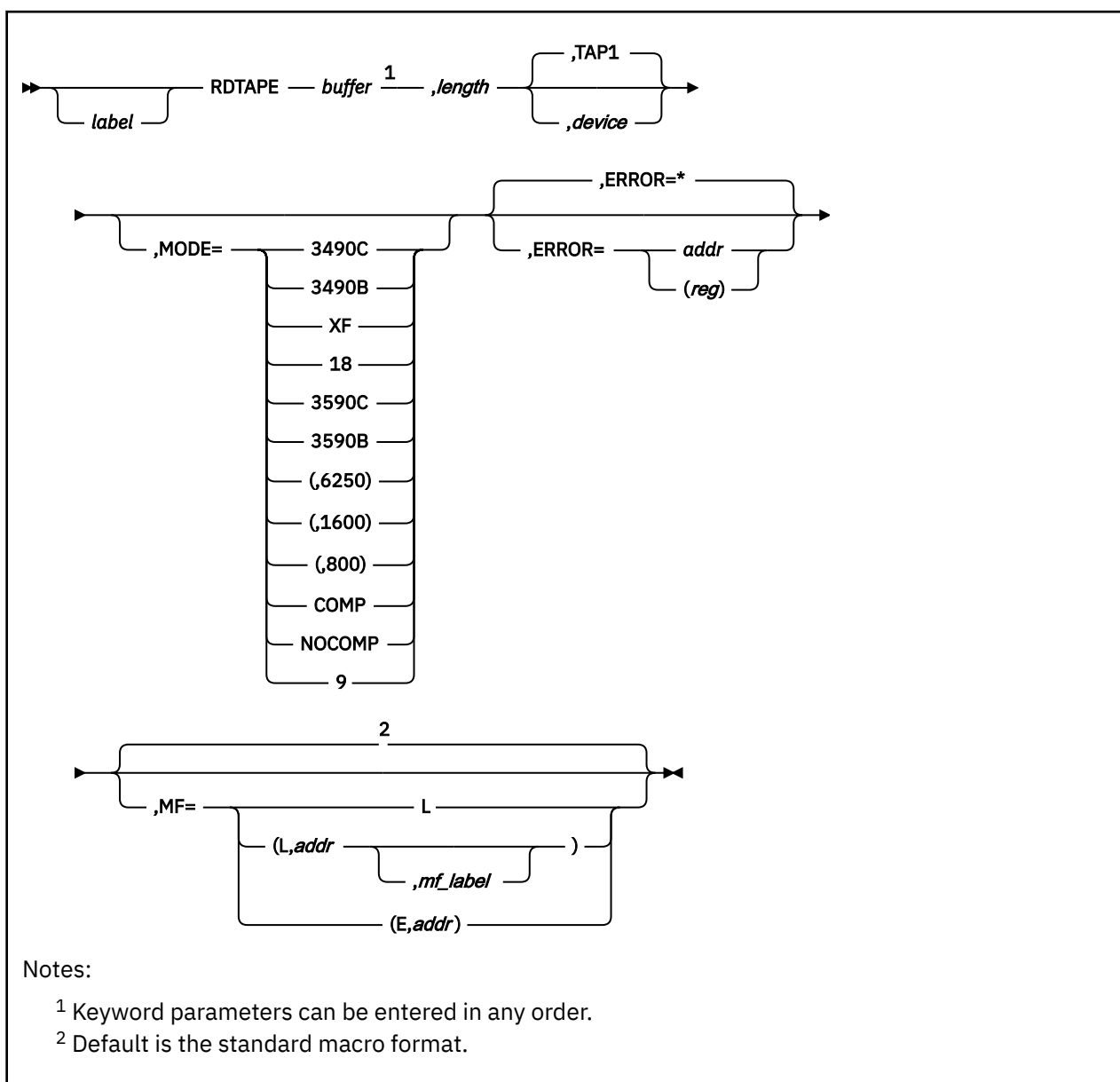
5

Length not equal to requested length.

100

Device not attached.

RDTAPE



Purpose

Use the RDTAPE macroinstruction to read a block from the specified tape device.

Parameters

Required Parameters:

buffer

specifies the address of the buffer into which the block is to be read. It can be:

lineaddr

the symbolic address of the buffer.

(reg)

a register (2-12) containing the address of the buffer.

length

specifies the length of the buffer into which the block is to be read. If the block is larger than the buffer, CMS truncates it. Acceptable values are:

- n**
a number indicating the length.
- (reg)**
a register (2-12) containing the length.

Optional Parameters:

label
is an optional assembler label for the statement.

device
specifies the device name (TAP*n*) or virtual device number (*vdev*) of the virtual tape device from which the block is to be read. The following values are valid; see [z/VM: CMS User's Guide](#) for more information on tape device names and virtual device numbers for tape devices.

Device Name	Virtual Number	Device Name	Virtual Number
TAP0	0180	TAP8	0288
TAP1	0181	TAP9	0289
TAP2	0182	TAPA	028A
TAP3	0183	TAPB	028B
TAP4	0184	TAPC	028C
TAP5	0185	TAPD	028D
TAP6	0186	TAPE	028E
TAP7	0187	TAPF	028F

If you omit the *device* value, CMS uses device TAP1.

- (reg)**
a register containing a pointer to a storage location that contains the device name or virtual device number.

The following example shows how you might use the register form to identify the device:

```

      LA      2, MY181      Addr of device assignment
RDTAPE INBUF,4096,(2),ERROR=MYMSG
*                               Read block(4096 bytes)
      .
      .
MY181 DC      CL4'0181'      vdev definition
```

MODE=

This parameter indicates a recording format. It has no effect other than to make execution of the macro expansion fail if the tape device is not capable of writing that recording format (note that the RDTAPE macro does not cause writing on the tape under any circumstances). The MODE parameter exists largely for compatibility purposes.

See the description of the MODE parameter on the [“WRTAPE” on page 436](#) macro for details on coding of the MODE parameter.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

- ***
passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). See *z/VM: CMS Macros and Functions Reference* for details on the MF parameter. Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Note

1. The maximum supported tape block length is recorded in NUCON field MAXTAPBS. If the value in this field is 0, the maximum block length is 65,535 bytes.

Return Codes

The return codes (found in register 15) from a RDTAPE call are as listed below.

Code**Meaning****0**

The RDTAPE call executed normally.

1

Invalid function or parameter list or the specified device is incapable of writing in the specified format.

2

Tape mark detected.

3

I/O error.

4

Invalid *device* value.

5

Virtual tape device not attached (device does not exist).

7

Specified device is not a tape device.

8

The block read is larger than the buffer provided.

9

Manual rewind/unload of tape.

Return Code 0: RDTAPE executed normally. A block has been successfully read. The data is in the buffer specified in the RDTAPE parameters and the volume is positioned one block ahead of where it was. The block on the volume is smaller than or the same size as your buffer. If it is smaller, CMS has placed it at the beginning of your buffer and not modified bytes in the buffer past the end of the block.

When the macro completes, register 0 contains the number of bytes read.

Return Code 1: Invalid parameter or bad format. One of the following is true of the RDTAPE call:

- One of the parameter values is not valid.
- The parameter values are not compatible with each other.
- The MODE parameter indicates a recording format which the device (identified by the *device* parameter) is not capable of writing. Note that the RDTAPE call will always fail if this is true, even though RDTAPE never attempts to write on the tape. This fact, and the existence of the MODE parameter at all is for compatibility purposes.

For the invalid parameter cases, the RDTAPE invocation needs to be corrected. For the recording format incapable case, it is usually best to eliminate the MODE parameter.

Return Code 2: Tape mark detected. The volume was positioned to a tape mark, rather than a data block. The volume is now positioned past the tape mark. No data has been read.

A tape mark often marks the end of a file or the end of recorded data on the volume.

Return Code 3: I/O error. The device was unable to read a block or tape mark for one of many reasons. Following is a list of some of these reasons. You should not consider any data to have been read and cannot assume any particular positioning of the device. The buffer indicated by the *buffer* parameter may have been modified:

- The device or channel has detected an internal malfunction in the device or channel.
- There is a defect on the recording medium.
- The data on the tape was written in error.
- The tape reel or cartridge is damaged.
- The device was positioned past the end of recorded data. The end of recorded data is defined as the point just after the block, tape mark, or gap that was most recently written on the tape. Note that you will not necessarily get this return code when this is the case. On newer devices which place a definitive End of Data mark on the volume, Return Code 3 is guaranteed. But on older devices there are several other return codes you could get, including 0, so you must use other means to know where the end of recorded data is.
- The tape or a block on it is recorded in a recording format which the device is incapable of reading, or does not even recognize. Another device may be able to read it.
- The block which RDTAPE would have read is too large for the device to handle. Another device might be able to read it.
- The device is in a Volume Fenced condition. This is a condition which arises for reasons in which the device will not perform most operations on the volume. You can undo this condition by unloading the device; other times by rewinding the device. You can do either of these with the TAPE command or TAPECTL macro.
- The virtual device is a *shareable* one (see [z/VM: CMS User's Guide](#)). CMS does not support shareable devices and the failure of RDTAPE in this way is just one of the possible effects.

Return Code 4: Invalid device value. The value of the *device* parameter is not a valid selection. CMS cannot tell from what device to read. The RDTAPE invocation must be corrected.

Return Code 5: Device not attached. No virtual device exists with the virtual device number given by the *device* parameter or, if *device* specifies a device name, with the device number CMS associates with that name.

You must either specify a different device name or number or create one with the proper virtual device number. The [z/VM: CMS User's Guide](#) explains this.

Return Code 7: Device is not a tape device. The device which has the device number given by the *device* parameter or, if *device* specifies a device name, with the device number CMS associates with that name, is not a tape device.

You must either specify a different device name or number or detach the attached device and specify a tape device instead with that virtual device number.

Return code 8: Block larger than buffer. CMS has successfully read a block and the volume is positioned one block ahead of where it was. The block was larger than the buffer you provided, as indicated by the *length* parameter, and CMS has truncated the block to fit.

Note that CMS has read the entire block, regardless of the size of your buffer. The next read will read from the beginning of the next block on the volume.

When the macro completes, register 0 contains the number of bytes read.

Return Code 9: Manual rewind/unload. Someone has previously rewound or unloaded the volume on the real device associated with the virtual device by operating manual controls on the physical device. In order to warn you of this, CMS has returned this return code to RDTAPE without attempting to read anything. Your buffer has not been modified and the position of the volume has not changed.

You get this warning once, so if you want to read the block, you can just repeat the RDTAPE call.

CMS gives you this warning because the volume you intended to read may not be mounted now.

With older devices, you do not get this warning.

REGEQU



Purpose

Use the REGEQU macroinstruction to generate a list of EQU (equate) statements to assign symbolic names for the general, floating-point, extended control, and access registers.

Parameters

Optional Parameters:

- AREGS=**
indicates whether equate statements should be generated for access registers. If it is omitted, equate statements are not generated. Acceptable values are:
- NO**
prevents generation of equate statements for access registers. NO is the default.
 - YES**
causes equate statements to be generated for access registers.

Usage Notes

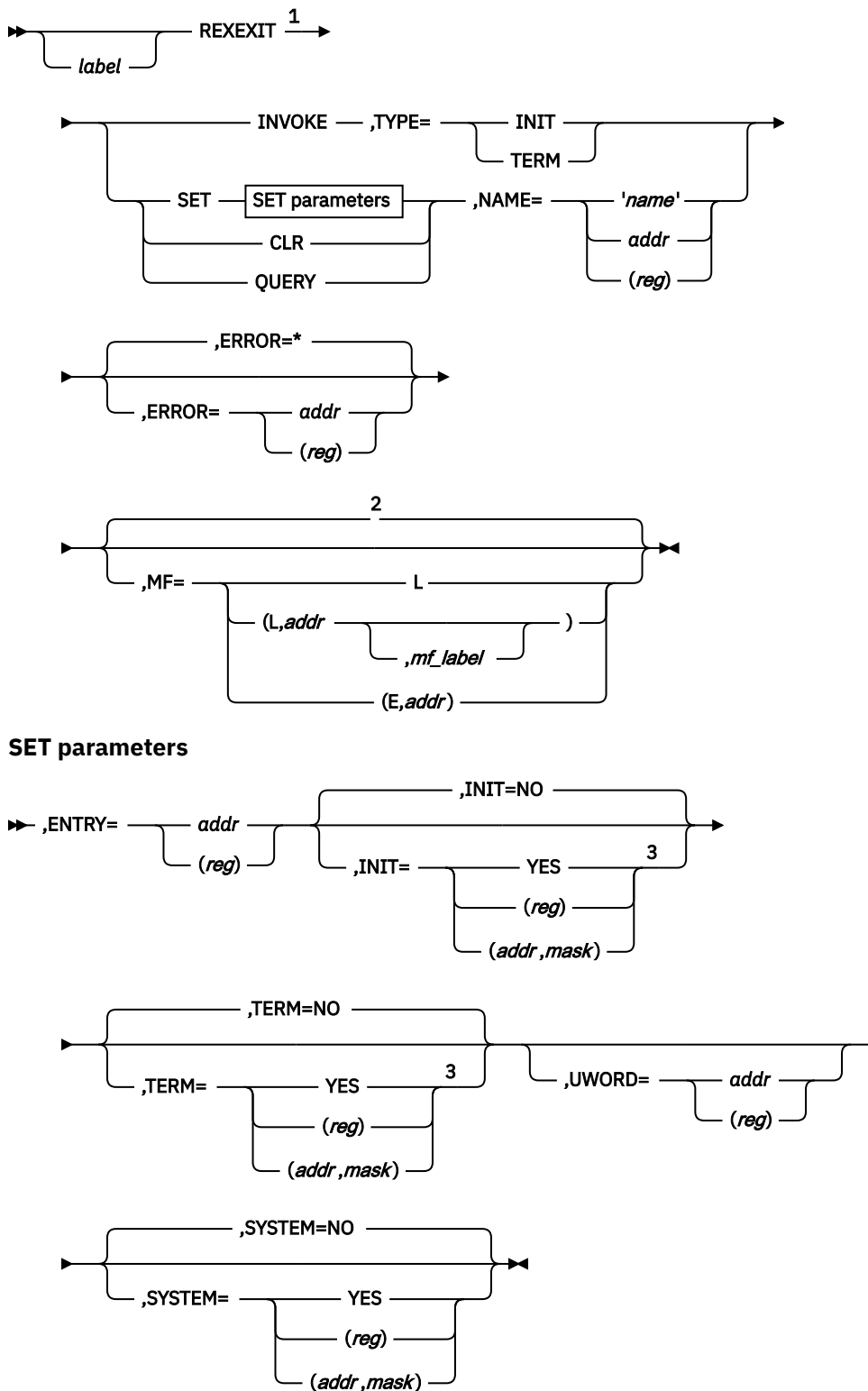
- 1. REGEQU generates these equate statements. The access register statements are produced only when AREGS=YES is specified.

Table 22. Equate statements generated by REGEQU

General Registers			Extended-control Registers			Floating-point Registers			Access Registers*		
R0	EQU	0	C0	EQU	0	F0	EQU	0	AR0	EQU	0
R1	EQU	1	C1	EQU	1	F2	EQU	2	AR1	EQU	1
R2	EQU	2	C2	EQU	2	F4	EQU	4	AR2	EQU	2
R3	EQU	3	C3	EQU	3	F6	EQU	6	AR3	EQU	3
R4	EQU	4	C4	EQU	4				AR4	EQU	4
R5	EQU	5	C5	EQU	5				AR5	EQU	5
R6	EQU	6	C6	EQU	6				AR6	EQU	6
R7	EQU	7	C7	EQU	7				AR7	EQU	7
R8	EQU	8	C8	EQU	8				AR8	EQU	8
R9	EQU	9	C9	EQU	9				AR9	EQU	9
R10	EQU	10	C10	EQU	10				AR10	EQU	10
R11	EQU	11	C11	EQU	11				AR11	EQU	11
R12	EQU	12	C12	EQU	12				AR12	EQU	12
R13	EQU	13	C13	EQU	13				AR13	EQU	13
R14	EQU	14	C14	EQU	14				AR14	EQU	14
R15	EQU	15	C15	EQU	15				AR15	EQU	15

* Generated only when AREGS=YES is specified.

REXEXIT



Notes:

- ¹ Keyword parameters can be entered in any order.
- ² Default is the standard macro format.

³ INIT or TERM must be specified as YES.

Purpose

Use the REXEXIT macroinstruction in an application program to create and maintain a list of global exit routines to be called by exec processors. Use it in an exec processor to call the exit routines before or after processing an exec (see the INVOKE parameter).

Parameters

Required Parameters:

INVOKE

is used by an alternate format exec processor to call the global exit routines defined by user programs. Use the TYPE parameter to select pre- or post-processing exit routines. See Usage Notes “1” on page 366 to “4” on page 367 and “8” on page 367 to “11” on page 367.

TYPE=

is used with the INVOKE parameter to select preprocessing or postprocessing routines.

INIT

Call preprocessing routines.

TERM

Call postprocessing routines.

SET

declares a global exit with the name and entry point name that you specify with the NAME= parameter.

CLR

deletes the named global exit from the list of exits.

QUERY

queries the named global exit.

NAME=

is the name of the exit routine to be declared, deleted, or queried.

'name'

specifies the exit routine as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage location containing the exit routine name. This is any valid assembler language expression.

(reg)

specifies the register containing the address of the storage area holding the exit routine name. Valid registers are 2-12 enclosed in parentheses.

ENTRY=

defines the entry point of the exit routine.

addr

specifies the entry point at the 8-byte storage location defined by *addr*. This is any valid assembler language expression.

(reg)

specifies the entry point at the address contained in the register. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

INIT=

indicates whether the exit routine receives control during initialization processing by an exec processor. The acceptable values are:

NO

specifies that the exit routine does not receive control during initialization processing. This is the default value.

YES

specifies that the exit routine does receive control during initialization processing.

(*reg*)

specifies the register that contains the value for INIT. The macro checks the value of the specified register and, if it is 0, sets INIT to NO. If the register contains a nonzero value, the macro sets INIT to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the INIT parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then INIT is set to NO. If the bit is 1, then INIT is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the INIT parameter as

```
INIT=(APPFLAG,X'80')
```

Note: If you do not set the value of INIT to YES, you must specify YES for the value of TERM. You can give INIT a value of YES by specifying YES, (*reg*), or (*addr,mask*).

TERM=

indicates whether the exit routine receives control during termination processing by an exec processor. The acceptable values are:

NO

specifies that the exit routine does not receive control during termination processing. This is the default value.

YES

specifies that the exit routine does receive control during termination processing.

(*reg*)

specifies the register that contains the value for TERM. The macro checks the value of the specified register and, if it is 0, sets TERM to NO. If the register contains a nonzero value, the macro sets TERM to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the TERM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then TERM is set to NO. If the bit is 1, then TERM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the TERM parameter as

```
TERM=(APPFLAG,X'80')
```

Note: If you do not set the value of TERM to YES, you must specify YES for the value of INIT. You can give TERM a value of YES by specifying YES, (*reg*), or (*addr,mask*).

UWORD=

specifies an optional fullword available to the exit routine. The userword will be passed to an invoked exit in the fourth fullword of the plist. Omitting this parameter causes a value of 0 to be passed to the exit routine.

addr

specifies the address of UWORD. This is any valid assembler language expression.

(reg)

specifies a register that contains the address of the UWORD. Valid registers are 2-12 enclosed in parentheses.

SYSTEM=

indicates whether the exit routine survives abend processing. Acceptable values are:

NO

specifies that the exit routine does not survive. This is the default value.

YES

specifies that the exit routine does survive. If you specify SYSTEM=YES, the exit must reside in storage that is not reclaimed during abend processing.

(reg)

specifies the register that contains the value for SYSTEM. The macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(addr,mask)

defines a single bit in storage that sets the value of the SYSTEM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. For more information on using the REXEXIT macroinstruction with the REXX/VM Interpreter, see [z/VM: REXX/VM Reference](#). For an alternate format exec processor, see the documentation for the exec processor.

2. The REXX/VM Interpreter calls global initialization exits before the RXINI exit, which it calls before it interprets the first instruction of the exec. Likewise, an alternate format exec processor should call the initialization exit routines before it processes the first instruction of the exec.
3. The REXX/VM Interpreter calls global termination exits after the RXTER exit, which it calls after it interprets the last instruction of the exec. Likewise, an alternate format exec processor should call the termination exit routines after it processes the last instruction of the exec.
4. The REXX/VM Interpreter makes its EXECCOMM interface available to global exit routines. Likewise, an alternate format exec processor should make its EXECCOMM interface available to global exit routines.
5. Global exit routines are invoked in the reverse order from the order in which they were SET.
6. If a global exit is SET using an existing exit name, its position in the invocation order will be the same as the existing exit. The existing exit will remain on the list. However, the most recently added version will be invoked during initialization and termination processing. Also, a subsequent CLR of this exit name clears the most recently loaded version.
7. You must provide the proper entry and exit linkage for your exit routine. When your routine receives control, the register contents are as follows:

Register	Contents
1	Address of a parameter list described as follows:
	Hex Disp. Description
	0 Exit name
	8 Exit code (See usage note “11” on page 367)
	INIT = 9
	TERM = 10
	A Exit subfunction (See usage note “11” on page 367)
	C User word
13	Address of an 18 fullword savearea
14	Return address
15	Entry point address of your exit routine

The exit routine must save registers 0-14 on entry, and restore them before returning control to the address in Register 14.

8. Exit routines will be invoked in 31-bit addressing mode. Addresses passed to the exit routine may reside above the 16 MB line, so the exit routine must be capable of addressing above the line.
9. During its execution, a global exit may use REXEXIT to SET, CLR or QUERY any global exit, including itself.
10. If an exec is invoked during the execution of a global exit, no global exits are invoked for the called exec.
11. The exit code and the exit subfunction passed in the plist correspond to the exit codes and exit subfunctions defined for REXX exits (see [z/VM: REXX/VM Reference](#)).

Return Codes

The following return codes are from a REXEXIT operation:

Code

Meaning

0

No other exits with the same name exist.

REXEXIT

- 1** Other exits with the same name exist. (This is not an error.)
- 4** INIT must have a value of YES or TERM must have a value of YES.
- 8** Unrecoverable error occurred.
- 28** Named exit not found.

RXITDEF

►► RXITDEF ◄◄

Purpose

Use the RXITDEF macroinstruction to assign the correct values to the symbols used for the exit routine function and subfunction codes. This macroinstruction may be used for CMS and GCS programs.

Usage Notes

1. For more information on using this macro, see [z/VM: REXX/VM Reference](#).
2. The following symbols are assigned by this macro:

		Function	Subfunction	Description
RXFNC	EQU	X'0002'		Process a function request.
RXFNCAL	EQU		X'0001'	FNC Call a function/subroutine.
RXCMD	EQU	X'0003'		Process a command request.
RXCMDHST	EQU		X'0001'	CMD Process a host command request.
RXMSQ	EQU	X'0004'		Manipulate the session queue.
RXMSQPLL	EQU		X'0001'	MSQ Pull an entry from queue.
RXMSQPSH	EQU		X'0002'	MSQ Push an entry onto queue.
RXMSQSZ	EQU		X'0003'	MSQ Determine the queue size.
RXSIO	EQU	X'0005'		Perform Session Input/Output.
RXSIO SAY	EQU		X'0001'	SIO Output a SAY string.
RXSIO TRC	EQU		X'0002'	SIO Output a TRACE string.
RXSIO TRD	EQU		X'0003'	SIO Terminal Read.
RXSIO DTR	EQU		X'0004'	SIO Debug Terminal Read.
RXSIO TLL	EQU		X'0005'	SIO Determine line length.
RXMEM	EQU	X'0006'		Memory management services.
RXMEMGET	EQU		X'0001'	MEM Get memory.
RXMEMRET	EQU		X'0002'	MEM Return memory.
RXHLT	EQU	X'0007'		Halt services.
RXHLTCLR	EQU		X'0001'	HLT Clear the halt status.
RXHLTTST	EQU		X'0002'	HLT Test the halt status.
RXTRC	EQU	X'0008'		Test the TRACE status.
RXTRCTST	EQU		X'0001'	TRC Test the TRACE status.
RXINI	EQU	X'0009'		Initialization service.
RXINIEXT	EQU		X'0001'	INI Initialization exit.
RXTER	EQU	X'000A'		Termination service.
RXTEREXT	EQU		X'0001'	TER Termination exit.

RXITPARM

►► RXITPARM ◄◄

Purpose

Use the RXITPARM macroinstruction to map the parameter list used to pass information between the language processor and an exit routine. This macroinstruction may be used for CMS and GCS programs.

Usage Notes

1. For more information on the macro, see [z/VM: REXX/VM Reference](#).
2. The following symbols are defined by this macro:

```

RXITPARM DSECT ,
*****
* The following parameters are common to all exit routines.
*****
RXIEXIT DS H Exit code (input)
RXISUBFN DS H Exit subfunction (input)
RXIUSER DS F User word (input)
RXICFLAG DS X Exit processing control flags
RXIFFLAG DS X Exit specific flags
RXIFEVAL EQU X'01' String returned via EVALBLOK (output)
RXIPLN DS H Length of plist in bytes (input)
DS F Reserved for IBM use
RXITMAPX DS CL24 Beginning of exit specific parameters
RXITMAPZ EQU * End of exit parameter list
RXITMAPL EQU RXITMAPZ-RXIEXIT Length of the parameter list
*****
* The following parameters are unique to the RXFNC exit.
*****
ORG RXITMAPX
RXFFERR EQU X'80' Invalid call to routine (output)
RXFFNFND EQU X'40' Routine not found (output)
RXFFSUB EQU X'20' Subroutine call (input)
RXFFNC DS A Pointer to the routine name (input)
RXFFNCL DS F Length of the routine name (input)
RXFARG DS A Pointer to argument list (input)
RXFRET DS A Pointer to EVALBLOK for
* function RETURN result (output)
RXFPLEN EQU *-RXITMAPX
*****
* The following parameters are unique to the RXCMD exit.
*****
ORG RXITMAPX
RXCFFAIL EQU X'80' Command FAILURE occurred (output)
RXCFERR EQU X'40' Command ERROR occurred (output)
RXCADDR DS CL8 Current ADDRESS setting (input)
RXCCMD DS A Pointer to the command (input)
RXCCMDL DS F Length of the command (input)
RXCRETC DS A Pointer to return code buffer (in+out)
RXCRETCL DS F Length of return code (in+out)
RXCPLEN EQU *-RXITMAPX
*****
* The following parameters are unique to the RXMSQ exit.
*****
* The following parameters are used for the RXMSQPLL function.
ORG RXITMAPX
RXMFEMPT EQU X'40' Queue was empty (output)
RXMRETC DS A Pointer to return value buffer (in+out)
RXMRETCL DS F Length of return value (in+out)
RXMPLLPL EQU *-RXITMAPX
* The following parameters are used for the RXMSQPSH function.
ORG RXITMAPX
RXMFLIFO EQU X'80' Stack the line LIFO (input)
RXMVAL DS A Pointer to line to stack (input)
RXMVALL DS F Length of line to stack (input)
RXMPSHPL EQU *-RXITMAPX

```

```

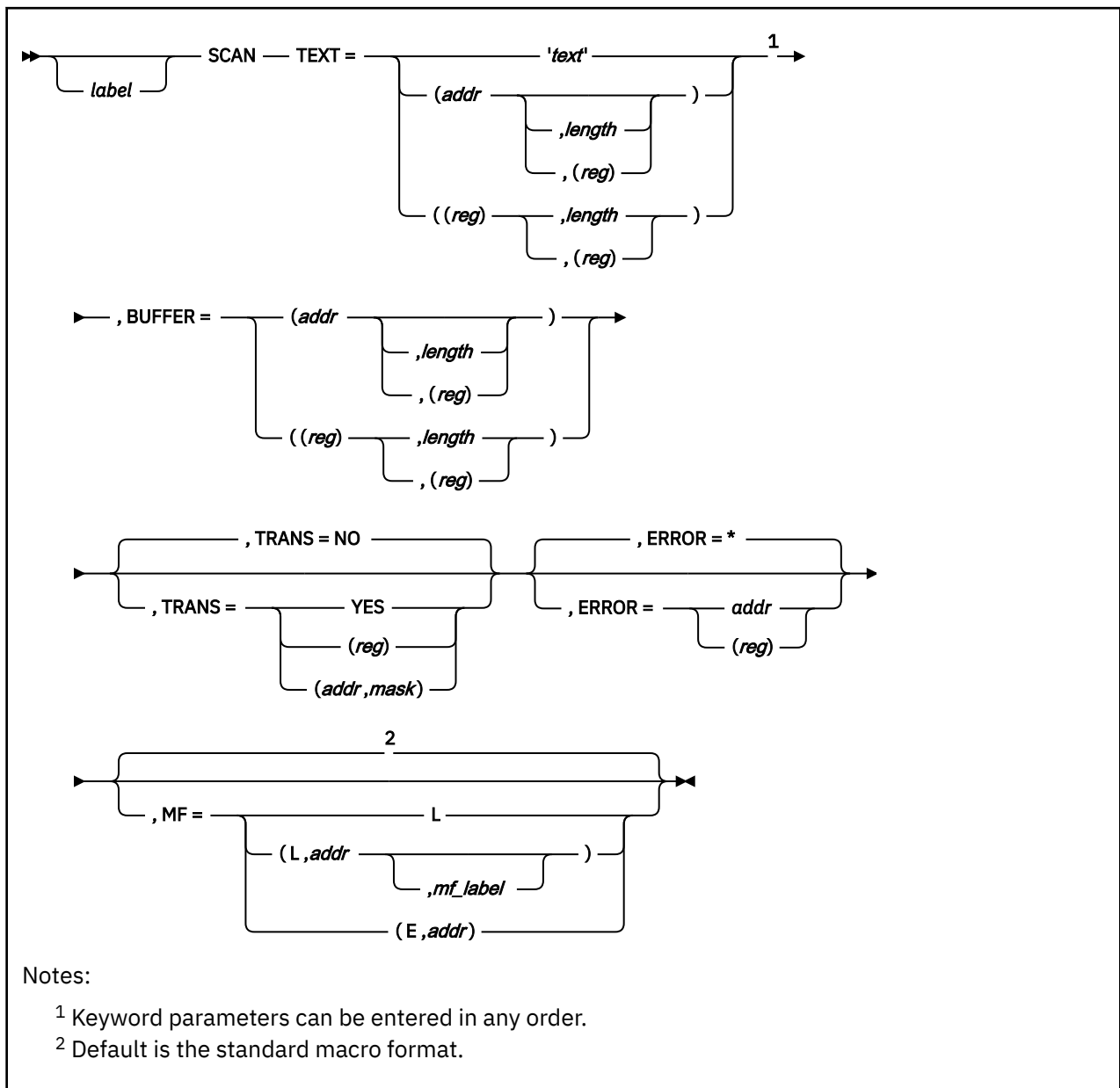
* The following parameters are used for the RXMSQSIZ function.
      ORG      RXITMAPX
RXMQSIZE DS    F          Number of lines in stack          (output)
RXMSIZPL EQU   *-RXITMAPX
*****
* The following parameters are unique to the RXSIO exit.
*****
* The following parameters are used for the RXSIOTLL function.
      ORG      RXITMAPX
RXSSIZE  DS    F          Size of terminal in bytes          (output)
RXSSIZPL EQU   *-RXITMAPX

* The following parameters are used for RXSIO SAY and RXSIOTRC.
      ORG      RXITMAPX
RXSVAL   DS    A          Address of line to display         (input)
RXSVALL  DS    F          Length of line to display          (input)
RXSOUTPL EQU   *-RXITMAPX

* The following parameters are used for RXSIOTRD and RXSIODTR.
      ORG      RXITMAPX
RXSRETC  DS    A          Pointer to return value buffer     (in+out)
RXSRETCL DS    F          Length of return value             (in+out)
RXSINPPL EQU   *-RXITMAPX
*****
* The following parameters are unique to the RXMEM exit.
*****
* The following parameters are used for RXMEMGET and RXMEMREL.
      ORG      RXITMAPX
RXMFL024 EQU   X'80'     Storage must be allocated below
*                               the 16Mb line.                 (input)
RXMSSIZE DS    F          Size of storage (in double words)
*                               to be allocated or released     (input)
RXMADDR  DS    A          Address of storage allocated        (in-out)
*                               or being released
RXMPLEN  EQU     *-RXITMAPX
*****
* The following parameters are unique to the RXHLT exit.
*****
* The following parameters are used for RXHLTST.
* (No unique parameters are required for RXHLTCLR.)
      ORG      RXITMAPX
RXHFHALT EQU   X'80'     HALT condition occurred             (output)
RXHSTR   DS    A          Pointer to EVALBLOK containing an
*                               optional HALT string             (output)
RXHPLEN  EQU     *-RXITMAPX
*****
* The following parameters are unique to the RXTRC exit.
*****
      ORG      RXITMAPX
RXTFTRAC EQU   X'80'     External TRACE setting              (output)
RXTPLEN  EQU     *-RXITMAPX
*****
* No unique parameters are used for the RXINI and RXTER exits.
*****

```

SCAN



Purpose

Use the SCAN macro to build tokenized and extended parameter lists and optionally translate an input line.

Parameters

Required Parameters:

TEXT=

is the input data to be scanned. Acceptable values are:

'text'

scans the data enclosed in the quotation marks.

(*addr,length*)

specifies the address of the data as an assembler expression and, optionally, specifies the length as an absolute expression.

(*addr, (reg)*)

specifies the address of the data as an assembler expression and, optionally, specifies the length of the data as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

(*(reg), length*)

specifies a register that contains the address of the data and specifies the length of the data as an absolute expression. If you use a register to specify the address, you must specify a length.

(*(reg), (reg)*)

specifies a register that contains the address of the data and a register that contains the length of the data. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

BUFFER=

is a user-provided buffer to contain the extended and tokenized parameter lists. You must specify a buffer large enough to contain the extended parameter list, all of the tokenized arguments, and an 8-byte fence delimiting the end of the tokenized parameter list—a minimum of 48 bytes. Acceptable values are:

(*addr,length*)

specifies the buffer address as an assembler expression and, optionally, specifies the buffer length as an absolute expression.

(*addr, (reg)*)

specifies the buffer address as an assembler expression and, optionally, the buffer length as a value contained in a register. Valid registers are 2-12 enclosed in parentheses.

(*(reg), length*)

specifies a register that contains the buffer address and specifies the buffer length as an absolute expression. If you use a register to specify the address, you must specify a length.

(*(reg), (reg)*)

specifies a register that contains the buffer address and a register that contains the buffer length. If you use a register to specify the address, you must specify a length. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:***label***

is an optional assembler label for the statement.

TRANS=

indicates whether CMS translates the input. Acceptable values are:

NO

indicates that CMS will not translate the input. No translation, including uppercase translation, is done to the input when building both the tokenized and extended parameter list. This is the default value.

YES

specifies that CMS use the user translate table (created with the CMS SET INPUT command) to translate the input data. If you have not created a user translate table, CMS uses the system uppercase translate table. If you have created a user translate table, CMS uses the system uppercase translate table with the changes specified by the SET INPUT command applied when building both the tokenized and extended parameter lists.

(*reg*)

specifies the register to be checked by the macro. If the value is 0, the macro sets TRANS to NO. If the register contains a nonzero value, the macro sets TRANS to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the TRANS parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can

specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then TRANS is set to NO. If the bit is 1, then TRANS is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the TRANS parameter as

```
TRANS=(APPFLAG,X'80')
```

To set the value of the TRANS parameter at assembly time, specify TRANS=YES or TRANS=NO. The default value is TRANS=NO. To set the value at execution time, specify TRANS=(*reg*) or TRANS=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr,mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. When the SCAN macro completes successfully, it stores the address of the tokenized parameter list in register 1 and the address of the extended parameter list in register 0.
2. The SCAN macro creates a tokenized parameter list and an extended form parameter list in the following format:

DC A(CMNDNAME)	Command name
DC A(BEGARG)	Beginning of argument list
DC A(ENDARG)	End of argument list
DC F'0'	User word
DC A(0)	Address of function argument list
DC A(0)	Address for return of function data
DC 2F'0'	Padding
DC CL8'	Tokens (as required)
DC CL8'	Tokens (as required)
DC X'FFFFFFFFFFFFFFFF'	Fence

The SCAN macro uses EPLIST to map the extended parameter list.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

4

The user-supplied buffer area is too short to contain all of the tokens in the tokenized parameter list. The list is truncated.

8

The user-supplied buffer area is less than 48-bytes.

104

CMS is unable to obtain enough storage to do the translation.

SCBLOCK



Purpose

Use the SCBLOCK macro to generate a DSECT for the SCBLOCK control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the SCBLOCK macro expansion is labeled SCBLOCK.

Usage Notes

1. The SCBPSW field cannot be used in a LOAD PSW (LPSW) instruction.
2. For more information on the SCBLOCK macro, see [“NUCEXT” on page 317](#).
3. The SCBLOCK macroinstruction expands as follows:

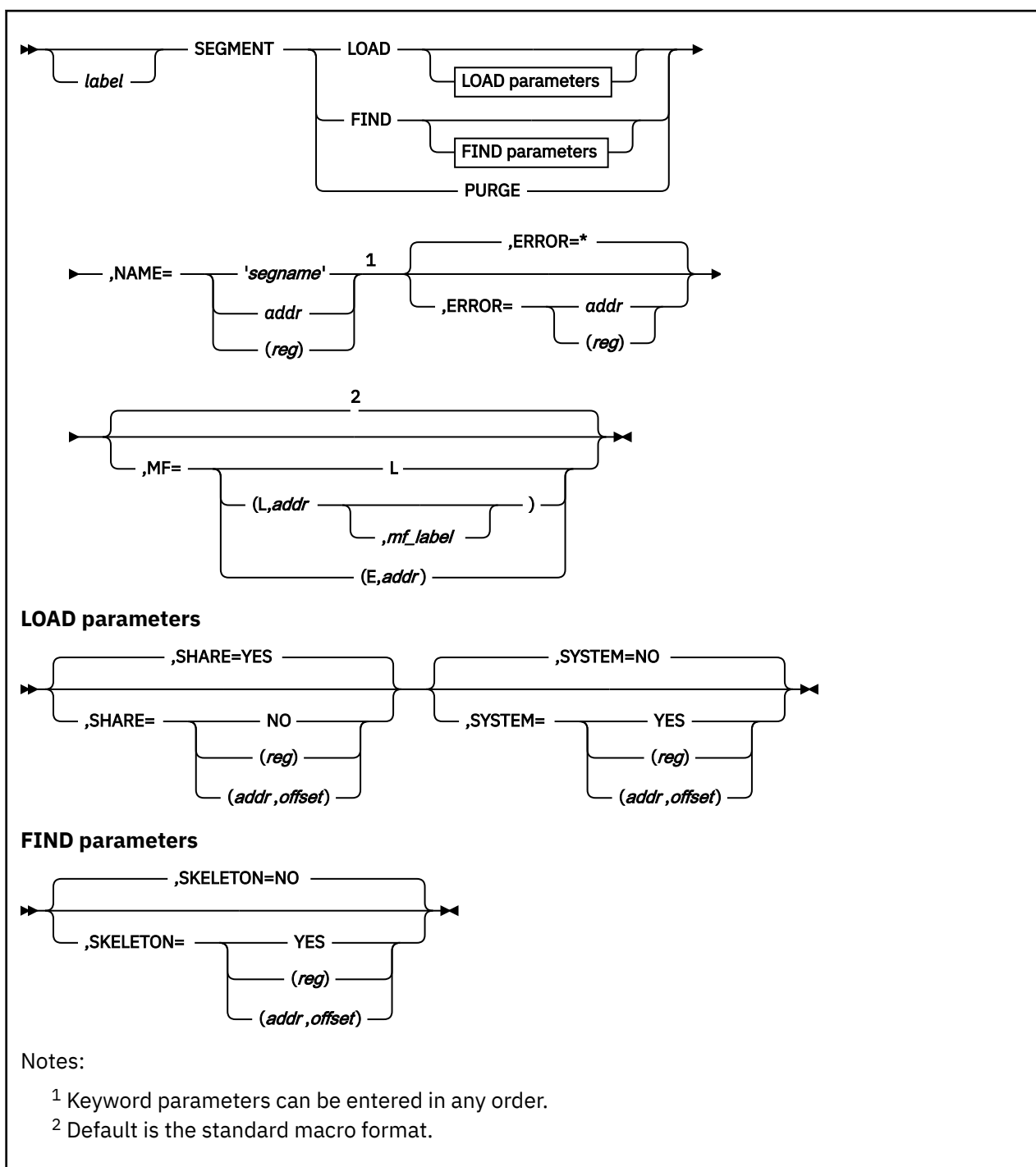
```
SCBLOCK DSECT
SCBFWPTR DS F          CHAIN POINTER TO NEXT SCBLOCK
SCBWKWRD DC A(0)        AVAILABLE FOR USER INFORMATION
SCBNAME DS CL8          NAME OF SUBCOMMAND ENVIRONMENT
SCBPSW DS D             STARTING PSW FOR SUBCOMMAND
SCBINT DS 1X            PSW INTERRUPT BITS
*
* The following table shows the combination of bits
* in SCBPSW that determine what the various INTTYPEs are.
*
* INTTYPE          |BIT 0|BIT 1|      |BIT 7|HEX value
*
* -----
* NONE              0      0      ... 0      00
* EXTERNAL           0      0      ... 1      01
* NONCONSOLE         0      1      ... 0      40
* NONCONSOLE & EXTERNAL 0      1      ... 1      41
* CONSOLE            1      0      ... 0      80
* CONSOLE & EXTERNAL  1      0      ... 1      81
* IO                  1      1      ... 0      C0
* ALL                 1      1      ... 1      C1
*
* * * * *
* NUCX FIELDS. THESE ARE PRESENT, BUT NOT USED,
* IN SCBLOCKS ON THE NUCSCBLK CHAIN.
* * * * *
SCBKEY DS 1X          PSW KEY
SCBSFLAG DS 1X        SYSTEM FLAG BYTE.
SCBSFSYS EQU X'80'     DENOTES "SYSTEM" ROUTINE --
* WILL NOT BE AUTOMATICALLY DELETED DURING ABEND
* PROCESSING.
SCBSFSER EQU X'40'     DENOTES "SERVICE" ROUTINE --
* WILL BE CALLED WITH "PURGE" ARGUMENT DURING ABEND
* PROCESSING.
SCBSFABN EQU X'20'     USED DURING ABEND
* PROCESSING.
SCBSFEND EQU X'10'     DENOTES 'END OF COMMAND'
* ROUTINE
SCBSFINT EQU SCBSFABN  USED DURING END OF COMMAND
* PROCESSING.
SCBSPERM EQU X'08'     DENOTES THAT THIS NUCLEUS
* EXTENSION WON'T BE DELETED
* DURING NUCXDROP ALL PROCESS
```

```

SCBSFIMM EQU X'04'      DENOTES THAT THIS NUCLEUS
*
*                        EXTENSION CAN ALSO BE CALLED
*                        AS AN IMMEDIATE COMMAND
SCBSFX EQU X'02'        DENOTES A LOOK-ASIDE
* ENTRY POINTING TO A REAL CMS NUCLEUS ROUTINE.
SCBSHIDE EQU X'01'      USED TO HIDE A NUCLEUS
*                        EXTENSION TEMPORARILY.
SCBUFLAG DS 1X          USER FLAG BYTE.
*
*
SCBENTR DS A            ENTRY POINT ADDRESS IN PSW
*
*
SCBXORG DS A            ADDRESS WHERE NUCLEUS
* EXTENSION WAS LOADED IN FREE STORAGE.
*
*
SCBXLEN DS F            LENGTH IN BYTES OF NUCLEUS
* EXTENSION. MAY BE ZERO FOR SECONDARY ENTRY POINTS.
*
*
SCBSFLG2 DS X           F*2 SECOND FLAG BYTE
SCBSFA31 EQU X'80'      EXTENSION IS AMODE 31
SCBSFA24 EQU X'40'      EXTENSION IS AMODE 24
* WHEN BOTH ON, EXTENSION IS AMODE ANY,
* WHEN BOTH OFF, EXTENSION IS AMODE SAME.
SCBSFSEG EQU X'20'      SEGMENT RESIDENT
SCBSFUNC EQU X'10'      Indicate function can not be
* invoked from the command line
SCBSMT EQU X'08'        mt subcom
SCBSNEWT EQU X'04'      subcom on own thread
DS 3X                   RESERVED FOR FUTURE USE
SCBSEGID DS CL8         LOGICAL SEGMENT IDENTIFIER
SCBTESTK DS F           thread excomm stack
DS 0D                   KEEP DOUBLEWORD ALIGNED
SCBLOCKB EQU *-SCBLOCK  LENGTH IN BYTES
SCBLOCKD EQU (SCBLOCKB+7)/8 LENGTH IN DWORD

```

SEGMENT



Purpose

Use the **SEGMENT** macro in an application program to load, purge or find saved segments. The **SEGMENT** macro is a macro interface to the CP **DIAGNOSE** code X'64' instruction, which supports the loading, finding and purging of saved segments. Note, you cannot use **SEGMENT PURGE** and the **DIAGNOSE** code X'64' **PURGESYS** function interchangeably.

Parameters

Required Parameters:

LOAD

indicates that the saved segment specified on the NAME parameter is to be added to the virtual machine address space.

If the saved segment is loaded successfully, general register 1 contains the address of the loaded saved segment. If the loaded saved segment is a logical saved segment, the programs contained within it are established as nucleus extensions or subcommand processors, EXECs are established as EXECs-in-storage, CSL libraries are made usable by the GLOBAL CSLLIB command, and application language information is processed. Objects in other logical saved segments within the physical saved segment are not processed.

Register 15 contains a return code indicating the results of the load operation. See [“Return Codes” on page 383](#) for specific return codes.

FIND

indicates that the starting address and highest address of the saved segment specified on the NAME parameter are to be returned.

If the saved segment was successfully located, then general register 1 contains the address of the saved segment and register 0 contains the highest address of the saved segment. Register 15 contains a return code indicating the status of the saved segment. See [“Return Codes” on page 383](#) for specific return codes.

PURGE

indicates that the saved segment specified on the NAME parameter is to be removed from the virtual machine address space. If the purged saved segment is a logical saved segment, the purge operation removes the objects contained within the saved segment from use by CMS. Nucleus extensions and EXECs are dropped, subcommand processors are cleared, language information is deleted, libraries are removed from the list of callable services libraries, and any associated minidisks are released. If no other logical saved segments within the physical saved segment are active, the physical saved segment is detached from the virtual machine. If the physical saved segment is a member of a CP segment space, and no other members of the segment space are active, the segment space is detached. The reserved storage is released (returned to CMS) if it was obtained by the SEGMENT LOAD operation.

Use SEGMENT PURGE to purge a saved segment that was loaded using SEGMENT LOAD. If the saved segment was loaded using the DIAGNOSE code X'64' LOADSYS function, you must use the DIAGNOSE code X'64' PURGESYS function to purge it. You cannot use SEGMENT PURGE and the DIAGNOSE code X'64' function interchangeably.

Register 15 contains a return code indicating the results of the purge operation. See [“Return Codes” on page 383](#) for specific return codes.

NAME=

is the name of the saved segment to load, purge or find. Acceptable values are:

'segname'

specifies the saved segment name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the saved segment name. This is specified as any valid assembler expression.

(reg)

specifies the register containing the address of the storage area holding the name. Valid registers are 2-12 enclosed in parentheses.

For a logical saved segment to be found, its definition must appear in the SYSTEM SEGID file, which is generated and updated by the SEGGEN command.

Optional Parameters:

label

is an optional assembler label for the statement.

SHARE=

indicates whether a shared or nonshared copy of the saved segment is to be loaded.

The SHARE attribute of the physical saved segment that contains logical saved segments is set by the first logical saved segment loaded in the physical saved segment. Subsequent SEGMENT LOAD operations for logical saved segments within that physical saved segment cannot change the SHARE attribute of the physical saved segment. Thus, all logical saved segments within the same physical saved segment must have the same SHARE attribute. If the SHARE operand value does not match the SHARE attribute of the physical saved segment, the saved segment will **not** be loaded and a nonzero return code will be issued. The SHARE parameter is valid only with the LOAD operation.

To set the value of the SHARE parameter at assembly time, specify SHARE=YES or SHARE=NO. To set the value at execution time, specify SHARE=(*reg*) or SHARE=(*addr,offset*). Acceptable values are:

YES

indicates that a shared copy of the saved segment is loaded. This is the default value.

NO

indicates that a nonshared copy is loaded. See Usage Note “6” on page 382.

(*reg*)

the macro checks the value of the specified register and, if it is zero, sets SHARE to NO. If the register contains a nonzero value, the macro sets SHARE to YES.

(*addr,offset*)

defines a single bit in storage that is to be used to set the value of the SHARE parameter. The *addr* is the address of a byte in storage and the *offset* determines which bit within the byte is to be tested. If the specified bit is zero, then SHARE=NO is assumed. If the bit is one, then SHARE is set to YES.

SYSTEM=

indicates whether this is a ‘system’ type loaded saved segment which survives ABEND processing.

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,offset*). The SYSTEM parameter is only valid with the LOAD operation. Acceptable values are:

YES

indicates that the loaded saved segment survives ABEND processing.

NO

indicates that the loaded saved segment does not survive ABEND processing. This is the default.

(*reg*)

indicates the macro checks the value of the specified register and, if it is zero, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(*addr,offset*)

defines a single bit in storage that is to be used to set the value of the SYSTEM parameter. The *addr* is the address of a byte in storage and the *offset* determines which bit within the byte is to be tested. If the specified bit is zero, then SYSTEM=NO is assumed. If the bit is one, then SYSTEM is set to YES.

SKELETON=

indicates whether CMS should search only for a skeleton segment (Class S NSS).

To set the value of the SKELETON parameter at assembly time, specify SKELETON=YES or SKELETON=NO. To set the value at execution time, specify SKELETON=(*reg*) or SKELETON=(*addr,offset*). The SKELETON parameter is only valid with the FIND operation. Acceptable values are:

YES

indicates that CMS should search only for a skeleton segment.

NO

indicates that CMS should first search for a logical segment and then for a segment defined in CP. An active segment (Class A or R NSS) will be found before a skeleton segment (Class S NSS). This is the default.

(reg)

indicates the macro checks the value of the specified register and, if it is zero, sets SKELETON to NO. If the register contains a nonzero value, the macro sets SKELETON to YES.

(addr,offset)

defines a single bit in storage that is to be used to set the value of the SKELETON parameter. The *addr* is the address of a byte in storage and the *offset* determines which bit within the byte is to be tested. If the specified bit is zero, then SKELETON=NO is assumed. If the bit is one, then SKELETON is set to YES.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If the specified saved segment is a physical saved segment or a CP segment space, and if a storage space for this saved segment was previously reserved (through either the SEGMENT RESERVE command or the SEGMENT LOAD command or macro) then the SYSTEM= option is ignored.
2. If a SEGMENT LOAD macro is issued for a saved segment that is already loaded, nothing is done except to set the return code. If any objects in a saved segment have been explicitly dropped (for example, using the EXECDROP or NUCXDROP commands), the saved segment must be purged, then loaded, to get those objects back.
3. When a saved segment is loaded, all language information that matches the current system language is added to the active set of applications.
4. Nucleus extensions, subcommand processors, and EXECs-in-storage that are established through the load operation override previous definitions with the same name. Saved segment resident nucleus extensions can be dropped using the NUCXDROP command, at which time the previous definition comes into effect. Similarly, saved segment resident EXECs can be dropped using the

EXECDROP command. When the saved segment is purged, previous definitions of nucleus extensions, subcommand processors, and EXECs come back into effect.

5. CMS uses the following process to locate a saved segment to be loaded:

- a. CMS searches the list of logical saved segments for one with the name specified on the SEGMENT LOAD macro. If a logical saved segment is found, a storage space for the associated physical saved segment is reserved (if not already reserved). If the physical saved segment is a DCSS, the storage space is reserved for the DCSS. If the physical saved segment is a member of a CP segment space, the storage space is reserved for the entire segment space. Then the storage space is loaded (if not already loaded), and the contents of the logical saved segment are processed.
- b. If a logical saved segment with the specified name is not found, CMS searches the list of storage spaces previously reserved with the SEGMENT RESERVE command to determine if a space has been reserved for a saved segment with the requested name. If one is found, the storage space is loaded (if not already loaded).
- c. If no reserved storage space exists, CMS issues a DIAGNOSE codeX'64' FINDSEG to determine whether the requested saved segment has been defined in CP. If the saved segment has been defined in CP, CMS issues a SEGMENT RESERVE command to create a reserved storage space, then loads the saved segment. If the saved segment is a member of a CP segment space, CMS reserves storage space for and loads the entire segment space.
- d. If the requested saved segment is none of the above, the appropriate return code (RC=44) is returned to the calling program.

This process allows an application to be loaded even if the saved segment resides within the virtual machine.

The same search order is used for the SEGMENT PURGE operation and SEGMENT FIND operation when SKELETON=NO.

For the SEGMENT FIND operation when SKELETON=YES, CMS uses only the DIAGNOSE code X'64' FINDSKEL to determine the existence and location of the skeleton segment.

6. If a nonshared copy of a saved segment is to be loaded outside the maximum storage size of the virtual machine (as defined on the USER or IDENTITY directory statement), the saved segment must be identified on a NAMESAVE control statement in the user's directory entry. If the specified saved segment is a physical saved segment that is a discontinuous saved segment (DCSS), or a logical saved segment contained in a physical saved segment that is a DCSS, the name identified on the NAMESAVE statement is the DCSS. If the specified saved segment is a physical saved segment that is a member of a CP segment space, or a logical saved segment contained in a physical saved segment that is a member of a segment space, the name identified on the NAMESAVE statement must be the segment space.
7. Loading a physical saved segment does not give you access to the logical saved segments it contains. Loading a CP segment space does not give you access to its members. Use the SEGMENT LOAD macro to load the specific saved segments you need.
8. In rare cases, a SEGMENT LOAD or SEGMENT FIND could be delayed. This could happen if another user is using the CP SPXTAPE DUMP command to dump the same saved segment name onto tape. The delay will occur only if the system data file containing the saved segment was not loadable when the dump began, or if all other system data files with the same name become not loadable during the dump. The LOAD or FIND results will depend on whether the system data file is loadable when the delay ends.

A system data file is not loadable if any of the following conditions are true:

- It is a skeleton.
- It is class P (pending purge).
- It is a member of a segment space whose system data file is a skeleton because at least one member of the space is a skeleton.
- It is a member of a segment space that is missing one or more members, resulting from either of the following:

- The CP PURGE NSS command was used without the ASSOCIATES operand to purge the members.
- Only some of the system data files were loaded from tape by the CP SPXTAPE LOAD.

Return Codes

The following return codes are from a SEGMENT LOAD operation:

Code

Meaning

0

The saved segment was successfully loaded.

1

The saved segment is defined as a VMGROUP and cannot be loaded with SEGMENT LOAD.

12

The saved segment exists and has already been loaded. It has not been reloaded.

24

Parameter specified was not valid.

28

No segment storage spaces in virtual machine.

36

The requested saved segment is of a different level (segment space, physical saved segment, logical saved segment, or skeleton segment) than what is already loaded into the same virtual storage area. For example, if physical saved segment PSEG1 contains logical saved segment LSEG1, and LSEG1 is already loaded, then a SEGMENT LOAD of PSEG1 will give this return code.

This return code also indicates that the SHARE parameter is not valid, as would be the case if the physical saved segment containing the specified logical saved segment is already loaded with the opposite SHARE attribute.

41

The storage required to load the saved segment is already in use. Either the required virtual machine storage has already been allocated by CMS or storage outside the virtual machine has been reserved for another saved segment.

44

The saved segment does not exist.

53

The user who issued SEGMENT is not in the CP directory.

60

The saved segment location does not match the reserved storage location. This can occur if a DEFSEG command has been issued which moved the location of a saved segment between the time storage was reserved for the saved segment and the time the SEGMENT LOAD was issued.

104

There is insufficient storage to allocate the SEGMENT function work area.

174

Paging I/O errors occurred while attempting to load the saved segment.

203

Not used

256

An error occurred while processing the contents of a logical saved segment.

449

The user is not authorized to load a nonshared copy of the saved segment.

475

A fatal I/O error occurred while reading the CP directory.

SEGMENT

1352

CP has detected an unacceptable condition; this is most likely to occur in the event of a CP soft abend.

1357

Not used.

1358

Load of a CP DCSS attempted.

1367

The user attempted to load a member saved segment in a mode different from the segment space that contains it.

The following are the return codes from a SEGMENT PURGE operation:

Code

Meaning

0

The saved segment was successfully purged.

24

Parameter specified was not valid

40

The saved segment is not currently loaded by CMS.

44

The saved segment does not exist or was unloaded by a previous DIAGNOSE code X'64' operation.

104

There is insufficient storage to allocate the SEGMENT function work area.

256

An error occurred while processing the contents of a logical saved segment.

1352

CP has detected an unacceptable condition; this is most likely to occur in the event of a CP soft abend.

The following are the return codes from a SEGMENT FIND operation:

Code

Meaning

0

The saved segment exists and is not currently loaded.

1

The saved segment is defined as a VMGROUP and cannot be processed with DIAGNOSE code X'64'.

12

The saved segment exists and has already been loaded. This is not an error condition.

24

Parameter specified was not valid

41

The physical saved segment could not be loaded to determine the address of the logical saved segment.

44

The saved segment does not exist.

104

There is insufficient storage to allocate the SEGMENT function work area.

174

Paging I/O errors occurred while attempting to find the saved segment.

203

Not used.

449

This saved segment has restricted access and the user is not authorized to use it.

1352

CP has detected an unacceptable condition; this is most likely to occur in the event of a CP softabend.

SGMTEXTIT



Purpose

Use the SGMTEXTIT macroinstruction to generate a DSECT or a CSECT for the SGMTEXTIT control block. This control block is used by the programs specified in the USER Record used by the SEGGEN command. For more information on the SEGGEN command, see [z/VM: CMS Commands and Utilities Reference](#).

Parameters

Optional Parameters:

CSECT

specifies that a CSECT is to be generated rather than a DSECT.

DSECT

is the default.

Usage Notes

1. For more information on the SGMTEXTIT macro, see [z/VM: Saved Segments Planning and Administration](#).
2. The SGMTEXTIT macro expands as follows:

	SGMTEXTIT	
SGMTEXTIT	DSECT	Queue Manager plist
SGMRTN	DS CL8' '	Name of user routine
SGMNAME	DS CL8' '	Name of user object
SGMLSEG	DS CL8' '	Name of lseg user info is in
SGMFUNC	DS XL2'00'	Function code
SGMLDS	EQU 0	Load in shared mode
SGMLDNS	EQU 4	Load in nonshared mode
SGMPURGE	EQU 8	Purge
	DS XL2'00'	Reserved field
SGMSTART	DS F'0'	Start of user info
SGMEND	DS F'0'	End of user info
SGMPSTRT	DS F'0'	Start address of parm list
SGMPLEN	DS F'0'	Length of parm list
	DS CL12' '	Reserved field
SGMBLKBY	EQU *-SGMTEXTIT SGMTEXTIT length in bytes	
SGMBLKDW	EQU (SGMBLKBY+7)/8 SGMTEXTIT length in dwords	

SHVBLOCK



Purpose

Use the SHVBLOCK macroinstruction to generate a DSECT for the SHVBLOCK control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the SHVBLOCK macroinstruction expansion is labeled SHVBLOCK.

Usage Notes

1. For more information on the SHVBLOCK macroinstruction, see [z/VM: REXX/VM Reference](#).
2. The SHVBLOCK macroinstruction expands as follows:

```

SHVBLOCK
*
*      ***  LAYOUT OF SHARED-VARIABLE ACCESS CONTROL BLOCK  ***
*
*  THE CONTROL BLOCKS FOR ACCESSING SHARED VARIABLES ARE CHAINED
*  AS A LIST TERMINATED BY A NULL POINTER.  THE LIST IS ADDRESSED
*  VIA THE 'PRIVATE INTERFACE' PLIST IN A SUBCOMMAND CALL TO A
*  PUBLIC VARIABLE-SHARING ENVIRONMENT (E.G. AS SET UP BY THE
*  EXEC 2 OR REXX/VM).
*
SHVBLOCK DSECT ,
SHVNEXT DS    A      (+0)  CHAIN POINTER (0 IF LAST)
SHVUSER DS    A      (+4)  NOT USED, AVAILABLE FOR PRIVATE
*                          use EXCEPT DURING 'FETCH NEXT'
SHVCODE DS    CL1    (+8)  INDIVIDUAL FUNCTION CODE
SHVRET  DS    XL1    (+9)  INDIVIDUAL RETURN CODE FLAG
*                          RESERVED, SHOULD BE ZERO
SHVBUFL DS    F      (+12) LENGTH OF 'FETCH' VALUE BUFFER
SHVNAMA DS    A      (+16) ADDR OF PUBLIC VARIABLE NAME
SHVNAML DS    F      (+20) LENGTH OF PUBLIC VARIABLE NAME
SHVVALA DS    A      (+24) ADDR OF VALUE BUFFER (0 IF NONE)
SHVVALL DS    F      (+28) LENGTH OF VALUE (SET BY 'FETCH')
SHVBLEN EQU   *-SHVBLOCK  (LENGTH OF THIS BLOCK = 32)
*
*  FUNCTION CODES (SHVCODE):
*
SHVFETCH EQU   C'F'          COPY VALUE OF SHARED VAR TO BUFFER
SHVSTORE EQU   C'S'          STORE GIVEN VALUE IN SHARED VARIABLE
*  The following function codes only apply to the System
*  Product Interpreter.
*
*  (Note that the symbolic name codes are lowercase)
SHVDROPV EQU   C'D'          DROP VARIABLE
SHVSYPFET EQU   X'86'        =C'f'  SYMBOLIC NAME FETCH VARIABLE
SHVSYSET EQU   X'A2'        =C's'  SYMBOLIC NAME SET VARIABLE
SHVSYDRO EQU   X'84'        =C'd'  SYMBOLIC NAME DROP VARIABLE
SHVNEXTV EQU   C'N'          FETCH 'NEXT' VARIABLE
SHVPRIV EQU    C'P'          FETCH PRIVATE INFORMATION
*
*  RETURN CODE FLAGS (SHVRET):
*
SHVCLEAN EQU   X'00'          EXECUTION WAS OK
SHVNEWV EQU    X'01'          VARIABLE DID NOT EXIST
*                          (SP interpreter only)
SHVLVAR EQU    X'02'          LAST VARIABLE TRANSFERRED (FOR N)

```

SHVBLOCK

SHVTRUNC	EQU	X'04'	TRUNCATION OCCURRED FOR 'FETCH'
SHVBADN	EQU	X'08'	INVALID VARIABLE NAME
SHVBADV	EQU	X'10'	INVALID VARIABLE VALUE, e.g. too
*			long (EXEC 2 ONLY).
SHVBADF	EQU	X'80'	INVALID FUNCTION CODE (SHVCODE)

SUBCOM

Purpose

Use the SUBCOM macroinstruction to access the SUBCOM function. The SUBCOM macroinstruction provides all the functions available with the SUBCOM function; it also lets you specify the addressing mode of the subcommand processor entry point.

The four basic functions of the SUBCOM macroinstruction are:

SUBCOM ANCHOR

Obtain the anchor pointer for the chain of SCBLOCKs that describe the current list of subcommand processors.

SUBCOM CLR

Delete a subcommand processor from the chain of SCBLOCKs that describe the current list of subcommand processors.

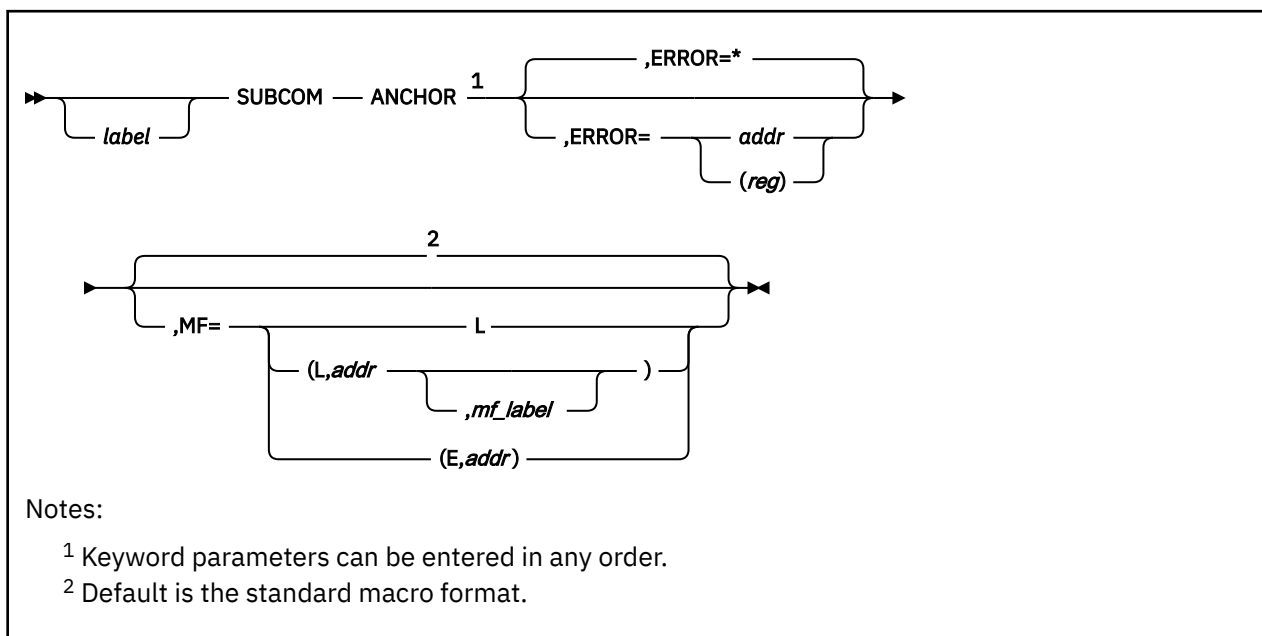
SUBCOM QUERY

Determine if a subcommand processor is defined.

SUBCOM SET

Declare a subcommand processor.

SUBCOM ANCHOR



Purpose

Use SUBCOM ANCHOR to obtain the anchor pointer for the chain of SCBLOCKs that describe the current list of subcommand processors.

Parameters

Required Parameters:

ANCHOR

returns in register 1 the pointer to the first entry in the SUBCOM chain of SCBLOCKs.

Note: The ANCHOR option requires a read and write parameter list; therefore, use the execute form (MF=(E,addr)) of the macro if you require reentrant code.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

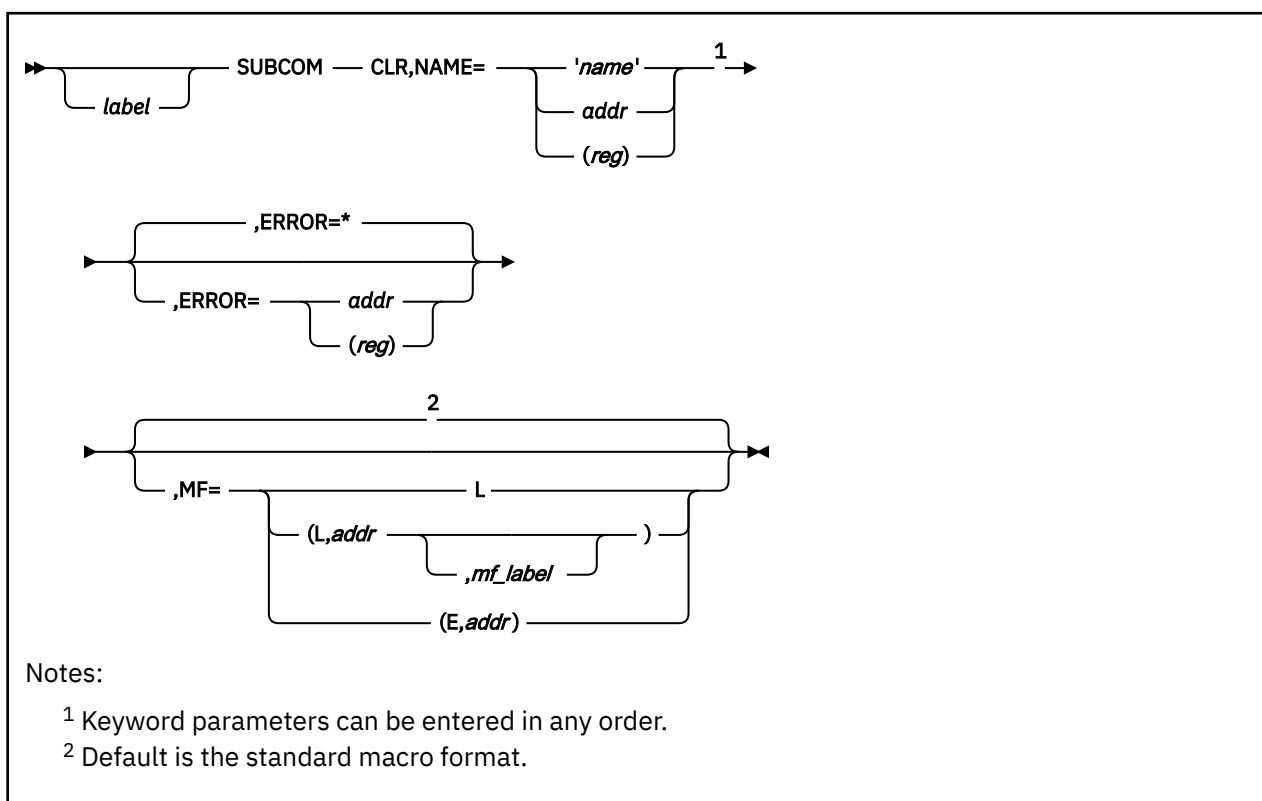
(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

SUBCOM CLR



Purpose

Use SUBCOM CLR to delete a subcommand processor from the chain of SCBLOCKs that describe the current list of subcommand processors.

Parameters

Required Parameters:

CLR

deletes the named subcommand processor from the list of subcommand processors.

NAME=

names the subcommand processor to be cleared. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the name. *addr* is any valid assembler expression.

(reg)

specifies a register that contains the address of the storage area holding the name. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

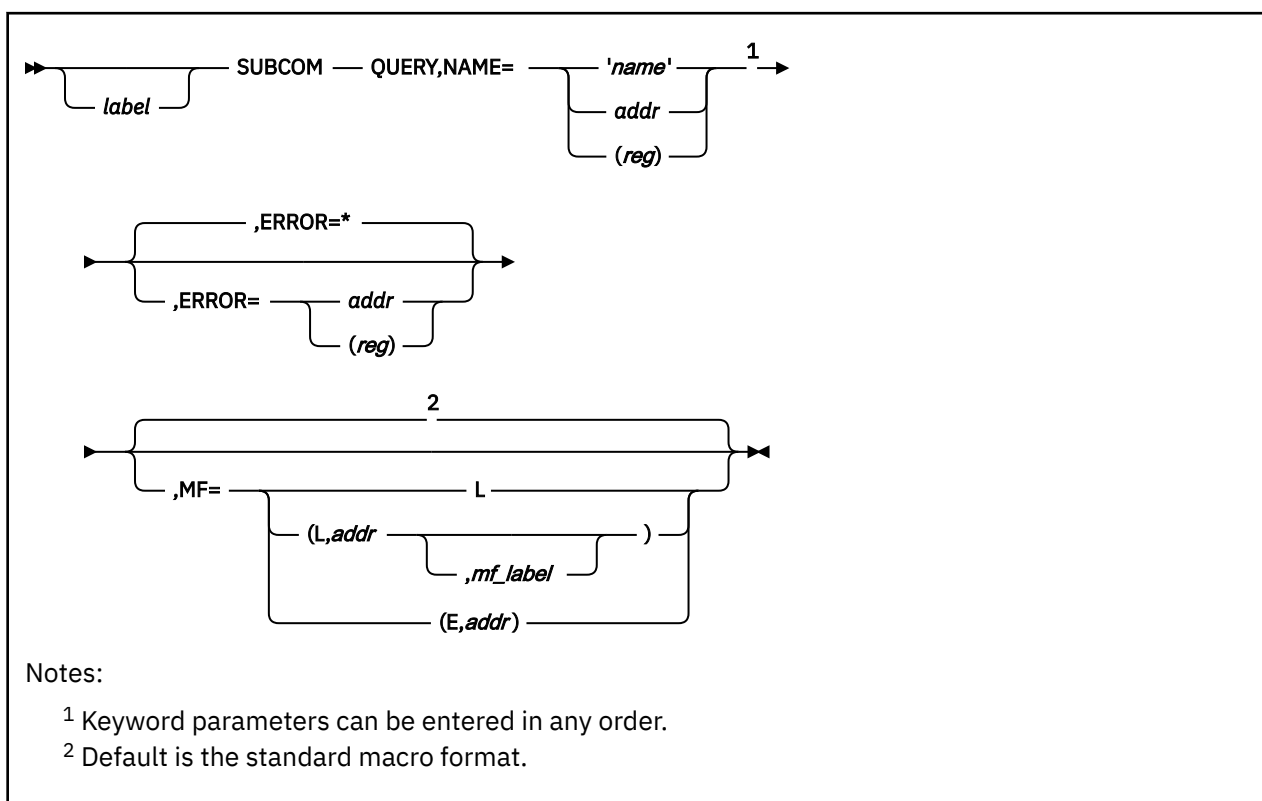
If an error occurs, register 15 contains the following return code:

Code**Meaning**

1

No SCBLOCK exists for the specified program or routine. This is the return code for a delete or a query.

SUBCOM QUERY



Purpose

Use SUBCOM QUERY to determine if a subcommand processor exists.

Parameters

Required Parameters:

QUERY

returns in register 1 the pointer to the SCBLOCK that describes the named subcommand processor.

Note: The QUERY option requires a read and write parameter list; therefore, use the execute form (MF=(E,addr)) of SUBCOM if you require reentrant code.

NAME=

names the subcommand processor to be queried. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the name. This is any valid assembler expression.

(reg)

specifies a register that contains the address of the storage area holding the name. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr*,*mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr*,*mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Return Codes

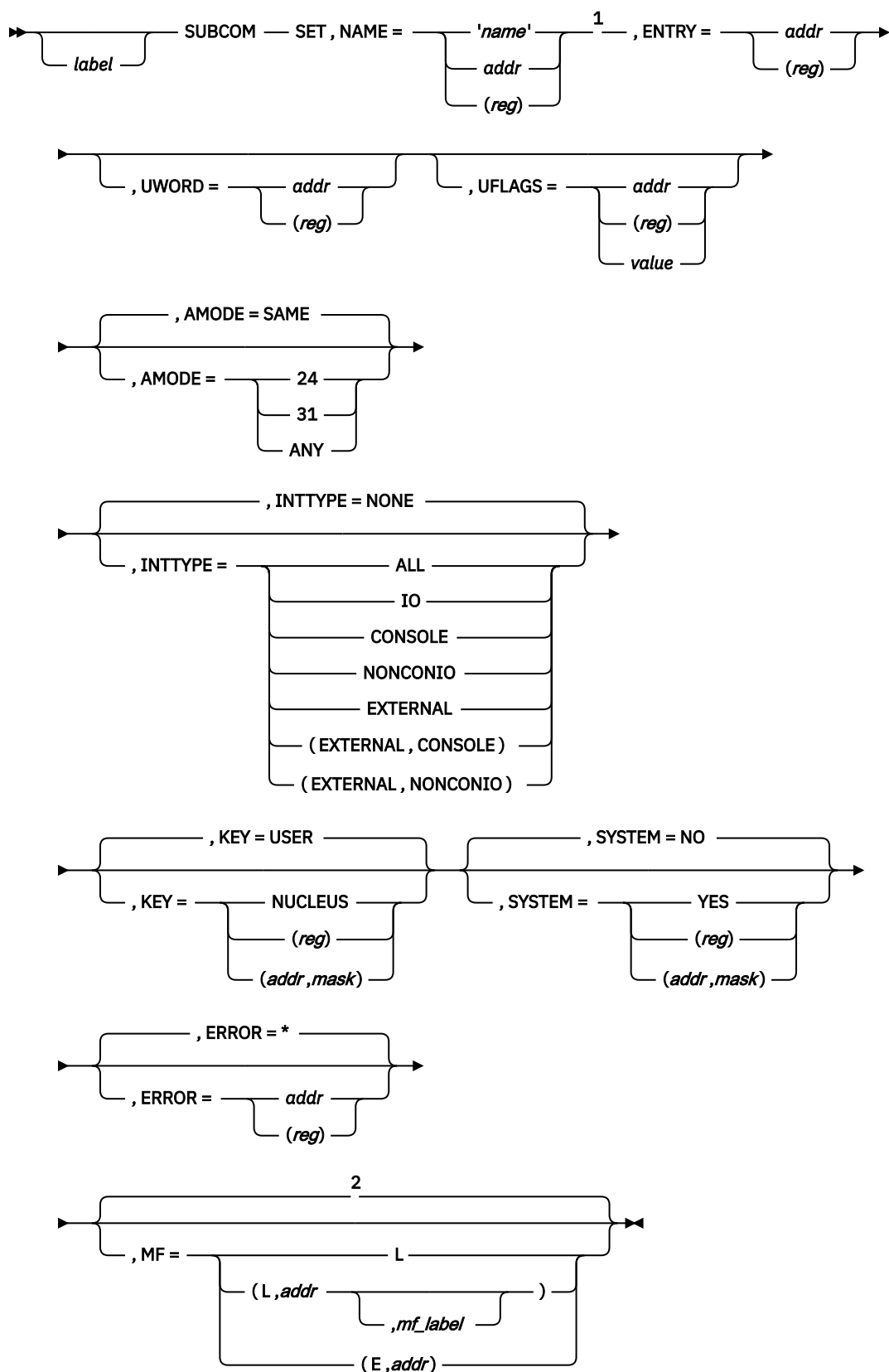
If an error occurs, register 15 contains the following return code:

Code**Meaning**

1

No SCBLOCK exists for the specified program or routine. This is the return code for a delete or a query.

SUBCOM SET



Notes:

¹ Keyword parameters can be entered in any order.

² Default is the standard macro format.

Purpose

Use the SUBCOM SET macro to declare a subcommand processor.

Parameters

Required Parameters:

SET

declares the named entry point as a subcommand processor.

NAME=

names the subcommand processor to be defined. Acceptable values are:

'name'

specifies the name as a 1- to 8-character literal string enclosed in single quotation marks.

addr

specifies the address of the 8-byte storage area containing the name. This is any valid assembler expression.

(reg)

specifies a register that contains the address of the storage area holding the name. Valid registers are 2-12 enclosed in parentheses.

ENTRY=

defines the entry point of the subcommand processor. Acceptable values are:

addr

specifies the entry point at the 8-byte storage location defined by *addr*. This is any valid assembler expression.

(reg)

specifies the entry point at the address contained in the register. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

UWORD=

specifies an optional fullword available in the SCBWKWRD field of the SCBLOCK, which register 2 points to when the subcommand processor is invoked. Acceptable values are:

addr

defines *addr* as the UWORD. This is any valid assembler expression.

(reg)

defines the contents of *(reg)* as the UWORD. Valid registers are 2-12 enclosed in parentheses.

UFLAGS=

specifies an optional 1-byte field available in the SCBUFLAG field of the SCBLOCK, which register 2 points to when the subcommand processor is invoked. Acceptable values are:

addr

defines UFLAGS as the contents of the 1-byte field pointed to by *addr*. *addr* is any valid assembler expression.

(reg)

defines UFLAGS as the contents of low-order byte of *(reg)*. Valid registers are 2-12 enclosed in parentheses.

value

defines UFLAGS as a self-defining 1-byte constant (such as X'01' or C'F').

AMODE=

specifies the addressing mode in which the subcommand processor is entered. Acceptable values are:

SAME

enters the subcommand processor in the same addressing mode as the program that issues the SUBCOM macro. This is the default.

24

enters the subcommand processor in 24-bit addressing mode.

31

enters the subcommand processor in 31-bit addressing mode.

ANY

enters the program in the same addressing mode as the calling routine.

INTTYPE=

specifies the PSW interrupt mask the CMS SVC interrupt handler is to use when invoking the subcommand processor. Acceptable values are:

NONE

disables all interrupts. This is the default value.

ALL

enables all interrupts.

IO

enables all I/O interrupts.

CONSOLE

enables only I/O interrupts from the virtual machine console. The interrupt subclass (ISC) for the console is enabled.

NONCONIO

enables only nonconsole I/O interrupts. All ISCs except for the console ISC are enabled.

EXTERNAL

enables external interrupts.

(EXTERNAL, CONSOLE)

enables external interrupts and I/O interrupts from the virtual machine console. The interrupt subclass (ISC) for the console is enabled.

(EXTERNAL, NONCONIO)

enable for external interrupts and nonconsole I/O interrupts. All ISCs except for the console ISC are enabled.

See [“ENABLE” on page 178](#) for more information on the INTTYPE parameter.

KEY=

specifies the storage key in which the routine executes (either NUCLEUS or USER key). Acceptable values are:

USER

specifies the storage key as USER key. This is the default value.

NUCLEUS

specifies the storage key as NUCLEUS key.

(reg)

the macro checks the value of the specified register and, if it is 0, sets KEY to USER. If the register contains a nonzero value, the macro sets KEY to NUCLEUS.

(addr,mask)

defines a single bit in storage that sets the value of the KEY parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0,

then KEY is set to USER. If the bit is 1, then KEY is set to NUCLEUS. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the KEY parameter as

```
KEY=(APPFLAG,X'80')
```

To set the value of the KEY parameter at assembly time, specify KEY=NUCLEUS or KEY=USER. The default is KEY=USER. To set the value at execution time, specify KEY=(*reg*) or KEY=(*addr,mask*).

SYSTEM=

indicates whether the subcommand processor survives abend processing. Acceptable values are:

NO

specifies the subcommand processor does not survive abend processing. This is the default value.

YES

specifies the subcommand processor does survive abend processing. If you specify SYSTEM=YES, the subcommand processor must reside in storage that is not reclaimed during abend processing.

(*reg*)

the macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the SYSTEM parameter. The *addr* is the address of a byte in storage and the *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. The default value is SYSTEM=NO. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

*

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,*addr,mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,*addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. When a subcommand environment is created by a multitasking application, it becomes associated with the process that created it, while also being known throughout the session. If a thread in another process invokes the subcommand processor, a thread is created in the process that established it to run the SUBCOM invocation. In this way, the subcommand processor runs in the language environment of its process and if it abends, VMERROR event handlers established in that process can attempt recovery. See [z/VM: CMS Application Multitasking](#) for more information.
2. An entry point that is to be a subcommand processor must not be the multitasking initialization routine VMSTART. While a subcommand processor can perform multitasking operations, it cannot be the starting point for a new process. See [z/VM: CMS Application Multitasking](#) for more information.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

20

The name specified on the SUBCOM macro contains an invalid character. The following characters are not valid: =, *, (,) and X'FF'.

25

No more free storage available. SCBLOCK cannot be created for the specified program or routine.

SUBPOOL

Purpose

Use the SUBPOOL macro to manage CMS free storage subpools. SUBPOOL has three functions:

SUBPOOL CREATE

Creates a storage subpool.

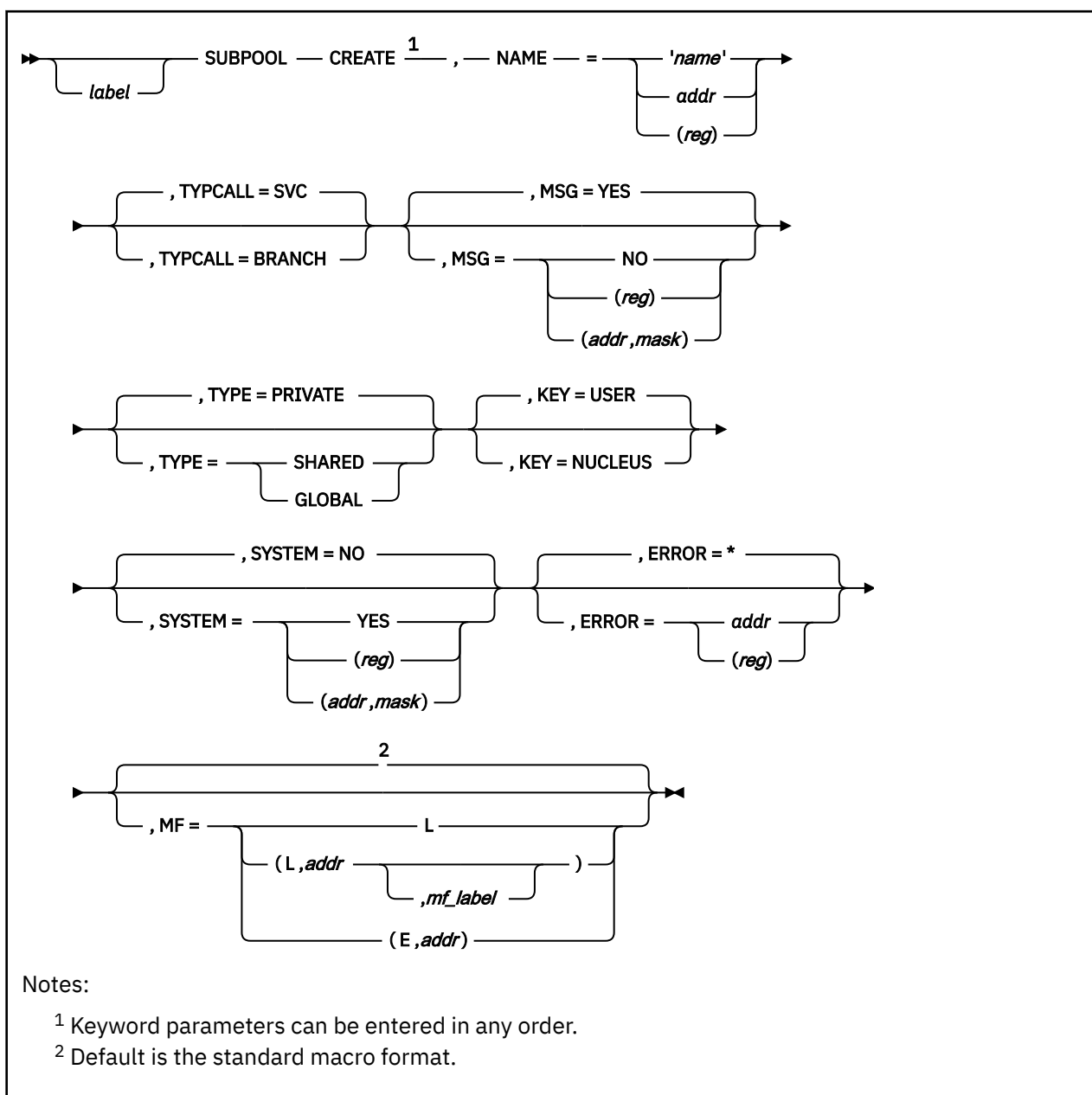
SUBPOOL DELETE

Deletes free storage subpools from the list of active subpools and returns any free storage associated with the subpool

SUBPOOL RELEASE

Releases the storage associated with a subpool but does not delete the subpool from the list of active subpools.

SUBPOOL CREATE



Purpose

Use the SUBPOOL CREATE macro to create subpools.

Parameters

Required Parameters:

CREATE

creates a free storage subpool.

NAME=

indicates the 1- to 8-character name of the subpool to be created. Acceptable values are:

'name'

specifies a 1- to 8-character literal string in single quotation marks.

addr

specifies the address of an 8-byte storage area containing the name. This may be any assembler expression.

(reg)

specifies a register that contains the address of the storage area holding the name. Valid registers are 2-12 enclosed in parentheses.

There is no restriction on the characters you can use for subpool names. However, the subpool names DMSxxxxx are reserved for OS/MVS storage subpools, and the names USER, USERG, and NUCLEUS are for reserved system subpools.

Optional Parameters:

label

is an optional assembler label for the statement.

TYPCALL=

indicates how control is passed to the CMS subpool management routines. Nucleus resident routines can use TYPCALL=BRANCH to branch directly to the subpool management routine. Routines that aren't nucleus resident must use TYPCALL=SVC.

SVC

indicates that the calling routine is not nucleus resident. This is the default value.

BRANCH

branches directly to the subpool management routine.

Note: Routines that specify TYPCALL=BRANCH must use the proper storage key and disable interrupts to call SUBPOOL.

MSG=

indicates whether CMS displays an error message if it cannot allocate sufficient storage to satisfy the request. Acceptable values are:

YES

specifies that messages are to be displayed. This is the default value.

NO

specifies that messages are not to be displayed.

(reg)

the macro checks the value of the specified register and, if it is 0, sets MSG to NO. If the register contains a nonzero value, the macro sets MSG to YES.

(addr,mask)

defines a single bit in storage that sets the value of the MSG parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then MSG is set to NO. If the bit is 1, then MSG is set to YES. For example, to test the first bit in the single byte of storage at location MSGFLAG, specify the MSG parameter as

```
MSG=(MSGFLAG,X'80')
```

To set the value of the MSG parameter at assembly time, specify MSG=YES or MSG=NO. To set the value at execution time, specify MSG=(reg) or MSG=(addr,mask).

TYPE=

indicates the accessibility and scope of the subpool. Acceptable values are:

PRIVATE

the subpool is available only to the routine that creates it. This is the default value.

SHARED

the subpool is available to any routine of a lower nested SVC level than the routine that created it.

GLOBAL

the subpool is available to any routine that runs in the virtual machine. CMS does not delete a global subpool when the program that creates it terminates. To retain global subpools across CMS abend processing, specify the SYSTEM=YES parameter.

KEY=

indicates whether the subpool is allocated from X'E' USER key storage (KEY=USER) or X'F' NUCLEUS key storage (KEY=NUCLEUS). Acceptable values are:

USER

specifies the storage key as USER key. This is the default value.

NUCLEUS

specifies the storage key as NUCLEUS key.

SYSTEM=

indicates whether a GLOBAL subpool survives abend processing.

The SYSTEM=YES parameter is valid only for global subpools (TYPE=GLOBAL). Acceptable values are:

NO

specifies that the subpool does not survive abend processing. This is the default value.

YES

specifies that the subpool survives abend processing.

(*reg*)

the macro checks the value of the specified register and, if it is 0, sets SYSTEM to NO. If the register contains a nonzero value, the macro sets SYSTEM to YES.

(*addr,mask*)

defines a single bit in storage that sets the value of the SYSTEM parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then SYSTEM is set to NO. If the bit is 1, then SYSTEM is set to YES. For example, to test the first bit in the single byte of storage at location APPFLAG, specify the SYSTEM parameter as

```
SYSTEM=(APPFLAG,X'80')
```

To set the value of the SYSTEM parameter at assembly time, specify SYSTEM=YES or SYSTEM=NO. SYSTEM=NO is the default. To set the value at execution time, specify SYSTEM=(*reg*) or SYSTEM=(*addr,mask*).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

*

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(*reg*)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,*addr,mf_label*)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. You can assign the same name to more than one PRIVATE or SHARED subpool, but not to GLOBAL subpools. When you create a subpool, CMS "pushes" it onto a LIFO stack and uses it for any subsequent storage requests that specify the name of the SUBPOOL. This continues until the subpool is deleted and gets "popped" off the stack.

The exception to this rule occurs when a PRIVATE subpool exists on the current SVC level, and a program running on that SVC level creates a SHARED or GLOBAL subpool with the same name. CMS always obtains storage from a PRIVATE subpool before a SHARED or GLOBAL subpool with the same name. The same thing happens when a SHARED subpool exists and you create a GLOBAL subpool with the same name—CMS obtains storage from the SHARED subpool.

2. Specifying KEY=NUCLEUS does not affect the cleanup action taken upon a named subpool. CMS cleans up named subpools in NUCLEUS key and USER key in the same way. Unless a subpool is TYPE=GLOBAL, CMS deletes it when the program that creates the subpool terminates. To survive CMS abend recovery, the named subpool must be TYPE=GLOBAL and the SYSTEM=YES parameter must be specified.

Return Codes

Code

Meaning

0

Normal completion.

1

Not enough storage is available to create a subpool descriptor for the specified subpool.

2

An attempt was made to CREATE a subpool with the name of USER, USERG, or NUCLEUS, or a name reserved for a CMS internal subpool.

3

An attempt was made to CREATE a GLOBAL subpool and a GLOBAL subpool with the specified name already exists.

6

A SUBPOOL CREATE for either a PRIVATE or SHARED subpool was requested and there is no existing SSAVE to anchor the subpool on. This could happen from an interrupt handler.

9

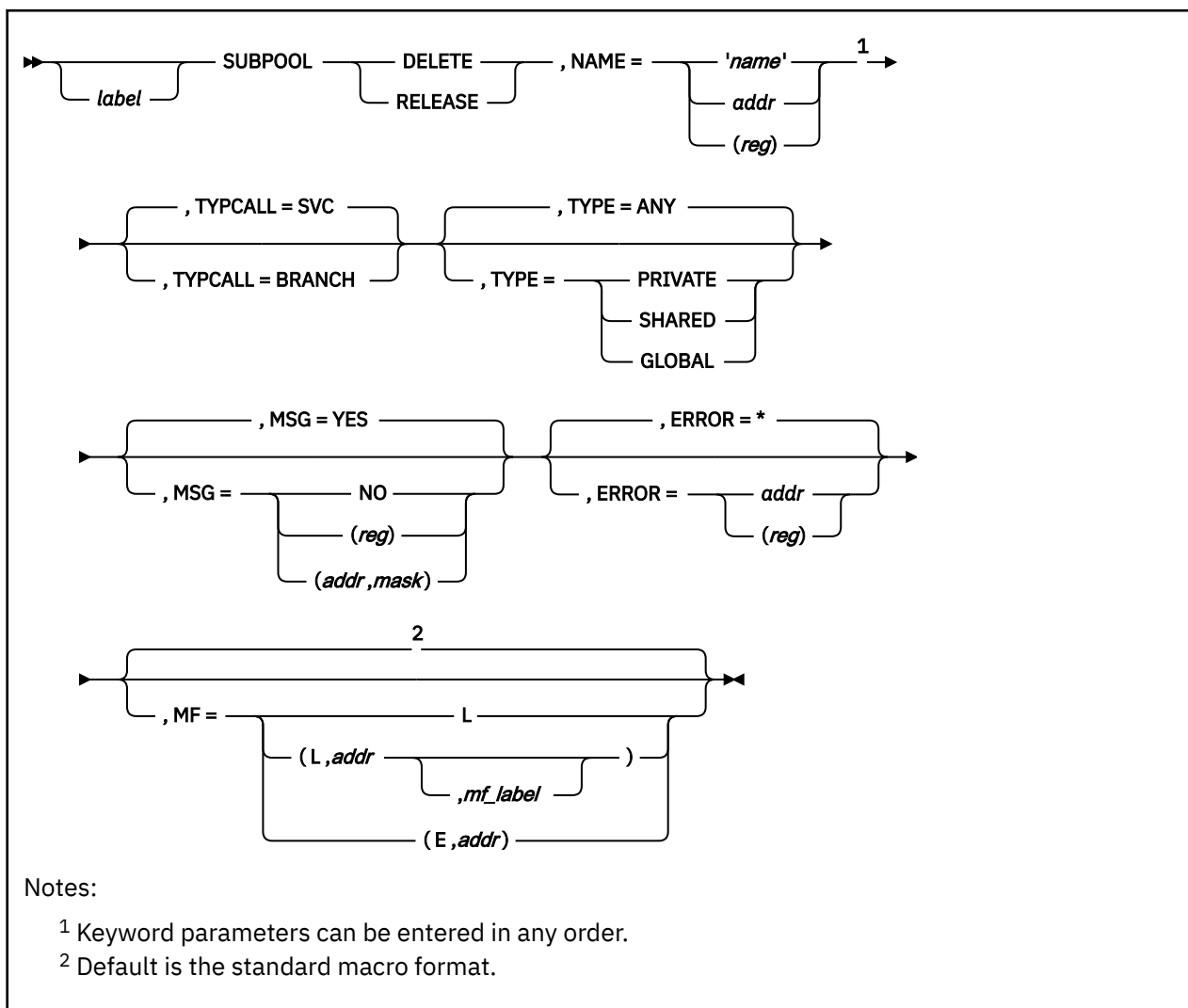
Unexpected and unexplained error in the storage management routine.

10

An invalid parameter list was detected. This happens when you use a combination of macro forms to build a parameter list and you (a) omit parameters or (b) specify conflicting parameters. This often occurs when you re-use a parameter list without first zeroing it out. The error is caused by one of the following:

- The NAME= parameter was not specified.
- SYSTEM=YES was specified for a SHARED or PRIVATE subpool.
- TYPE=ANY was specified (this can happen with an MF=(E,addr) invocation where TYPE=ANY was specified on a previous form of the macro, such as MF=(L,addr).

SUBPOOL DELETE and RELEASE



Purpose

Use the SUBPOOL DELETE and RELEASE macroinstructions to delete a free storage subpool or to release storage allocated to a free storage subpool.

Parameters

Required Parameters:

DELETE

deletes a free storage subpool from the active subpool list and returns its storage to the pool of unallocated storage.

RELEASE

releases all storage allocated to a free storage subpool and returns the storage to the pool of unallocated storage. SUBPOOL RELEASE does not remove the subpool name from the list of available subpools.

NAME=

indicates the 1- to 8-character name of the subpool being managed. Acceptable values are:

'name'

deletes the named (1- to 8-characters) subpool or releases its storage.

addr

deletes the subpool named at the specified 8-byte area or releases its storage. The variable *addr* may be any assembler expression.

(reg)

deletes the subpool named at the address contained in the specified register or releases the subpool's storage. Valid registers are 2-12 enclosed in parentheses.

Optional Parameters:

label

is an optional assembler label for the statement.

TYPCALL=

indicates how control is passed to the CMS subpool management routines. Nucleus resident routines can use TYPCALL=BRANCH to branch directly to the subpool management routine. Routines that are not nucleus resident must use TYPCALL=SVC. Acceptable values are:

SVC

indicates that the calling routine is not nucleus resident. This is the default.

BRANCH

branches directly to the subpool management routine.

Note: Routines that specify TYPCALL=BRANCH must use the proper storage key and disable interrupts to call SUBPOOL.

TYPE=

indicates the scope of the subpool to be deleted or released. Acceptable values are:

ANY

searches the subpool chains (first PRIVATE, then SHARED, then GLOBAL) for the named subpool. If the named subpool is found, it is deleted or its storage is released. This is the default value.

PRIVATE

deletes or releases the subpool only if it is on the private subpool chain.

SHARED

deletes or releases the subpool only if it is on the shared subpool chain.

GLOBAL

deletes or releases the subpool only if it is on the global subpool chain.

MSG=

indicates whether an error message is displayed if CMS cannot delete the subpool or release its storage. Acceptable values are:

YES

specifies that messages are to be displayed. This is the default value.

NO

specifies that messages are not to be displayed.

(reg)

the macro checks the value of the specified register and, if it is 0, sets MSG to NO. If the register contains a nonzero value, the macro sets MSG to YES.

(addr,mask)

defines a single bit in storage that sets the value of the MSG parameter. The variable *addr* is the address of a byte in storage and the variable *mask* determines which bit within the byte the macro tests. You can specify *addr* and *mask* in any form allowed on a TM assembler instruction. If the specified bit is 0, then MSG is set to NO. If the bit is 1, then MSG is set to YES. For example, to test the first bit in the single byte of storage at location MSGFLAG, specify the MSG parameter as

```
MSG=(MSGFLAG,X'80')
```

To set the value of the MSG parameter at assembly time, specify MSG=YES or MSG=NO. To set the value at execution time, specify MSG=(reg) or MSG=(addr,mask).

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. When a subpool is deleted, it is "popped" off a LIFO stack. If another subpool with the same name exists, CMS uses it to satisfy subsequent requests for free storage from a subpool with that name.
2. If the subpool to be deleted or released is not found within the scope of the TYPE parameter, an error is returned.
3. See [Table 13 on page 104](#) for information about when subpools are automatically deleted or released.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code

Meaning

2

An attempt was made to DELETE or RELEASE a subpool with the name of USER, USERG, or NUCLEUS, or a name reserved for a CMS internal subpool.

4

The specified subpool was not found.

9

Unexpected and unexplained error in the storage management routine.

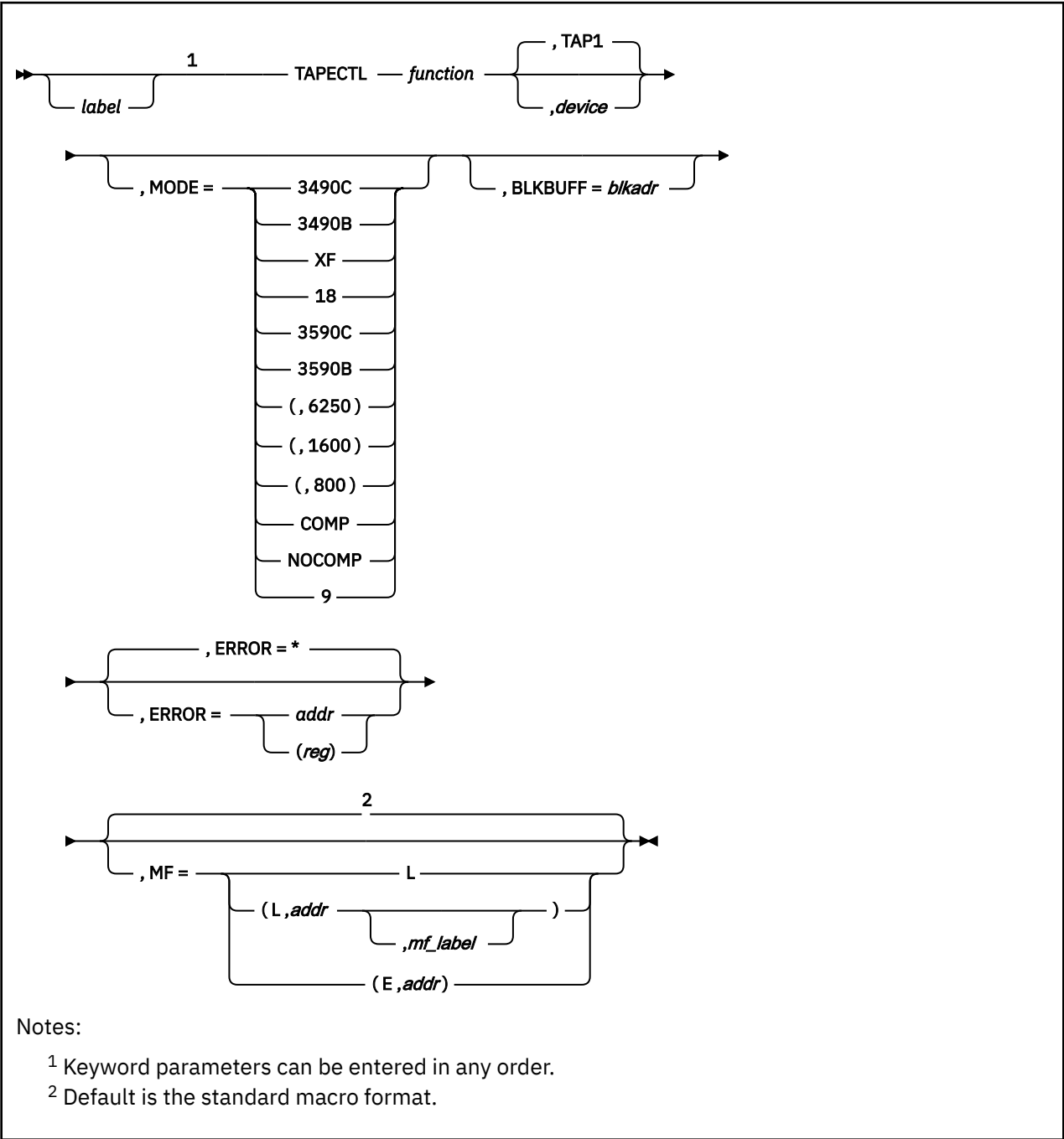
10

An invalid parameter list was detected. This happens when you use a combination of macro forms to build a parameter list and you (a) omit parameters or (b) specify conflicting parameters. This often occurs when you re-use a parameter list without first zeroing it out. The error is caused by one of the following:

- The NAME= parameter was not specified.

- SYSTEM=YES was specified.

TAPECTL



Purpose

Use the TAPECTL macroinstruction to position the specified tape according to the specified function code.

Parameters

Required Parameters:

function
specifies the control function to be performed. The functions are as follows:

REW

Rewind the tape

RUN

Rewind and unload the tape

ERG

Erase a defective section of the tape

BSR

Backspace one record

BSF

Backspace one file

FSR

Forward-space one record

FSF

Forward-space one file

WTM

Write a tape mark

LOCBLK

Locate block

RDBLKID

Read block ID

See Usage Note “1” on page 413 for descriptions of the LOCBLK and RDBLKID functions.

Optional Parameters:

label

is an optional assembler label for the statement.

device

specifies the device name (TAP*n*) or virtual device number (*vdev*) of the virtual tape device from which the block is to be read. The following values are valid; see *z/VM: CMS User's Guide* for information on tape device names and virtual device numbers for tape devices.

Device Name	Virtual Number	Device Name	Virtual Number
TAP0	0180	TAP8	0288
TAP1	0181	TAP9	0289
TAP2	0182	TAPA	028A
TAP3	0183	TAPB	028B
TAP4	0184	TAPC	028C
TAP5	0185	TAPD	028D
TAP6	0186	TAPE	028E
TAP7	0187	TAPF	028F

If you omit the *device* value, CMS uses device TAP1.

(reg)

a register containing a pointer to a storage location that contains the device name or virtual device number.

The following example shows how you might use the register form to identify the device:

```

LA      2, MY181      Addr of device assignment
TAPECTL REW, (2)
      .
      .

```

MY181	DC	CL4'0181'	vdev definition
-------	----	-----------	-----------------

MODE=

indicates a recording format. This is meaningful only if *function* is WTM or ERG. Regardless of whether CMS is attempting to write on the tape, the macro expansion will fail if the tape device is not capable of writing the indicated recording format.

Note that this parameter has no effect if the tape device is not positioned to the beginning of the volume and the recording format it specifies is not allowed to coexist on a volume with the recording format that is recorded at the beginning of the volume. See [z/VM: CMS User's Guide](#) for details on selecting recording formats.

Values are:

3490C

3490 Compacted recording format

3490B

3490 Basic recording format

XF

3480 Compacted recording format

18

3480 Basic recording format

3590C

3590 Compacted recording format

3590B

3590 Basic recording format

(,6250)

GCR recording format

(,1600)

PE recording format

(,800)

NRZI recording format

COMP

Any compacted recording format

NOCOMP

Any uncompact recording format

9

Any 9 track recording format

For compatibility with previous levels of VM, the following values are also valid, but the above values are preferred.

(,38K)

3480 Basic recording format

(18,38K)

3480 Basic recording format

(9,6250)

GCR recording format

(9,1600)

PE recording format

(9,800)

NRZI recording format

Note: The different syntax of this parameter is for compatibility purposes.

If you omit the MODE parameter, CMS selects a recording format for you. CMS's choice may be the result of a previous TAPE command, so you may use the TAPECTL macro in conjunction with the TAPE command to select tape recording formats. See *z/VM: CMS User's Guide* and the description of the TAPE command in *z/VM: CMS Commands and Utilities Reference* for details and examples.

BLKBUFF=

BLKBUFF is used with the LOCBLK or RDBLKID functions only.

- For LOCBLK, the BLKBUFF parameter gives the address of a 4-byte block ID.
- For RDBLKID, the BLKBUFF parameter gives the address of an 8-byte buffer in which CMS returns 2 block IDs.

See Usage Note “1” on page 413 for details on block IDs and using the LOCBLK and RDBLKID functions.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see “CMS Macro Formats” on page 15. Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The LOCBLK and RDBLKID functions are useful only with a tape which is recorded in a recording format that contains block IDs. These recording formats are:

- 3490 Compacted
- 3490 Basic
- 3480 Compacted
- 3480 Basic
- 3590 Compacted
- 3590 Basic.

If you use the LOCBLK or RDBLKID functions with another recording format, the TAPECTL macro call fails with return code 3.

The recording formats associate a 4-byte block ID with every block and tape mark on the tape. The RDBLKID function returns the block ID of a block or tape mark, while the LOCBLK positions the tape

to a particular block based on its block ID. LOCBLK is typically much faster than other means of block positioning.

RDBLKID returns two block IDs in the 8-byte buffer you provide:

- a. The first block ID (bytes 0-3) is called the channel block ID. It identifies the block or tape mark that you would read or write if you issued a read or write operation from the current tape position.
- b. The second block ID (bytes 4-7) is called the device block ID. It identifies the block or tape mark that will next be transferred to or from the physical tape (this would be different from the channel block ID if there are blocks in the device's buffer). This block ID is of limited value.
 - i) If the device has been reading, then it may have read ahead on the physical tape and the device block ID refers to the next block to be read from the tape into the buffer.
 - ii) If the device has been writing, there may be blocks still in the buffer that have not been transferred to the tape and the device block ID refers to the next block to be transferred from the buffer to the tape.

See the manuals for the recording format or device you are using for details on block IDs.

Note:

- a. The first block or tape mark on a volume has block IDX'01000000'.
- b. Some of the complexity in the block IDs that exists in other systems is eliminated in CMS because CMS does not do read backward operations.

The LOCBLK function causes the device to position to the indicated block or tape mark. The block ID contains physical location information, so the device is able to get to the vicinity of the block or tape mark at high speed (faster than the FSF function, for example). It then locates the exact block or tape mark at normal speed. It positions the volume so that the next block or tape mark you read or write will have the block ID you specified.

The block with the block ID you specify doesn't actually have to exist, but the block right before it must exist. The device figures out the block ID of the block *before* the one you indicated and positions just *after* it. This is so you can position to a block that doesn't exist yet—one that you are about to create.

LOCBLK works regardless of where the tape is positioned when the operation starts. It may be positioned before or after the block in question, or even in an unreadable section of the tape.

You use LOCBLK and RDBLKID together. You use a RDBLKID to get the channel block ID for a particular position on the tape and save it. Later, when you want to get to that particular position on that tape, you use LOCBLK with the block ID you saved.

2. The 9346 tape cartridge can only be written to at load point and at logical end of tape.
3. If the tape is under the control of a Tape Library Dataserver machine, and the DFSMS/VM Removable Media Services (RMS) FSMPPSI CSLLIB is available to CMS, the RUN function calls the RMS FSMRMDMT (Demount) CSL routine to have the Dataserver unmount the tape.

Return Codes

The return codes (found in register 15) from a TAPECTL call are listed below. Detailed explanations of them follow.

Code

Meaning

0

The TAPECTL call executed normally.

1

Invalid function or parameter or the specified device is incapable of writing in the specified format.

2

Tape mark or End of Volume area detected.

- 3 I/O error.
- 4 Invalid *device* value.
- 5 Virtual tape device not attached (device does not exist).
- 6 Volume is write protected.
- 7 Specified device is not a tape device.
- 9 Manual rewind/unload of tape.

Return Code 0: TAPECTL call executed normally. The requested operation has been performed. For particular functions, the specific meaning is as follows.

Function	Meaning of return code 0
BSF	The volume is positioned just before the first tape mark before the place where the volume was positioned.
BSR	The device has backed over one data block and the volume is positioned just before it.
ERG	A gap has been written (except on devices which do not have (or need) an <i>erase gap</i> function). The volume is not positioned to the End of Volume area. If it were, you would get return code 2 instead.
FSF	The volume is positioned just after the first tape mark after the place where the volume was positioned.
FSR	The device has spaced forward over one data block and the volume is positioned just after it.
LOCBLK	The volume is positioned to read or write the requested block with the next read or write operation. It is also possible to get this return code when the block does not logically exist on the volume.
RDBLKID	The block ID corresponding to the present position of the volume is in your buffer.
REW	The volume is positioned to Logical Beginning of Volume.
RUN	The volume has been unloaded.
WTM	A tape mark has been written. The volume is positioned just after it, and not in the End of Volume area. If it were, you would get return code 2 instead.

Return Code 1: Invalid parameter or bad format. One of the following is true of the TAPECTL call:

- One of the parameter values is not valid.
- The parameter values are not compatible with each other.
- The MODE parameter indicates a recording format which the device (identified by the *device* parameter) is not capable of writing. Note that the TAPECTL call will fail if this is true, even though some of the TAPECTL functions do not involve writing on the tape. This is because of compatibility purposes.

For the invalid parameter cases, the TAPECTL invocation needs to be corrected. For the recording format incapable case, it is usually best to eliminate the MODE parameter if the function is not WTM or ERG. If the function is WTM or ERG, you can either specify a different recording format, specify a different device, or attach a different device. It may be better to specify a nonspecific recording format (like MODE=COMP or omit MODE completely), so that CMS chooses a recording format that the device can write. See [z/VM: CMS User's Guide](#) for information on recording formats and device capabilities.

Return Code 2: Tape mark or End of Volume. This return code is possible only for the BSR, FSR, WTM, or ERG functions.

For the BSR or FSR functions, this return code means a tape mark, rather than a data block, was spaced over. In all other respects, it is the same as return code 0.

A tape mark often marks the end of a file or the end of recorded data on the volume.

For the WTM or ERG functions, this return code is the same as return code 0, except that the volume is positioned in the End of Volume area after having performed the operation.

Return Code 3: I/O error. The device was unable to perform the function for many reasons. Following is a list of some of the general reasons, followed by a list specific to individual functions:

- The device or channel has detected an internal malfunction in the device or channel.
- There is a defect on the recording medium.
- The data on the tape was written in error.
- The tape reel or cartridge is damaged.
- The device is in a Volume Fenced condition. This is a condition which arises for reasons in which the device will not perform most operations on the volume. You can undo this condition by unloading the device; other times by rewinding the device. You can do either of these with the TAPE command or TAPECTL macro.
- The virtual device is a *shareable* one (see [z/VM: CMS User's Guide](#)). CMS does not support shareable devices and the failure of TAPECTL in this way is just one of the possible effects.

Function	Meaning of Return Code 3
BSF	<ul style="list-style-type: none"> • There is no tape mark between where the volume was positioned and Logical Beginning of Volume. The volume is now positioned to Logical Beginning of Volume. • The tape or a block on it is recorded in a recording format which the device is incapable of reading, or does not recognize.
BSR	<ul style="list-style-type: none"> • The volume was already positioned to Logical Beginning of Volume. It is still positioned there. • The tape or a block on it is recorded in a recording format which the device is incapable of reading, or does not recognize.
ERG	<ul style="list-style-type: none"> • There is no room left on the volume for a gap. • The tape cartridge contains a length of tape that the device cannot properly handle, so the device is preventing you from writing on it. See remarks under WTM.
FSF	<ul style="list-style-type: none"> • The volume was positioned past the end of recorded data or there is no tape mark between where it was and the end of recorded data. See other remarks under FSR. • The tape or a block on it is recorded in a recording format which the device is incapable of reading, or does not recognize.
FSR	<ul style="list-style-type: none"> • The volume was positioned past the end of recorded data. The end of recorded data is defined as the point just after the block, tape mark, or gap that was most recently written on the tape. Note that you will not necessarily get this return code when this is the case. On newer devices which place a definitive End of Data mark on the volume, return code 3 is guaranteed. But on older devices there are several other return codes you could get, including 0, so you must use other means to know where the end of recorded data is. • The tape or a block on it is recorded in a recording format which the device is incapable of reading, or does not recognize.
LOCBLK	<ul style="list-style-type: none"> • The block before the requested block does not exist on the tape. Note that blocks after the end of recorded data are considered not to exist. You are not guaranteed to get this return code when the block does not exist. You may get other return codes, including return code 0. The position of the volume is undefined. It is valid for the requested block not to exist, as long as the one before it exists because this lets you position the volume to <i>write</i> that block. • The mounted volume is recorded in a recording format that does not have block IDs.
RDBLKID	<ul style="list-style-type: none"> • The mounted volume is recorded in a recording format that does not have block IDs.
REW	No special meaning.
RUN	No special meaning.

Function	Meaning of Return Code 3
WTM	<ul style="list-style-type: none"> • There is no room left on the volume for the tape mark. • The tape cartridge contains a length of tape that the device cannot properly handle, so the device is preventing you from writing on it. The position of the volume has not changed. Another device may be able to write the block. Otherwise, you can copy the data on the cartridge to a cartridge which the device can handle. Note that the same device may allow you to read the tape even though it will not write it. You take a risk of damaging the device or the medium by reading it, but the device allows it to give you a chance of recovering your data.

Return Code 4: Invalid device value. The value of the *device* parameter is not one of the valid ones listed for it. CMS cannot tell on what device to perform the operation. The TAPECTL invocation must be corrected.

Return Code 5: Device not attached. No virtual device exists with the virtual device number given by the *device* parameter or, if *device* specifies a device name, with the device number CMS associates with that name.

You must either specify a different device name or number or create one with the proper virtual device number. The *z/VM: CMS User's Guide* explains this.

Return Code 6: Volume is write protected. This return code is only possible with the ERG and WTM functions.

The volume was mounted in read only. If it is a cartridge, its write protect switch is activated. If it is a reel, it doesn't have a write enable ring present.

Nothing has been written on the volume and its position has not changed.

To write on the volume, the real volume must be unloaded and an operator must write enable the volume before loading it again.

Return Code 7: Device not a tape device. The device which has the device number given by the *device* parameter or, if *device* specifies a device name, with the device number CMS associates with that name, is not a tape device.

You must either specify a different device name or number or detach the attached device and create a tape device instead with that virtual device number.

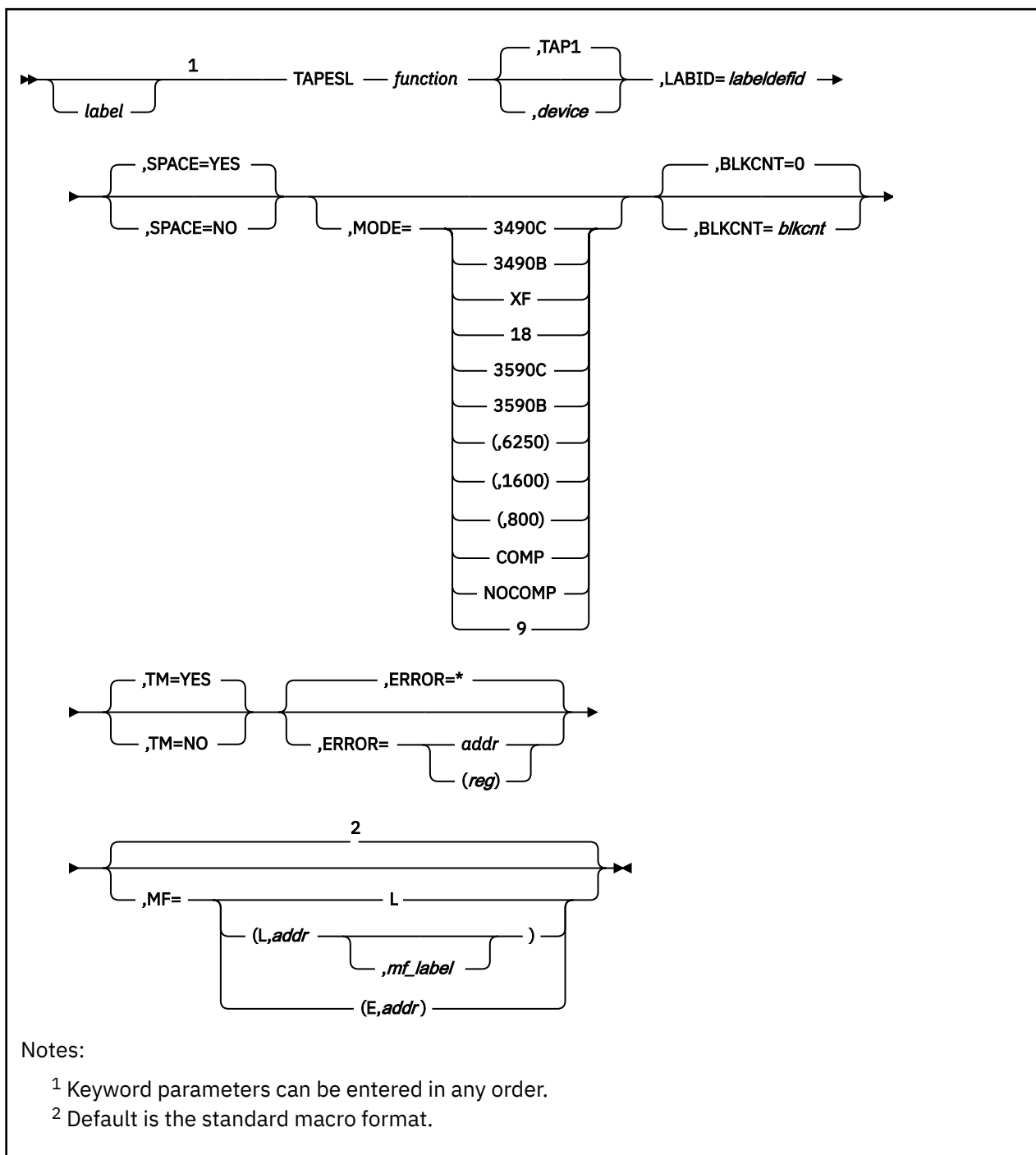
Return Code 9: Manual rewind/unload. Someone has previously rewound or unloaded the volume on the real device associated with the virtual device by operating manual controls on the physical device. In order to warn you of this, CMS has returned this return code to TAPECTL without attempting to perform the requested function. The position of the volume has not been changed.

You get this warning once, so if you want to perform the operation, you can just repeat the TAPECTL call.

CMS gives you this warning because the volume upon which you intended to operate may not be mounted now.

With older devices, you do not get this warning.

TAPESL



Purpose

The TAPESL macroinstruction processes IBM standard HDR1 and EOF1 labels without using DOS or OS/MVS OPEN and CLOSE macroinstructions. It is used with RDTAPE, WRTAPE, and TAPECTL. TAPESL processes only HDR1 and EOF1 labels. It does not process other labels such as standard user labels or HDR2 labels. It does not perform any functions of opening a tape file other than label checking or writing. The same macroinstruction is used both to check and to write tape labels.

You must issue a LABELDEF command separately to use TAPESL. Position the tape correctly (at the label to be checked or at the place where the label is to be written) before you issue the macro. TAPECTL may be used to position the tape. TAPESL reads or writes only one tape block unless SPACE=YES is specified.

Parameters

Required Parameters:

function

is one of the following:

HIN

checks input HDR1 label.

HOUT

writes HDR1 label.

EIN

checks input EOF1 label.

EOUT

writes output EOF1 label.

EVOUT

writes output EOVI label.

LABID=*labeldefid*

identifies the LABELDEF command that is to be used with this label operation. This is a 1-to 8-character name that you specify in the applicable LABELDEF command. (You must issue a LABELDEF command specifying this *labeldefid* before the TAPESL macro expansion executes).

Optional Parameters:

label

is an optional assembler label for the statement.

device

specifies the device name (TAP*n*) or virtual device number (*vdev*) of the virtual tape device from which the label is to be read or to which the label is to be written. The following values are valid; the see [z/VM: CMS User's Guide](#) for information on tape device names and virtual device numbers for tape devices.

Device Name	Virtual Number	Device Name	Virtual Number
TAP0	0180	TAP8	0288
TAP1	0181	TAP9	0289
TAP2	0182	TAPA	028A
TAP3	0183	TAPB	028B
TAP4	0184	TAPC	028C
TAP5	0185	TAPD	028D
TAP6	0186	TAPE	028E
TAP7	0187	TAPF	028F

If you omit the *device* parameter, CMS uses device TAP1.

(reg)

a register containing a pointer to a storage location that contains the device name or virtual device number.

The following example shows how you might use the register form to identify the device:

```
LA      2, MY181      Addr of device assignment
TAPESL  HOUT, (2), LABID=MYLABEL, ERROR=MYMSG
```

MY181	DC	CL4'0181'	vdev definition
-------	----	-----------	-----------------

SPACE=

controls the spacing for the HIN and EIN functions. Acceptable values are:

YES

requests that, after processing, CMS leave the tape positioned just past the tape mark at the end of the label file. This is the default value.

NO

requests that CMS leave the tape positioned just after the block containing the label it processed.

MODE=

indicates a recording format. This is meaningful only if *function* is HOUT, EOUT, or EVOUT. Regardless of whether CMS is attempting to write on the tape, the macro call will fail if the tape device is not capable of writing the indicated recording format.

Note that this parameter has no effect if:

- The tape device is not positioned to the beginning of the volume.
- The recording format it specifies is not allowed to coexist on a volume with the recording format that is recorded at the beginning of the volume. See [z/VM: CMS User's Guide](#) for details on selecting recording formats.

Because of these restrictions and the usual requirements of tape label positioning and format, there is rarely any reason to code the MODE parameter.

Values are:

3490C

3490 Compacted recording format

3490B

3490 Basic recording format

XF

3480 Compacted recording format

18

3480 Basic recording format

3590C

3590 Compacted recording format

3590B

3590 Basic recording format

(,6250)

GCR recording format

(,1600)

PE recording format

(,800)

NRZI recording format

COMP

Any compacted recording format

NOCOMP

Any uncompact recording format

9

Any 9 track recording format

For compatibility with previous levels of VM, the following values are also valid, but the above values are preferred.

(,38K)

3480 Basic recording format

(18,38K)

3480 Basic recording format

(9,6250)

GCR recording format

(9,1600)

PE recording format

(9,800)

NRZI recording format

Note: The different syntax of this parameter is because of compatibility purposes.

If you omit the MODE parameter, CMS selects a recording format for you. CMS's choice may be the result of a previous TAPE command, so you may use the TAPESL macro in conjunction with the TAPE command to select tape recording formats. See [z/VM: CMS User's Guide](#) and the description of the TAPE command in the [z/VM: CMS Commands and Utilities Reference](#) for details and examples.

BLKCNT=blkcnt

specifies the block count to be inserted in an EOF1 or EOVI label on output or used to check against on input. This field is used for functions EOUT, EIN, or EVOUT only. If not specified, the output block count is set to 0. This field may also be specified as a register number enclosed within parentheses when a general register contains the block count.

TM=

controls placement of tape marks on the tape. This parameter is meaningful only for the HOUT, EOUT, and EVOUT functions.

The values are:

YES

requests that CMS place a single tape mark after a HDR1 or EOVI label and two tape marks after an EOF1 label, and leave the tape positioned after the tape marks. This is the default value.

NO

requests that CMS place no tape marks after the label and leave the tape positioned immediately after the label.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

If you request the EIN function and a block count error is detected, control transfers to your error routine. If you do not specify an error routine or the ERROR= address is the same as the normal return, a block count error causes message 425R to be issued.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L, *addr*, *mf_label*)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E, *addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The input functions HIN and EIN read a tape label and check to see if it is the type specified. They also check any fields in the tape label that have been specified explicitly (no default) in the LABELDEF statement (indicated by LABID). Any discrepancies between the fields in the LABELDEF statement and the fields on the tape label cause an error message to be issued and an error return to be made.
2. The output functions HOUT, EOUT, and EVOUT write a tape label of the requested type on the specified tape. The values of fields within the labels are those specified or defaulted to in the LABELDEF command. For more information on the LABELDEF command, see [z/VM: CMS Commands and Utilities Reference](#).
3. For a more complete discussion of tape label processing, see [z/VM: CMS Application Development Guide for Assembler](#).
4. The 9346 tape cartridge can only be written to at load point and at logical end of tape.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code**Meaning****4**

The tape has been manually rewound and unloaded (the requested tape function may not have been executed).

24

The *device* value is not one of the valid values or it is valid, but the virtual device attached is not a tape device.

28

LABELDEF cannot be found.

32

Error in checking tape label or block count error.

36

The output tape is file-protected.

39

Tape mount error.

40

End-of-file or end-of-tape occurred.

100

A tape I/O error occurred or the virtual tape device is not attached (does not exist) or the device is not capable of writing in the recording format specified by the MODE parameter.

104

Virtual storage capacity exceeded.

TRANTBL



Purpose

Use the TRANTBL macroinstruction to generate a DSECT for the system character set translation tables.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the TRANTBL macro expansion is labeled TRANTBL.

Usage Notes

1. The address of the system character set translation table is located in the LANGTRTS field of the LANGBLK control block for the application ID DMS. Use the LANGFIND function to locate the proper LANGBLK.

For more information on LANGBLK control block, see [“LANGBLK” on page 287](#).

For more information on LANGFIND function, see [“LANGFIND” on page 449](#).

2. The TRANTBL macroinstruction expands as follows:

TRANTBL	DSECT		
TRANSTD	DS	CL256	Standard uppercase table
TRAST77	DS	CL256	EBCDIC → 3277 Character Set
TRAST78	DS	CL256	EBCDIC → 3278 Character Set
TRAAPL77	DS	CL256	EBCDIC → 3277 APL Character Set
TRAAPL78	DS	CL256	EBCDIC → 3278 APL Character Set
TRATX77	DS	CL256	EBCDIC → 3277 Text Character Set
TRATX78	DS	CL256	EBCDIC → 3278 Text Character Set
TRAPL7EC	DS	CL256	EBCDIC → 3277/APL Compound Chars
TRAPL7CE	DS	CL256	3277/APL Compound Chars → EBCDIC
TRAPL8EC	DS	CL256	EBCDIC → 3278/APL Compound Chars
TRAPL8CE	DS	CL256	3278/APL Compound Chars → EBCDIC
TRATX7EC	DS	CL256	EBCDIC → 3277/Text Compound Char
TRATX7CE	DS	CL256	3277/Text Compound Char → EBCDIC
TRATX8EC	DS	CL256	EBCDIC → 3278/Text Compound Char
TRATX8CE	DS	CL256	3278/Text Compound Char → EBCDIC
TRATX7ES	DS	CL256	EBCDIC → 3277/Text Single Char
TRATX7SE	DS	CL256	3277/Text Single Char → EBCDIC
TRAPL2EC	DS	CL256	EBCDIC → 3278/Extended APL (APL2®) Compound Character Set
TRAST78C	DS	CL256	EBCDIC -> CECF Character Set

TVISECT

► TVISECT ◄

Purpose

Use the TVISECT macro to generate a DSECT for information to be passed to a nucleus extension module named DMSTVI. This interface routine, DMSTVI, can be used to give control to a different multivolume switching routine than the one supplied with z/VM (DMSTVS) or a tape management system.

Usage Notes

1. For more information on the TVISECT macro usage or the DMSTVI routine, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The TVISECT macroinstruction expands as follows:

```

MACRO
TVISECT

*
*   DSECT FOR THE PLIST PASSED TO DMSTVI
*
TVISECT DSECT
TVIMOD DS CL8'DMSTVI' MODULE NAME FOR SVC 202
TVIFUNCT DS CL8 CALL FUNCTION KEYWORD
*          SYSPARM - CALL FOR SYSPARM
*          PROCESSING
*          OPEN - CALL FROM OPEN MACRO
*          VOLIDBAD - TAPE / USER SPECIFIED
*          VOLIDS DO NOT MATCH
*          EOVS - CALL FOR END OF VOLUME
*          CLOSE - CALL FROM CLOSE MACRO
*
*   THE FOLLOWING FIELDS WILL BE FILLED IN FROM INFORMATION THAT
*   IS STORED IN THE LABSECT. REFER TO THE LABELDEF COMMAND FOR
*   MORE INFORMATION ABOUT THESE FIELDS.
*
TVIFILE DS CL8 DDNAME
TVIFID DS CL17 FILE ID (RIGHTMOST 17 CHARACTERS)
TVISEC DS CL1 SECURITY TYPE
TVIVOLID DS CL6 VOLUME ID TO BE MOUNTED
TVIVSEQ DS CL4 VOLUME SEQUENCE NUMBER
TVIFSEQ DS CL4 FILE SEQUENCE NUMBER
TVIGENN DS CL4 GENERATION NUMBER
TVIGENV DS CL2 GENERATION VERSION
TVICRD DS CL6 CREATION DATE
TVIEXD DS CL6 EXPIRATION DATE
TVISYSPL DS H LENGTH OF SYSPARM STRING
TVISYSPA DS A ADDRESS OF SYSPARM STRING
TVIFILID DS CL44 FILE IDENTIFIER
TVISCRAT DS CL8'SCRATCH' SCRATCH | NOSCRATC
*          SCRATCH WAS (OR WAS NOT)
*          SPECIFIED AS THE LABELDEF VOLID
*
*   THE FOLLOWING FIELDS WILL BE FILLED IN FROM INFORMATION
*   THAT IS STORED IN THE FCBSECT (SUPPLIED BY THE FILEDEF
*   COMMAND).
*
TVICUU DS CL4 VIRTUAL DEVICE NUMBER
*          (TAPO THROUGH TAPF)
*          X'180' - X'187', X'288' - X'28F'
TVILABEL DS CL8 Filled with tape label type code
TVIRFMT DS X RECORDING FORMAT
TVIMODE EQU TVIRFMT,1,C'X' Old label for TVIRFMT
TVIALT DS CL4 ALTERNATE TAPE DRIVE TAPE ID
TVIRING DS CL6'RING' RING | NORING - WRITE ENABLE RING
TVIFLAGS DS X FLAG FOR TVI EXPANSION FIELDS
*
*   BIT INDICATORS (MAY BE USED IN COMBINATIONS)
*   X'00' NO EXPANSION DATA AVAILABLE
*   X'80' DATA BEING PASSED FROM TAPE SUB-SYS

```

```

*          X'40' - RESERVED -
*          X'20' - RESERVED -
*          X'10' - RESERVED -
*          X'08' - RESERVED -
*          X'04' - RESERVED -
*          X'02' - RESERVED -
*          X'01' EXPANSION DATA FIELDS AVAILABLE
TVIBLKCT DS    F          BLOCK COUNT
* -----
* -- EXPANSION DATA FIELDS -----
*
* THE FOLLOWING FIELDS WILL BE FILLED IN FROM INFORMATION
* THAT IS STORED IN THE APPLICATION PROGRAM DCB BLOCK AT
* "CLOSE" TIME TO BE MADE AVAILABLE TO THE TAPE SUBSYSTEM.
*
* IF THE TAPE SUBSYSTEM HAS THESE VALUES AVAILABLE WHEN THE
* DMSTVI "OPEN" CALL IS DONE, CMS WILL ACCEPT THESE VALUES
* AS RETURNED OUTPUTS FOR SUBSTITUTION INTO THE FCB (FILEDEF),
* IF THE USER HAS NOT ALREADY SPECIFIED THEM. THE FCB
* INFORMATION WILL THEN BE USED TO REPLACE ANY NON-SPECIFIED
* DCB ATTRIBUTES DURING THE CMS OPEN PROCESSING, JUST AS IS
* DONE FOR STANDARD LABEL TAPE PROCESSING.
*
* -----
TVIRECFM DS    X          FILE RECORD FORMAT
* Basic record format bit definitions:
*   X'80' = Fixed length records          RECFM= F
*   X'40' = Variable length records       RECFM= V
*   X'00' = Undefined length records      RECFM= U
*   X'20' = ASCII Variable length records RECFM= D
* Modifiers which further define records:
*   X'10' = Blocking being done
*   X'08' = For Fixed records: Standard size blocks used
*           For Variable recs: Spanned records used
*   X'04' = ASA Control characters in use
*   X'02' = Machine Control characters in use
*   X'01' = Record Key length specified
TVILRECL DS    F          FILE LOGICAL RECORD LENGTH
TVIBLKSI DS    F          FILE BLOCK SIZE
* -----
TVIEND  DS      0D          END OF TVISECT
TVISIZE EQU    (*-TVISECT+7)/8  SIZE OF TVISECT IN DOUBLEWORDS

```

3. The TVIFUNCT field identifies the function being processed when the call to DMSTVI is made. Possible call function keywords are:

Table 23. TVIFUNCT keyword meanings

Keyword	Meaning
SYSPARM	call made during FILEDEF command processing, lets DMSTVI check the SYSPARM string for syntax errors.
OPEN	call made during OPEN macro processing before any tape I/O is performed, lets DMSTVI mount a tape.
EOV	call made during end of volume processing before volume switching occurs, lets DMSTVI substitute another volume switching routine for the VM supplied volume switching routine.
VOLIDBAD	call made during OPEN macro or end of volume processing if the volid specified by the user is different from the volid on the tape, lets DMSTVI mount the correct tape.
CLOSE	call made during CLOSE macro processing after all tape I/O is done, lets DMSTVI do any necessary clean-up from OPEN macro processing.

4. TVISECT declares TVIMODE as a synonym for TVIRFMT because the field was called TVIMODE in earlier releases of VM.

The TVIRFMT field shows the recording format being used for writes to the tape. Even if no writing has been done to the tape, this field contains a valid recording format code. This code is determined

by recording format options on the relevant FILEDEF command, with defaulting (when the FILEDEF command has not specified a specific recording format).

Table 24. TVIRFMT byte meanings

Format Name	Hex Code (1)	Density (BPI)	Tracks
NRZI	X'CB'	800	9
PE	X'C3'	1600	9
GCR	X'D3'	6250	9
3480 Basic	X'10'	(2)	(2)
3480 Compacted	X'60'	(2)	(2)
3490 Basic	X'20'	(2)	(2)
3490 Compacted	X'30'	(2)	(2)

Note:

- a. Prior releases may use different codes for the TVIRFMT values than these detailed. See [z/VM: Migration Guide](#).
- b. For more details on tape formats, see [z/VM: CMS User's Guide](#) under the 'Tape Recording Formats' section.

USERSAVE



Purpose

Use the USERSAVE macroinstruction to generate a DSECT for the USERSAVE control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the USERSAVE macro expansion is labeled USERSAVE.

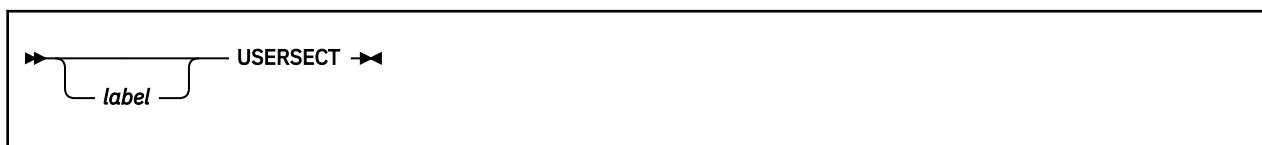
Usage Notes

1. For more information on the USERSAVE macro, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The USERSAVE macroinstruction expands as follows:

USERSAVE	USERSAVE	
	DSECT	
USERSAVE	DS 12D	Reserved for the user.
USERSIZE	EQU *-USERSAVE	Size of area reserved for user.
USERINFO	DS D	Information passed to user.
	ORG USERINFO	
USECTYP	DS X	Contains CALLTYP value.
USEUFLG	DS X	Contains UFLAGS value.
	DS 2X	Reserved for IBM use.
USEMFLG	DS X	Miscellaneous bits.
USECMS	EQU X'80'	Invoked by CMSCALL.
USEA31	EQU X'40'	Caller's AMODE is 31.
USESCBLK	EQU X'20'	SCBLOCK is available in R2.
USEPLIST	EQU X'10'	Extended PLIST available in R0,
*		only valid if invoked by CMSCALL.
USEAR	EQU X'08'	Caller was in AR mode when CMSCALL
*		issued.
	DS 3X	Reserved for IBM use.
USERSAVL	EQU (*-USERSAVE+7)/8	BLOCK LENGTH (DOUBLEWORD)

3. When a program receives control via SVC 202 or CMSCALL, R13 contains the address of the area mapped by the USERSAVE macro.

USERSECT



Purpose

Use the USERSECT macro to generate a DSECT for the USERSECT control block.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement. The first statement in the USERSECT macro expansion is labeled USERSECT.

Usage Notes

1. The address of USERSECT is found in the AUSER field of the NUCON macroinstruction. For more information, refer to macroinstruction “[NUCON](#)” on [page 333](#).
2. The USERSECT macroinstruction expands as follows:

```

                USERSECT
USERSECT DSECT
*
* GENERAL SCRATCH-STORAGE AREA PROVIDED FOR USER DEFINED PURPOSES
* NOTE -- MAY BE REDEFINED FOR INSTALLATION REQUIREMENTS
*
USCRTCH DC 18F'0'
```


VOLSECT

►► VOLSECT ◄◄

Purpose

Use the VOLSECT macroinstruction to generate a DSECT for additional user tape label information.

Usage Notes

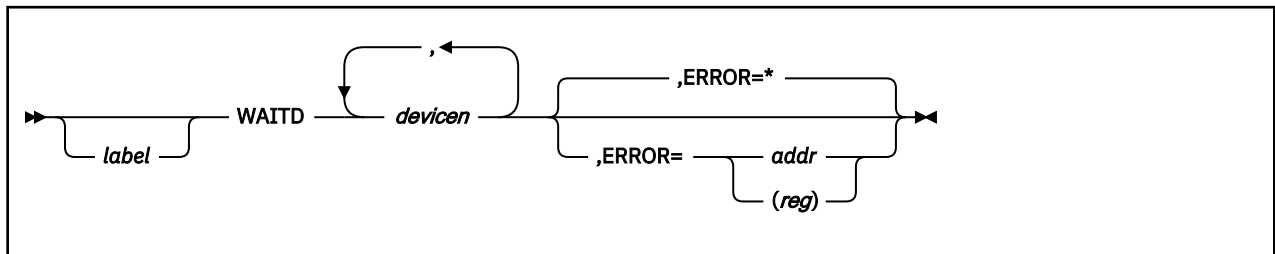
1. VOLSECT is supported for use with the tape label processing routines. For more information, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The VOLSECT control block holds up to 24 user-supplied volume IDs.
3. The first VOLSECT is forward pointed to by the LABVSECT pointer of the LABSECT macroinstruction, see “LABSECT” on [page 285](#) for more information.
4. The VOLSECT macro expands as follows:

```

MACRO
VOLSECT
***
*** VOLSECT - HOLDS ADDITIONAL USER SUPPLIED
*** VOLIDS. IF THE USER SUPPLIED MORE THAN 16 VOLIDS
*** STORAGE IS GOTTEN FOR A VOLSECT WHICH CAN HOLD 24 ADDITIONAL
*** VOLIDS. THE FIELD 'LABVSECT' IN LABSECT, POINTS TO THE FIRST
*** VOLSECT. ADDITIONAL VOLSECTS ARE FORWARD POINTED TO BY THE
*** FIELD 'VOLNSECT' CONTAINED IN THE 1ST 4 BYTES OF A VOLSECT.
SPACE 1
VOLSECT DSECT
VOLNSECT DS A FORWARD CHAIN POINTER
VOLAVOLS DS CL192 SPACE FOR 24 ADDITIONAL VOLIDS
VOLEND DS XL4'FF' FENCE FOR END OF VOLIDS
VOLSIZE EQU (*-VOLSECT+7)/8 SIZE OF VOLSECT IN DOUBLE WORDS

```

WAITD



Purpose

Use the WAITD macroinstruction to stop your program until the next interrupt occurs on the specified device.

Parameters

Required Parameters:

devicen

specifies the devices to be waited for. Specify device as:

symn

indicates the symbolic device name and number, where:

sym

is CON, DSK, PRT, PUN, RDR, or TAP.

n

indicates a device number.

user

is a 4-character symbolic name specified by a HNDINT or HNDIO macroinstruction issued for the same device.

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Usage Notes

1. The WAITD macroinstruction suspends program execution until the I/O interrupt for the specified device is processed. When the interrupt has been processed, WAITD stores the name of the interrupting device in register 1 and returns control to the caller.
2. If, for a specific device, you specify HNDIO or HNDINT with the ASAP option, the handler routine processes the interrupt when it receives it. If a subsequent WAITD is issued for the same device, the wait state is satisfied.

3. If you specify HNDIO or HNDINT with the WAIT option, and an interrupt is received, issue WAITD for the handler routine to receive control. (If you issue WAITD before the interrupt is received, the handler processes the interrupt immediately.)
4. The interrupt handler created by HNDINT or HNDIO determines whether an interrupt is considered processed or if more interrupts are necessary to satisfy the wait state. For more information on interrupt handling, see [*z/VM: CMS Application Development Guide for Assembler*](#).

Return Codes

If an error occurs, register 15 contains the following return code:

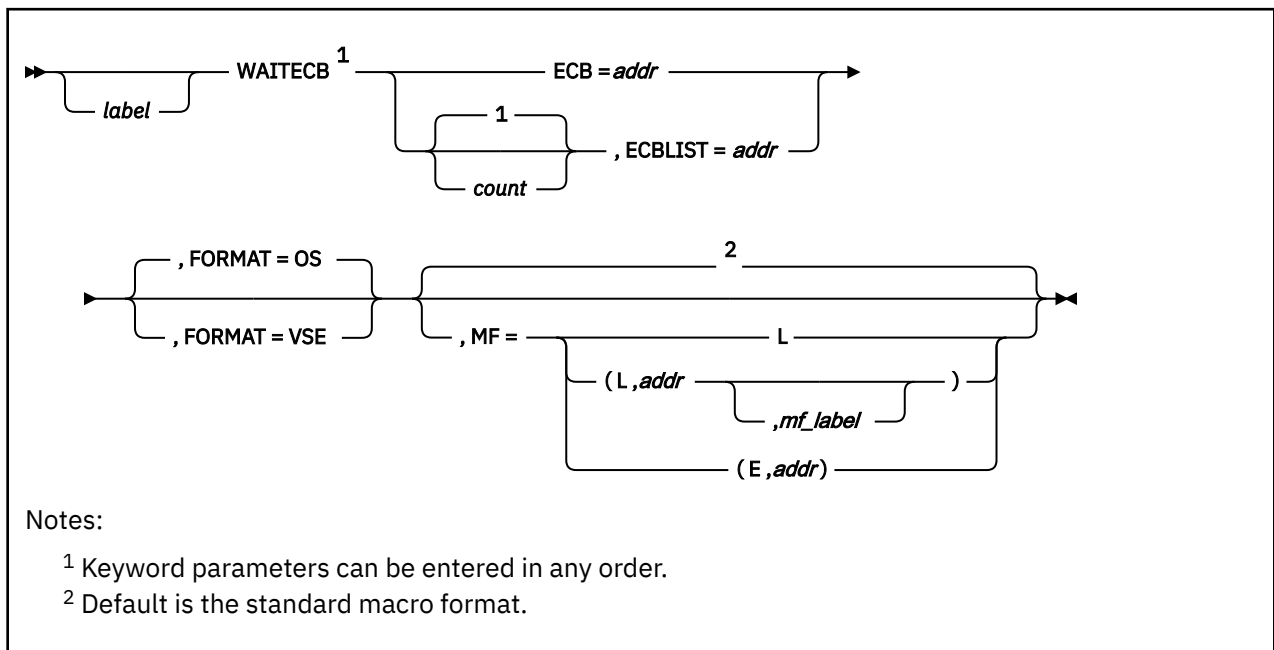
Code

Meaning

1

An invalid device number was specified.

WAITECB



Purpose

Use the WAITECB macroinstruction to wait on an event control block (ECB) or a list of event control blocks. Event control blocks are standard mechanisms used to synchronize multiple events. The process of turning on the *event complete* bit is referred to as *posting* the event control block. Asynchronous routines such as a timer or an external interrupt handler post event control blocks to signal completion. The WAITECB macro suspends processing until a specific event control block or the event control blocks in a list have been posted.

Parameters

Required Parameters:

ECB=addr

specifies the address of an event control block. It must be on a fullword boundary. Acceptable values for *addr* are:

label

specifies the event control block address as a program label.

(reg)

specifies a register that contains the event control block address.

ECBLIST=addr

specifies the address of a virtual storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an event control block. Acceptable values for *addr* are:

label

specifies the ECBLIST address as a program label.

(reg)

specifies a register that contains the ECBLIST address.

The end-of-list indicator has two forms: For FORMAT=OS, the high-order bit (bit 0) in the last fullword must be set to 1. For FORMAT=VSE, the byte following the last fullword of the list must be nonzero.

Optional Parameters:

label

is an optional assembler label for the statement.

count

specifies the number of event control blocks to be posted before returning to the caller. Specify it as a decimal number or a register (2-12) between parentheses. 1 is the default.

The count operand is valid with the ECBLIST form of the macro only. If you specify it with the ECB parameter, an MNOTE error message is issued. If the count is negative or 0, this function results in a no-op and the program proceeds to the next instruction.



Attention: If you specify a count larger than the number of event control blocks in the ECBLIST, execution of WAITECB results in a permanent wait.

FORMAT

specifies the format of the event control block(s) used. Acceptable values are:

OS

specifies the OS-format event control block. The bit tested for *event completed* is byte 0 bit 1. This is the default value.

VSE

specifies VSE-format event control block. The bit tested for event complete is byte 2 bit 0.

Note: If you specify ECBLIST, do not mix event control block formats.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. If you use ECBLIST to specify several event control blocks, CMS searches the event control blocks sequentially to determine if they have been posted.
2. For FORMAT=VSE, the ECBs must reside below 16 MB.
3. You can use the HNDEXT macro to post event control blocks. For more information on external interrupt handling, see *z/VM: CMS Application Development Guide for Assembler*.
4. The console event control block only applies to the terminal input buffer. No event control block is associated with the program stack.
5. Event control blocks are fullwords and have the following OS or VSE format:

• OS Format

bit 0	WAIT bit
bit 1	Event completed bit
bit 2-31	Completion code

• VSE Format

byte 0-1	Reserved
byte 2	

bit 0	Traffic bit
bit 1-7	Reserved
byte 3	Reserved

6. Console I/O Wait

The CMS nucleus contains an event control block that makes it easier to wait for a console I/O in a series of multiple events. This event control block is called CON1ECB and is defined in CMS system storage. CON1ECB has two *event completed* bits. The format of CON1ECB is:

byte 0	
bit 0	Wait bit
bit 1	Event completed bit number 1
bit 2-7	Completion code
byte 1	Reserved
byte 2	
bit 0	Event completed bit number 2
bit 1-7	Reserved
byte 3	Reserved

When the terminal input buffer contains a line, both event completed bits in the CON1ECB ECB are posted.

To obtain the CON1ECB field in the CMS nucleus, obtain the device information for the console from location ADEVTAB using the NUCON macroinstruction. CON1ECB is located at offset X'C' into this information. For more information, see [“NUCON” on page 333](#) for a description of the NUCON macroinstruction.

7. CMS signals the VMCONINPUT event whenever it receives unsolicited attention interrupts from the virtual machine console. It is a broadcast event with session scope and does not synchronize the handling process. It contains no event data. The monitoring application should perform a read operation to the console to obtain the input data. It has a loose signal limit of zero, so when an event monitor is created for this event, previously signaled console input notifications will not be seen by the corresponding event handler. See *z/VM: CMS Application Multitasking* for more information.
8. WAITECB serializes the virtual machine and may not be desirable for multitasking applications. See *z/VM: CMS Application Multitasking* for more information on event handling in a multitasking environment.

WAITT



Purpose

Use the WAITT macroinstruction to cause the program to wait until all of the pending terminal I/O completes.

Parameters

Optional Parameters:

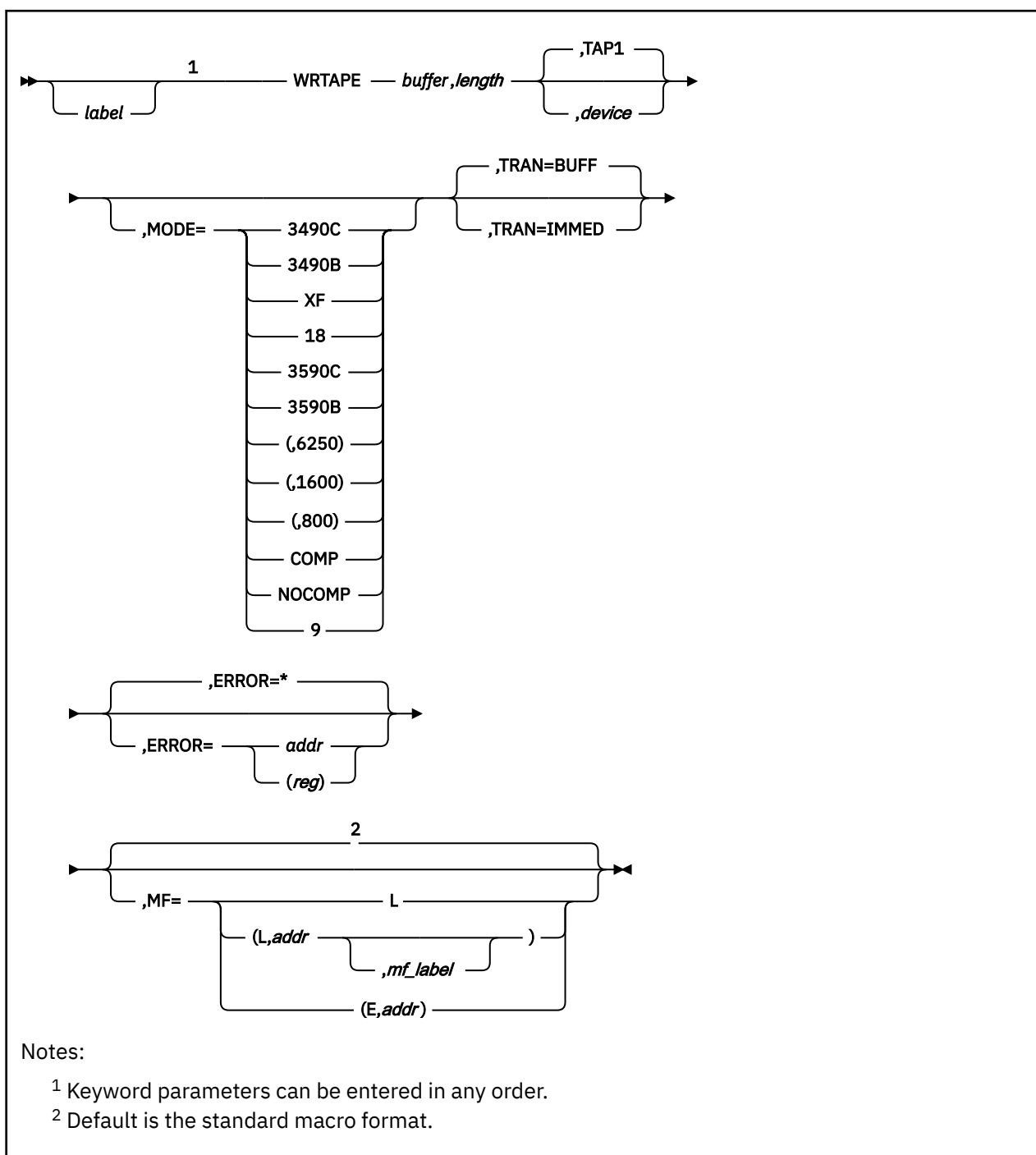
label

is an optional assembler label for the statement.

Usage Notes

1. The WAITT macroinstruction synchronizes input and output to the terminal; it ensures that any pending I/O is cleared before the program continues execution. You can also use WAITT to make sure that a read or write operation finishes before you issue another I/O operation.
2. For programs that are disabled, the WAITT macroinstruction may enable for interrupts. Use the ENABLE macro to reset the previous interrupt mask if you want to return to a disabled state.

WRTAPE



Purpose

Use the WRTAPE macroinstruction to write a block on the specified tape device.

Parameters

Required Parameters:

buffer

specifies the address of the buffer containing the block to be written. Acceptable values are:

lineaddr

the symbolic address of the line.

(reg)

a register containing the address of the buffer.

length

specifies the length of the block to be written, and thus the length of the buffer that contains the block. Acceptable values are:

n

a self-defining term indicating the length.

(reg)

a register containing the length.

Note: For a 9346 tape device, the length should be 32 KB or less.

Optional Parameters:

label

is an optional assembler label for the statement.

device

specifies the device name (TAP*n*) or virtual device number (*vdev*) of the virtual tape device from which the block is to be read. The following values are valid; see [z/VM: CMS User's Guide](#) for information on tape device names and virtual device numbers for tape devices.

Device Name	Virtual Number	Device Name	Virtual Number
TAP0	0180	TAP8	0288
TAP1	0181	TAP9	0289
TAP2	0182	TAPA	028A
TAP3	0183	TAPB	028B
TAP4	0184	TAPC	028C
TAP5	0185	TAPD	028D
TAP6	0186	TAPE	028E
TAP7	0187	TAPF	028F

If you omit the *device* parameter, CMS uses device TAP1.

(reg)

a register containing a pointer to a storage location that contains the device name or virtual device number.

The following example shows how you might use the register form to identify the device:

*	LA	2, MY181	Addr of device assignment
	WRTAPE	OUTBUF,4096,(2),	ERROR=MYEMSG
			Write block(4096 bytes)
	:		
	:		
MY181	DC	CL4'0181'	vdev definition

MODE=

specifies the recording format. Note that this parameter has no effect if the tape device is not positioned to the beginning of the volume and the recording format it specifies is not allowed to coexist on a volume with the recording format that is recorded at the beginning of the volume. For more information on selecting recording formats, see [z/VM: CMS User's Guide](#).

Values are:

3490C

3490 Compacted recording format

3490B

3490 Basic recording format

XF

3480 Compacted recording format

18

3480 Basic recording format

3590C

3590 Compacted recording format

3590B

3590 Basic recording format

(,6250)

GCR recording format

(,1600)

PE recording format

(,800)

NRZI recording format

COMP

Any compacted recording format

NOCOMP

Any uncompacted recording format

9

Any 9 track recording format

For compatibility with previous levels of VM, the following values are also valid, but the above values are preferred.

(,38K)

3480 Basic recording format

(18,38K)

3480 Basic recording format

(9,6250)

GCR recording format

(9,1600)

PE recording format

(9,800)

NRZI recording format

Note: The different syntax of this parameter is because of compatibility purposes.

If you omit the MODE parameter, CMS selects a recording format for you. CMS's choice may be the result of a previous TAPE command, so you may use the WRTAPE macro in conjunction with the TAPE command to select tape recording formats. See *z/VM: CMS User's Guide* and the description of the TAPE command in *z/VM: CMS Commands and Utilities Reference* for details and examples.

TRAN=

specifies the tape write mode, either buffered write mode (TRAN=BUFF) or immediate write mode (TRAN=IMMED). Buffered write mode is available only with a buffered tape device, like most newer tape devices. With a nonbuffered tape device, CMS ignores the TRAN parameter and always operates in immediate write mode. With a buffered tape device, the write mode defaults to buffered and specifying immediate write mode causes a severe performance degradation.

BUFF

specifies that the WRTAPE macroinstruction completes execution as soon as the data has been transferred to the tape subsystem's buffer. If the data was successfully transferred to the buffer,

the WRTAPE macroinstruction return code is 0. If the tape subsystem is unable to correctly write the data onto the tape because of subsystem or media failure, this will result in an error indication on a subsequent I/O request for the device.

You can use the REW, RUN, BSR, BSF, or WTM functions of the TAPECTL macroinstruction to force a buffer synchronization. That is, the TAPECTL macro execution will not complete until the tape subsystem has attempted to write all previously buffered write data onto the tape. If the tape subsystem was unable to correctly write all of the data onto the tape, the TAPECTL macroinstruction execution will complete with a return code of 3. If TAPECTL completes with return code 0, all data from previous write requests has been successfully written onto the tape.

IMMED

specifies that execution is not completed until the subsystem has written the data on the tape and verified it. If the tape subsystem is unable to correctly write the data onto the tape, the WRTAPE return code is 3. If the WRTAPE return code is 0, the data was correctly written on the tape.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E,addr)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

Usage Notes

1. The maximum supported tape block length is recorded in NUCON field MAXTAPBS. If the value in this field is 0, the maximum block length is 65,535 bytes.
2. The 9346 tape cartridge can only be written to at load point and at logical end of tape.

Return Codes

The return codes (found in register 15) from a WRTAPE call are listed below. Detailed explanations of them follow.

Code

Meaning

0

The WRTAPE call executed normally.

- 1** Invalid parameter or the specified device is incapable of writing in the specified format.
- 2** In End of Volume area.
- 3** I/O error.
- 4** Invalid *device* value.
- 5** Virtual tape device not attached (device does not exist).
- 6** Volume is write protected.
- 7** Specified device is not a tape device.
- 8** Block too big.
- 9** Manual rewind/unload of tape.

Return Code 0: WRTAPE executed normally. A block has been successfully written. The volume is positioned just after it. If the device is a buffered device and you are using TRAN=BUFF, the block may not be physically on the medium yet (it may be in the device's internal buffer instead).

The volume is not positioned in the End of Volume area. If it were, you would get return code 2 instead.

Return Code 1: Invalid parameter or bad format. One of the following is true of the WRTAPE call:

- One of the parameter values is not valid.
- The parameter values are not compatible with each other.
- The MODE parameter indicates a recording format which the device (identified by the *device* parameter) is not capable of writing.

For the invalid parameter cases, the WRTAPE invocation needs to be corrected. For the recording format incapable case, you can either specify a different recording format, specify a different device, or attach a different device. It may be better to specify a nonspecific recording format (like MODE=COMP or omit MODE completely), so that CMS chooses a recording format that the device *can* write. See [z/VM: CMS User's Guide](#) for information on recording formats and device capabilities.

Return Code 2: In End of Volume area. This is the same as return code 0 except that the volume is positioned in the End of Volume area as a result of writing the block. This gives you early warning that you are about to run out of tape. If you keep writing, the write will eventually fail when you are out of tape. See return code 3.

Return Code 3: I/O error. The device was unable to write a block for many reasons. Following is a list of some of these reasons. You should not consider any data to have been recorded on the medium or that anything you are trying to write over is still on the medium. You should not assume any particular positioning of the device.

- The device or channel has detected an internal malfunction in the device or channel.
- There is a defect on the recording medium.
- The tape reel or cartridge is damaged.
- There is no room left on the volume for the block. Note that CMS gives you warning through return code 2 when you are running out of space on the volume.
- The device is a buffered device and the device has been unable to write buffered data on the physical medium. CMS does not provide a way through a General Programming Interface to determine which block had the problem, because there could be many of them buffered. The block with the problem and

every one after it that is now in the buffer will be thrown away and the device remains positioned just after the last correctly written block.

- The tape cartridge contains a length of tape that the device cannot properly handle, so the device is preventing you from writing on it. The position of the volume has not changed. Another device may be able to write the block. Otherwise, you can copy the data on the cartridge to a cartridge which the device can handle. Note that the same device may allow you to read the tape even though it will not write it. You take a risk of damaging the device or the medium by reading it, but the device allows it to give you a chance of recovering your data.
- The device is in a Volume Fenced condition. This is a condition which arises for reasons in which the device will not perform most operations on the volume. You can undo this condition by unloading the device; other times by rewinding the device. You can do either of these with the TAPE command or TAPECTL macroinstruction.
- The virtual device is a *shareable* one (see *z/VM: CMS User's Guide*). CMS does not support shareable devices and the failure of WRTAPE in this way is just one of the possible effects.

Return Code 4: Invalid device value. The value of the *device* parameter is not one of the valid ones listed for it. CMS cannot tell to what device to write. The WRTAPE invocation must be corrected.

Return Code 5: Device not attached. No virtual device exists with the virtual device number given by the *device* parameter or, if *device* specifies a device name, with the device number CMS associates with that name.

You must either specify a different device name or number or create one with the proper virtual device number. The *z/VM: CMS User's Guide* explains this.

Return Code 6: Volume is write protected. The volume was mounted in read only. If it is a cartridge, its write protect switch is activated. If it is a reel, it doesn't have a write enable ring present.

Nothing has been written on the volume and its position has not changed.

To write on the volume, the real volume must be unloaded and an operator must write enable the volume before loading it again.

Return Code 7: Device is not a tape device. The device which has the device number given by the *device* parameter or, if *device* specifies a device name, with the device number CMS associates with that name, is not a tape device.

You must either specify a different device name or number or detach the attached device and create a tape device instead with that virtual device number.

Return Code 8: Block too big. The block you are attempting to write is too big for the device to handle. A different device may be able to write it.

Return Code 9: Manual rewind/unload. Someone has previously rewound or unloaded the volume on the real device associated with the virtual device by operating manual controls on the physical device. In order to warn you of this, CMS has returned this return code to WRTAPE without attempting to write anything.

You get this warning once, so if you want to write the block, you can just repeat the WRTAPE call.

CMS gives you this warning because the volume on which you intended to write may not be mounted now.

With older devices, you do not get this warning.

WUERROR

➤ WUERROR ➤

Purpose

Use the WUERROR macroinstruction to map the work unit extended error information returned in the *wuerror* parameter of callable services library (CSL) routines.

Usage Notes

- 1. The information in the WUERROR and FPERROR buffers can also be accessed as individual variables by using the CSL routine DMSWUERR (SFS WUERROR Deblocker). This routine is described in [z/VM: CMS Callable Services Reference](#).
- 2. The WUE... fields are filled in by CMS on a call to a shared file system (SFS) CSL routine when the *wuerror* parameter is used.
- 3. To receive all of the extended error information on a request, provide an area where the length of the *wuerror* buffer is:

$$12 + (N * \text{length of FPERROR})$$

where N is the number of file pools connected to under the work unit used for the request. If *namedefs* are used, add to N the number of *namedefs* defined for directories. The maximum for N is the number of file pools connected to under the work unit, plus the number of *namedefs* defined for directories (this may result in more space being allocated than is actually needed if more than one *namedef* is for the same file pool).

- 4. The WUERROR macroinstruction expands as follows:

The following describes what the Work Unit Extended Error Information area contains.

WUELEN	DS	F	Length of WUERROR buffer
WUERTNED	DS	F	Number of FP Error Info areas returned
WUETOTAL	DS	F	Total number of FP Error Info areas from SFS
WUEAREA	EQU	*	Area to put FPERROR data

Chapter 3. CMS Preferred Functions

This chapter describes the formats of the preferred CMS assembler language functions. All of the functions in this chapter are capable of supporting 31-bit addressing and run in ESA, XA, and XC virtual machines.

To execute CMS functions from application programs, set up a parameter list and then issue the CMSCALL macroinstruction.

The following CMS functions are described in this section:

- DISKID
- DMSSEQ
- LANGADD
- LANGFIND.

DISKID

Purpose

Use the DISKID function to obtain information on the physical organization of a reserved minidisk. The DISKID function obtains the virtual device number, the block size, and the offset of the minidisk.

A disk does not have to be accessed when you use the DISKID function.

You need this information to use the CP DASD Block I/O System Service with a CMS formatted minidisk that was reserved.

You can use CMSCALL with the following parameter list to execute DISKID from an assembler language program:

PLIST	DS	0D	
	DC	CL8'DISKID'	
	DC	CL8' <i>ddname</i> '	<i>ddname</i> for the minidisk
	DS	XL2	Virtual device number
	DS	H	Blocksize
	DS	F	Offset
	DS	D	Reserved

Usage Notes

1. The parameter list must begin on a doubleword boundary.
2. The program calling the DISKID function fills the first two doublewords of the parameter list.
3. The second doubleword contains the *ddname* specified in the FILEDEF command issued for the Block I/O disk.
4. The third doubleword is filled upon completion of the DISKID function. It contains:
 - The virtual device number of the minidisk for which a *ddname* exists
 - The block size of the minidisk
 - The offset of the minidisk. This offset associates a physical block number to the first data block of the unique file on disk that was previously reserved. The block number represents the number of sequential blocks used on the disk by the CMS file system to implement its structure. Data block number one for the file is then physical block number one plus offset.

Return Codes

On return from DISKID, register 15 contains one of the following codes:

Code

Meaning

0

Return information supplied in parameter list.

4

The function was called from a terminal or EXEC.

12

DASD not reserved with the RESERVE command.

28

ddname not defined or *ddname* not given a file definition to DISK vdev.

100

Disk not attached.

1xx

An I/O error occurred reading the volume label; xx = the return code from CP DIAGNOSE code X'A8'.

2xx

An I/O error occurred reading the volume label; xx = the return code from CP DIAGNOSE code X'20'.

DMSSEQ

Purpose

Use the DMSSEQ function to count the number of logical lines in the terminal input buffer. The number is returned as the return code of the call in register 15.

You can use the CMSCALL macro with the following parameter list to run DMSSEQ from a program:

```
DS  0D  
DC  CL8 'DMSSEQ'  
DC  FL8 '-1'
```

Usage Notes

1. DMSSEQ does not issue any error messages.

Return Codes

On return from DMSSEQ, register 15 contains the number of logical lines in the terminal input buffer.

LANGADD

Purpose

Use the LANGADD function to add a LANGBLK to the language block chain.

LANGADD does this by:

1. Making a copy of the LANGBLK pointed to in the parameter list.
2. Adding the copy to the language block chain of LANGBLKs (if a LANGBLK for the application is not already on the chain).

This allows an application to have one language as part of its *nucleus* just as CMS does.

You can use CMSCALL with the following parameter list to execute LANGADD from a program:

```
DS  OF
DC  CL8'LANGADD'
DC  A(addr of LANGBLK)
DS  A                      addr of active LANGBLK
DC  8X'FF'
```

Usage Notes

1. Upon return, the fourth fullword of the parameter list contains the address of the active LANGBLK.
2. The SET LANGUAGE command is unable to restore the information in the LANGBLK if another language is set and the original language is restored. The application must request LANGADD to add the LANGBLK again.
3. The SET LANGUAGE command is used for:
 - user repository and tables
 - system tables and referencing in an NLS segment.
4. For more information on using other national languages supported by VM and on installing a different system national language, see [z/VM: Installation Guide](#).
5. Prior to using the LANGADD function, the following LANGBLK field definition is required:

- LANGAPID-3 character application ID

The following LANGBLK field definitions are set as required if the application wants to use its own:

- LANGMSG-system message repository
- LANGSPA-system parser table
- LANGSSY-system synonym and abbreviation table
- LANGTRTS-NLS translation table
- LANGDISK-application HELP disk address.

Set all fields that are not being used to binary zero.

Return Codes

On return, register 15 contains one of the following return codes:

Code

Meaning

0

The function was successfully completed.

24

A LANGBLK for the application is already on the language block chain.

104

Insufficient storage is available.

LANGFIND

Purpose

Use the LANGFIND function to get the address of an application's language control block.

Each application may have a language control block (LANGBLK) which contains pointers to all language-related information. LANGFIND lets you locate the LANGBLK for a specific application by using the 3-character application ID. For the purpose of fullword alignment, the application ID is defined as 4 characters (the fourth character is ignored).

You can use CMSCALL with the following parameter list to execute LANGFIND from a program:

```
DS  0F
DC  CL8'LANGFIND'
DC  CL4'xxx'          3 character application id
DS  A                  addr of LANGBLK
DC  8X'FF'
```

Upon return, the 4 bytes following the application ID contain:

- The address of the LANGBLK, if the application ID requested was found
- Zero, if no LANGBLK contained the application ID that was requested.

Part 2. Compatibility Programming Interface

This section defines the macros and functions CMS supports for compatibility only. The following chapters describe these macros and functions, which make up the CMS compatibility programming interface:

- Chapter 4, “[CMS Compatibility Macros](#),” on [page 453](#) describes the CMS macros in the CMS compatibility interface group.
- Chapter 5, “[CMS Compatibility Functions](#),” on [page 493](#) describes the CMS functions that are considered part of the compatibility interface group.

Chapter 4. CMS Compatibility Macros

This chapter describes the macros CMS supports for compatibility only.

The macros in the CMS compatibility group do not support 31-bit addressing. Existing programs can continue to use the macros in programs that do not support 31-bit addressing. IBM does not recommend the use of the compatibility group macros in new programs.

The CMS compatibility group macros described in this section are:

- CMSCB
- DISPW
- DMSEXs
- DMSFREE
- DMSFRES
- DMSFRET
- DMSKEY
- IO
- LINEDIT
- RDTERM
- STRINIT
- TEOVEXIT
- WRTERM.

CMSCB

►► CMSCB ◄◄

Purpose

The CMSCB Macro maps the FCBSECT and IHADECB DSECTs.

FCBSECT consists of the CMS file control block (FCB) used for file management under CMS, the simulated OS job file control block (JFCB), input/output block (IOB), and data extent block (DEB). FCBSECT is dynamically allocated from CMS free storage each time the FILEDEF command is issued.

Usage Notes

1. This macro is contained in DMSOM MACLIB. The CMSCB and IO macros map internal CMS data areas and are used with the TEOVEXIT macro and FILEDEF AUXPROC facility to monitor or modify I/O operations in CMS. For more information on FILEDEF AUXPROC, see the [z/VM: CMS Application Development Guide for Assembler](#).
2. The CMSCB macroinstruction maps the FCBSECT DSECT as follows:

```
*
*   SIMULATED OS CONTROL BLOCKS
*
FCBSECT  DSECT
FCBINIT  DS      0X -      INTERESTING TIDBITS
FCBOPCB  EQU     X'08' -    OPEN ACQUIRED THIS CMS BLOCK
FCBPERM  EQU     X'04' -    PERMANENT CONTROL BLOCK
FCBBATCH EQU     X'02' -    SPECIAL BATCH DATA SET
FCBCATML EQU     X'01' -    CONCATENATED MACLIB DATA SET
FCBOS    EQU     X'10' -    FCB FOR OS FORMATTED DISK
FCBDOSL  EQU     X'20' -    CONCATENATED DOSLIB DATA SET
FCBCATLD EQU     X'40' -    CONCATENATED OS LOADLIB
FCBDID   EQU     X'80' -    ASSOCIATE DDNAME WITH ENTIRE
*                                     DISK FOR DISKID USAGE
FCBNEXT  DS      A -      AL3(NEXT CMSCB)
FCBPROC  DS      A -      A(SPECIAL PROCESSING ROUTINE)
FCBDD    DS      CL8 -    DATA DEFINITION NAME
FCBOP    DS      CL8 -    CMS OPERATION
IHAJFCB  DS      0D -    *** JOB FILE CONTROL BLOCK ***
JFCBDSNM DS      0X -    44 BYTES, DATA SET NAME
FCBDSNAM DS      CL8 -    DATA SET NAME
FCBDSTYP DS      CL8 -    DATA SET TYPE
FCBPRPU  EQU     FCBSTYP+4 - PRINTER/PUNCH COMMAND LIST
FCBTBSP  DS      0X -      2 BYTES, TAPE BACKSPCE COUNT
FCBDSMD  DS      CL2 -    DATA SET MODE
FCBDSMDC DS      CL2 -    Saved concat data set mode
FCBBUFF  DS      F -      A(INPUT-OUTPUT BUFFER)
FCBBYTE  DS      F -      DATA COUNT
FCBFORM  DS      CL1 -    FILE FORMAT: FIXED/VARIABLE RECS
FCBFLG   DS      X -      =FSCBFLG flag byte for extended plist bit
FCBFLG2  DS      X -      =FSCBFLG2 extended plist flag byte
FCBOTYP  DS      X -      OPEN intent (R,W,X,N)
FCBREAD  DS      F -      N'BYTES ACTUALLY READ
FCBITEM  DS      F -      EXTENDED PLIST ITEM COUNT.
FCBCOUT  DS      F -      EX. PLIST RECORDS / PHYSICAL BLK.
FCBWPTR  DS      F -      EXTENDED PLIST WRITE PTR.
FCBRPTR  DS      F -      EXTENDED PLIST READ PTR.
FCBDEV   DS      X -      DEVICE TYPE CODE
FCBDUM   EQU     0 -      DUMMY DEVICE
FCBPTR   EQU     4 -      PRINTER
FCBRDR   EQU     8 -      READER
FCBCON   EQU     12 -     CONSOLE TERMINAL
FCBTAP   EQU     16 -     TAPE
FCBDSK   EQU     20 -     DISK
FCBPCH   EQU     24 -     PUNCH
FCBCRT   EQU     28 -     CRT
FCBVSAM  EQU     32 -     VSAM
FCBRFMT  DS      X -      --> Working Recording Format
```

```

* FCBRFMT is defined for a tape file only. It is a TAPEIO
* recording format code denoting the recording format in
* effect with the file. While no file is open, it simply
* reflects the recording format request from the FILEDEF,
* which may be a nonspecific recording format code. While
* a file is open, though, it is always a specific recording
* format code -- the one indicated by the FILEDEF
* specification in conjunction with the capabilities of the
* device. See also FCBOFMT.
*
FCBOFMT DS X --> Original Recording Format
* FCBOFMT is defined only for a tape file. It is set by
* DMSFLO when the Filedef command is issued and should NEVER
* be changed anywhere else. This means that we can ALWAYS
* restore the original recording format after it has been
* changed. When the tape file is not open, FCBRFMT=FCBOFMT
* unless an Open error occurred, in which case the value of
* FCBRFMT is unpredictable.
*
DS X - RESERVED
FCBRECL DS H - DCB LRECL AT OPEN TIME
IOBIOFLG DS X - I/O FLAGS
FCBDCBCT DS X - NO. OF DCB'S USING THIS FCB
FCBR13 DS F - SAVEAREA VECTOR R13
FCBKEYS DS A - A(DDS IN'CORE KEY TABLE)
FCBPDS DS A - A(PDS IN-CORE DIRECTORY)
JFCBMASK DS 8X - VARIOUS MASK BITS
JFCBCRDT DS 3C - DATA SET CREATION DATE (YDD)
JFCBXPDT DS 3C - DATA SET EXPIRATION DATE (YDD)
JFCBIND1 DS X - INDICATOR ONE
JFCBIND2 DS X - INDICATOR TWO
JFCMOD EQU X'80' - DISP MOD specified on FILEDEF
* command.
JFCOLD EQU X'40' - DISP OLD specified on FILEDEF
* command.
JFCLIBSV EQU X'04' - Lib Dataserver usage noted by
* OPEN.
JFCEXTND EQU X'02' - EXTEND specified on OPEN macro
* (This flag is used only for the
* duration of OPEN processing -
* CMS usage differs from OS/MVS)
JFCM4FLG EQU X'01' - This is a filemode number 4
* file (CMS usage differs from
* OS/MVS)
*
JFCBUFNO DS X - NUMBER OF BUFFERS
JFCBFTEK DS 0X - BUFFERING TECHNIQUE
JFCBFALN DS X - BUFFER ALIGNMENT
JFCBUFL DS H - BUFFER LENGTH
JFCEROPT DS X - ERROR OPTION
JFCKEYLE DS X - KEYLENGTH
DS X - ---NOT USED---
JFCLIMCT DS 3X - BDAM SEARCH LIMIT
FCBDSORG DS 0X - DATA SET ORGANIZATION
JFCDSORG DS 2X -
FCBRECFM DS 0X - RECORD FORMAT
JFCRECFM DS X -
JFCOPTCD DS X - OPTION CODES
FCBBLKSZ DS 0H - BLOCK SIZE
JFCBLKSI DS H -
FCBLRECL DS 0H - LOGICAL RECORD LENGTH
JFCLRECL DS H -
FCBIOSW DS X - I/O OPERATION INDICATOR
FCBCLOSE EQU X'80' - DURING "CLOSE"
FCBMASTR EQU X'40' - Master FCB for Concatenation
FCBPROCC EQU X'20' - GOTO FCBPROC DURING CLOSE
FCBPROCO EQU X'10' - GOTO FCBPROC DURING OPEN
FCBCASE EQU X'08' - ON=LOWER CASE CONSOLE I/O
FCBPVMB EQU X'04' - PUT-MOVE-VAR-BLK
FCBIOWR EQU X'02' - WRITE/PUT
FCBIORD EQU X'01' - READ/GET
FCBIOSW2 DS 1X - I/O OPERATION INDICATORS
FCBMVPDS EQU X'01' - SW FOR MOVEFILE WITH PDS OPTION
FCBMV EQU X'02' - MOVE PDS SWITCH FOR FIND
FCBBYSVC EQU X'04' - The function currently in control
* was invoked via SVC. Used by
* routines that are called during
* FEOV processing.
FCBMVFIL EQU X'08' - Movefile is active
FCBCLEAV EQU X'10' - LEAVE positioning on Close
FCBCRERD EQU X'20' - REREAD positioning on Close
FCBTCLOS EQU X'40' - A CLOSE TYPE T was done
FCBWRTSW EQU X'80' - INDICATE DCB OPEN FOR WRITE

```

```

DEBLNGTH DS 0X - L'DEB IN DBLW WORDS
DS F - ---NOT USED---
IHADEB DS 0D - *** DATA EXTENT BLOCK ***
DEBTCBAD DS A - A(MOVE-MODE USER BUFFER)
SEBSAV DS F DYNAMIC SAVE FOR RET ADDR FOR
* SEB (OS I/O SIM)
DEBOFLGS DS 4X - DATA SET STAUS FLAGS
DEBOPATB DS 4X - OPEN/CLOSE OPTION BYTE
IOBFLG DS 0X - (START OF IOBPREFIX FOR NORMAL SCH)
IOBBFLG EQU 0 - DISPLACEMENT OF IOB FLAG IN IOB
IOBOUT EQU X'40' - "WRITE,PUT" IN PROGRESS
IOBIN EQU X'20' - "READ,GET" IN PROGRESS
IOBUPD EQU X'10' - "QSAM PUTX" IN PROCESS
IOBNXTAD DS A - A(NEXT BUFFER TO BE USED)
IOBECB DS F - ECB FOR QSAM NORMAL SCHEDULING
IHAIOB DS 0F - *** INPUT/OUTPUT BLOCK ***
DEBDEBID DS 0X - DEB IDENTIFICATION
DEBDCBAD DS A - A(DATA CONTROL BLOCK)
IOBECBCC DS 0X - ECB COMPLETION CODE
IOBBECBC EQU 12 - DISPLACEMENT OF ECB CODE IN IOB
IOBBECBP EQU 12 - DISPLACEMENT OF ECB PTR IN IOB
IOBECBPT DS A - A(EVENT CONTROL BLOCK)
IOBFLAG3 DS 0X - I/O ERROR FLAG
IOBBCSW EQU 16 - DISPLACEMENT OF CSW IN IOB
IOBCSW DS 8X - LAST CCW STORED(I.E., RESIDUAL COUNT)
IOBSTART DS A - X'ID-NEXT BUFFER',AL3(INITIAL BUFFER)
IOBDCBPT DS A - A(DATA CONTROL BLOCK)
IOBEND DS 0X - END-OF-INPUT/OUTPUT BLOCK
FCBMEMBR DS 2F OS PDS MEMBER NAME
FCBOSFST DS F POINTER TO OS FST
FCBOSDSN DS F POINTER TO OS DSNAME BLOCK
FCBXTENT DS F - NUMBER OF ITEMS IN EXTENT
FCBTEOV DS A - ADDRESS OF TEOVEXIT ROUTINE.
FCBTSAVE DS A - ADDRESS OF SYSTEM REGISTER
* SAVE AREA FOR TEOVEXIT.
FCBFLAG1 DS 1X - MISC. FLAG BITS.
FCBTEOVS EQU X'80' TAPE END-OF-VOLUME EXIT AVAILABLE
FCBTEOVA EQU X'40' TEOV EXIT IS ACTIVE.
FCBMVOL EQU X'20' PROCESSING MULTIVOLUME FILE
FCBVSECT EQU X'10' PROCESSING VOLIDS FROM A VOLSECT
FCBMTCAN EQU X'08' TAPE MOUNT CANCELED FROM DMSTVS
* Bits used for SFS directory-resident files:
FCBSPCHK EQU X'04' FSWRITE tracks file's SFS filespace usage
FCBDIR EQU X'02' this file is on an SFS directory
FCBERASE EQU X'01' erase file when it is closed: this flag is
* used ONLY for SFS directory files
FCBVCTR DS 1X - VOLID COUNTER
* The following field has two uses:
* It used as the volume sequence number for tape
* and as the original blocksize saved by DMSSOP for Console
FCBVSEQ DS H'0' TAPE VOLUME SEQUENCE NUMBER
FCBCNBLK EQU FCBVSEQ Console Original Blocksize
FCBALT DS F ALTERNATE TAPE DRIVE TAPID
* Or temp save of fcbosfst for dmssop
FCBTVIPL DS A DMSTVI PLIST ADDRESS
FCBSYSPA DS A SYSPARM STRING ADDRESS
FCBSYSPL DS H'0' SYSPARM STRING LENGTH (DWORDS)
FCBSYSPB DS H'0' SYSPARM STRING LENGTH (BYTES)
FCBDISP EQU IHADEB-FCBINIT Displacement of FCB address
FCBFLAG3 DS 1X -
FCBFMAST EQU X'80' FCB filemode was asterisk
FCBUSASI EQU X'40' Buffer Offset Flag bit for ANSI
* DMSSOP will define an FCB for any required global library
* if the user has not defined one. When the DCB is closed
* DMSOSC checks this flag to decide whether to clear the FCB
FCBSCLR EQU X'20' Do Filedef SCLEAR for this DDNAME
FCBLIBSV EQU X'10' Use RMS to call Tape Library Dataserver
FCBOSSIM EQU X'01' File needs true OS Sim limits
*
FCBBUFOF DS 1X - BUFFER OFFSET (0-99) FOR ANSI
FCBRSRV1 DS 1X RESERVED for future use
FCBLBOM DS 1X Saved first char of dataset name
FCBVCOU DS F Number of variable length records
* read into a block by QSAM
FCBFLAG4 DS 1X -
FCBTXTL EQU X'80' CONCATENATED TXTLIB DATA SET
FCBLRI EQU X'40' Logical Record Interface used
FCBBADDN EQU X'20' Fileid matches, DDN doesn't
* EQU X'10' RESERVED for future use
FCBDRFMT EQU X'08' Default RECFM filled in by OPEN
FCBDLREC EQU X'04' Default LRECL filled in by OPEN
FCDBLKZ EQU X'02' Default BLKSZ filled in by OPEN

```

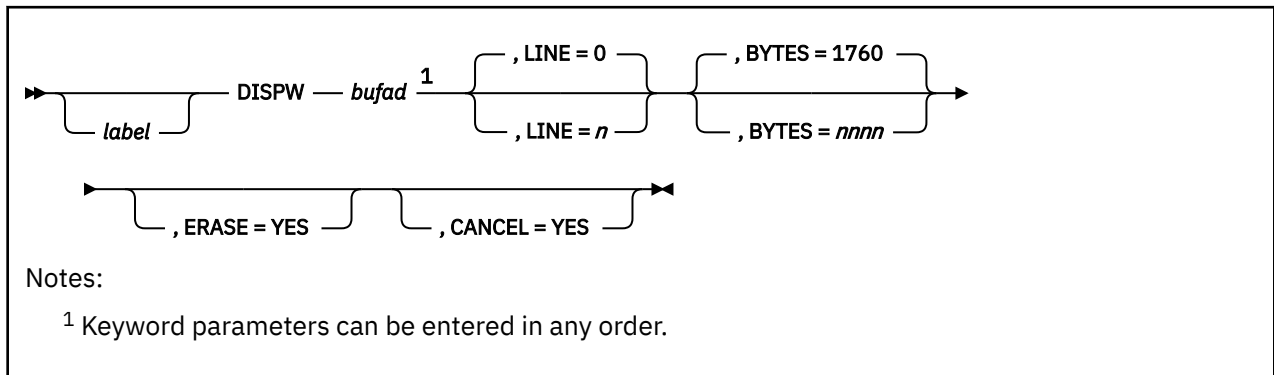
```

FCBDBUFO EQU X'01'      Default BUFOF filled in by OPEN
FCBRSRV2 DS 3X          RESERVED for future use
*
FCBAIC DS F            Number input recs in FST at OPEN
FCBPRIME DS CL4        PRIME TAPE ID ENTERED AT FILEDEF
FCBNEXTC DS A          Address of the next FCB in the
*                      concatenation, or zero if none
FCBFSEQ DS H           Tape File Sequence counter
DS H                 Reserved for future use
FCBFWVAL DS F          Fullword work field
FCBEND DS 0D -         END-OF FCB,JFCB,DEB,IOB BLOCKS
FCBENSIZ EQU (*-FCBSECT)/8 - SIZE OF FCB ENTRY, DOUBLEWORDS
* SPECIAL FIELDS FOR TAPE FILES ONLY ...
ORG FCBDSNAM
FCBTAPID DS CL4        TAPE IDENTIFICATION
FCBLABT DS 1X          TAPE LABEL TYPE
FCBOFF EQU X'00'       NO LABEL PROCESSING AT ALL
FCBBLP EQU X'01'       BYPASS LABELS - JUST POSITION TAPE
FCBSL EQU X'02'        IBM STANDARD LABELS
FCBUSER EQU X'04'       USER STANDARD LABELS
FCBSUL EQU FCBSL+FCBUSER IBM AND USER STANDARD LABELS
FCBNSL EQU X'08'       NONSTANDARD USER LABELS
FCBNSLMD EQU X'10'     NSL ROUTINE IS A MODULE
FCBNL EQU X'20'        NO LABELS
FCBAL EQU X'40'        ANSI LABELS
FCBAUL EQU FCBAL+FCBUSER ANSI AND USER STANDARD LABELS
FCBTPSW DS 1X          TAPE SWITCH
FCBLEAVE EQU X'80'     DO NOT REPOSITION TAPE FOR OPEN
FCBNOEOV EQU X'40'     DO NOT DO ANY EOV PROCESSING AT ALL
FCBFVLEV EQU X'20'     LEAVE specified on FEOV macro
FCBFVREW EQU X'10'     REWIND specified on FEOV macro
FCBFEOV EQU X'08'     FEOV was issued on the current
*                      tape volume. This flag is set
*                      on during FEOV and turned off
*                      either at CLOSE time or when a
*                      new tape volume is mounted.
FCBKEEP EQU X'04'      DISP KEEP specified in Filedef
*                      X'02'
*                      Reserved
FCBPOS DS 1H           POSITION PARAMETER
FCBNSLNM DS CL8        NSL ROUTINE NAME
ORG FCBMEMBR
FCBLABPT DS A          POINTER TO LABSECT
FCBBLKCT DS 1F         BLOCK COUNT FOR TAPE FILE
ORG FCBDSTYP+4
FCBIOOUT DS CL8 -      SPECIAL I/O COMMAND LIST
FCBIOBUF DS A -        A(DATA BUFFER)
FCBCONCR DS C -        CONSOLE COLOR CODE
FCBCONMS DS X -        CONSOLE MISCELLANEOUS INFO
FCBIOCNT DS H -        L'DATA BUFFER
*
* DATA EVENT CONTROL BLOCK
*
IHADECB DSECT
DECSDECB DS F -        EVENT CONTROL BLOCK
DECTYPE DS H -         TYPE OF I/O REQUEST
DECBRD EQU X'80' -     READ SF
DECBWR EQU X'20' -     WRITE SF
DECLNGTH DS H -        LENGTH OF KEY & DATA
DECDCBAD DS A -        V(DATA CONTROL BLOCK)
DECAREA DS A -         V(KEY & DATA, BUFFER)
DECIOBPT DS A -        V(IOB)
* BDAM EXTENSION
DECKYADR DS A -        V(KEY)
DECRECPT DS A -        V(BLOCK REFERENCE FIELD)
*
* SOME FREQUENTLY USED EQUATES
*
DDNAM EQU FCBSTYP -    FILETYPE = DATA SET NAME
BLK EQU X'10' -        RECFM=BLOCKED RECORDS
BS EQU X'20' -         MACRF=BSAM
DA EQU X'20' -         DSORG=DIRECT ACCESS
FXD EQU X'80' -        RECFM=FIXED LENGTH RECORDS
IS EQU X'80' -         DSORG=INDEXED SEQUENTIAL
LOC EQU X'08' -        MACRF=LOCATE MODE
MOV EQU X'10' -        MACRF=MOVE MODE
PS EQU X'40' -         DSORG=PHYSICAL SEQUENTIAL
POU EQU X'03' -        DSORG=PARTITIONED UNMOVEABLE
PO EQU X'02' -         DSORG=PARTITIONED ORGANIZATION
PREVIOUS EQU X'80' -   OFLGS=PREVIOUS I/O OPERATION
QS EQU X'40' -         MACRF=QSAM
UND EQU X'C0' -        RECFM=UNDEFIN FORMAT RECORDS
VAR EQU X'40' -        RECFM=VARIABLE LENGTH RECORDS

```

ANSID	EQU	X'20'	-	RECFM=VARIABLE LENGTH RECS (ANSI)
SPANNED	EQU	X'08'	-	RECFM=SPANNED

DISPW



Purpose

Use the DISPW macro to write data to a display screen. The CONSOLE macro supersedes the DISPW macro. DISPW is supported for compatibility only.

Required Parameters:

bufad

is the address of a buffer containing the data to be written to the display terminal.

Optional Parameters:

label

is an optional assembler label for the statement.

LINE=

is the number of the line, 0 to 23, on the display terminal that is to be written. Line number 0 is the default.

BYTES=

is the number of bytes (0 to 1760) to be written on the display terminal; 1760 bytes is the default.

ERASE=YES

specifies that the display screen is to be erased before the current data is written. The screen is erased regardless of the line or number of bytes to be displayed. Specifying ERASE=YES causes the screen to go into *MORE* status.

CANCEL=YES

causes the CANCEL operation to be performed. The output area is erased.

DMSEXS



Purpose



Attention: The use of this macro is not encouraged because it allows modification of internal data areas.

Use the DMSEXS, *execute in system mode*, macro to allow a routine executed with a user PSW key to execute a single instruction with a nucleus PSW key. The single instruction may be specified as the argument to the DMSEXS macro, and that instruction is executed with a nucleus PSW key. This macro can be used instead of two DMSKEY macros. Note that DMSEXS does not work from above the 16 MB line.

Parameters

Required Parameters:

op-code,operands

must be given as arguments to the DMSEXS macro.

For example, execution of the sequence,

```

USING  NUCON,0
DMSEXS OI,USERVLV,MYSWITCH

```

causes the OI instruction to be executed with a 0 protect key in the PSW. The instruction to be executed may be an EX instruction.

Optional Parameters:

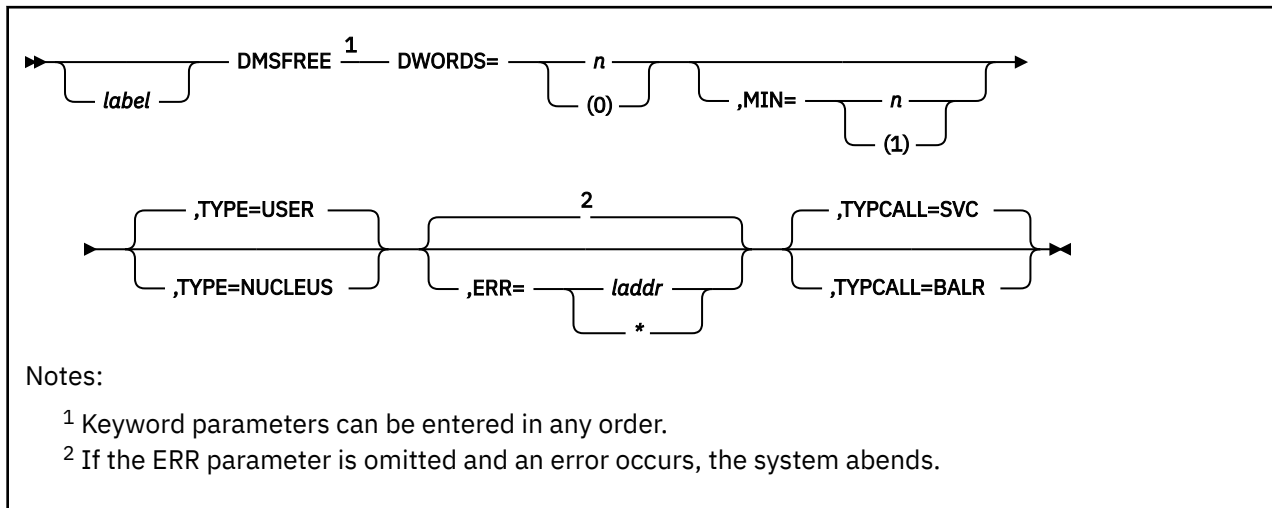
label

is an optional assembler label for the statement.

Usage Notes

1. Programs that modify or manipulate bits in CMS control blocks may hinder the operation of CMS causing it to function ineffectively.
2. Register 1 cannot be used in any way in the instruction being executed.
3. Whenever possible, CMS commands are executed with a user protect key. This protects the CMS nucleus in cases where there is an error in the system command that would otherwise destroy the nucleus. If the command must execute a single instruction or small group of instructions that modify nucleus storage, then the DMSKEY or DMSEXS macros are used so that the system PSW key is used for a short time.

DMSFREE



Purpose

Use the DMSFREE macro to allocate CMS free storage. For new programs and programs that are to support 31-bit addressing, use the CMSSTOR macro to manage and allocate free storage. DMSFREE continues to work, but only in 24-bit addressing mode (below 16 MB).

Parameters

Required Parameters:

DWORDS=

is the number of doublewords of free storage requested.

n

specifies the number of doublewords directly.

(0)

indicates that register 0 contains the number of doublewords requested. Do not (a) specify a register other than register 0 or (b) use an equated symbol to specify register 0.

Optional Parameters:

label

is an optional assembler label for the statement.

MIN=

indicates a variable request for free storage. If the exact number of doublewords indicated by DWORDS operand is not available, then the largest block of storage greater than or equal to the minimum is requested.

n

specifies the minimum number of doublewords of free storage directly.

(1)

indicates that the minimum is in register 1. Do not specify a register other than register 1.

Note that, when complete, DMSFREE stores in register 0 the actual amount of free storage allocated.

TYPE=

indicates the type of CMS storage requested, USER or NUCLEUS. USER is the default value.

ERR=

is the return address if an error occurs. *laddr* is any address that can be referred to in an LA (LOAD ADDRESS) instruction. The error return is taken if there is a macro coding error or if there is not enough free storage available to fill the request. If the asterisk (*) is specified for the return address, the error return is the same as a normal return. There is no default for this operand. **If it is omitted and an error occurs, the system abends.**

TYPICAL=

indicates how control is passed to DMSFREE.

SVC

uses SVC linkage to branch to DMSFREE. Routines that are not nucleus-resident must use SVC linkage (TYPICAL=SVC). This is the default value.

BALR

branches directly to DMSFREE. Because DMSFREE is a nucleus-resident routine, other nucleus-resident routines can branch directly to it.

Usage Notes

1. CMS does not support the AREA parameter. If specified, it is ignored.
2. When DMSFREE completes, CMS returns in register 0 the number of doublewords allocated and in register 1 the address of the allocated storage.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code**Meaning****1**

Insufficient storage space is available to satisfy the request for free storage. In the case of a variable request, even the minimum request could not be satisfied.

2

User storage pointers destroyed.

3

Nucleus storage pointers destroyed.

4

An invalid size was requested. This error exit is taken if the requested size is not greater than zero. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. (However, the latter error is not detected if DMSFREE is able to satisfy the maximum request.)

8 or greater

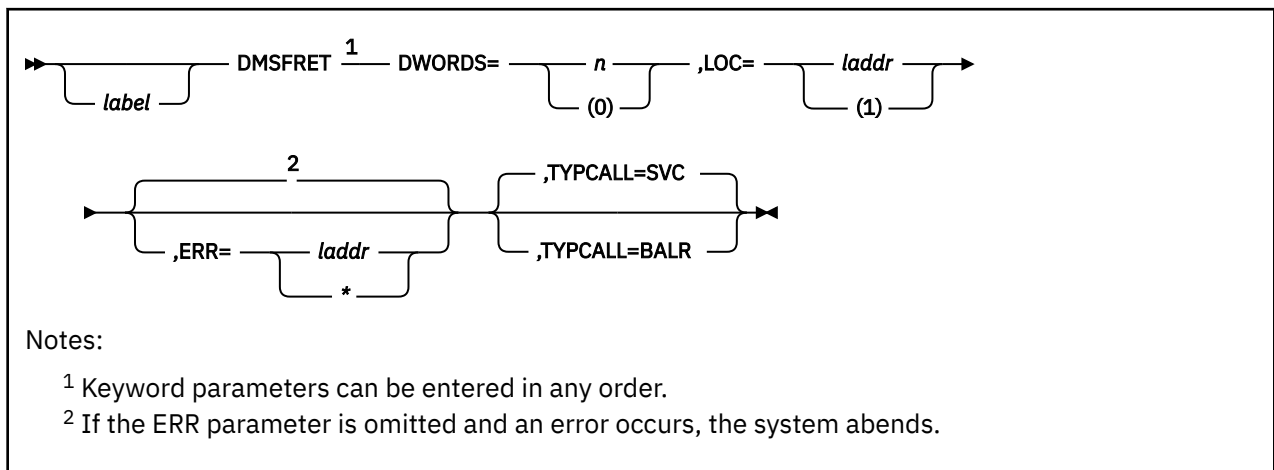
An unexpected and unexplained error has occurred in the free storage management routine.

DMSFRES

Purpose

The DMSFRES macro is treated as a no-op in CMS because the function is performed internally.

DMSFRET



Purpose

Use the DMSFRET macro to release free storage. For new programs, use the CMSSTOR macro to manage free storage. DMSFREE and DMSFRET continue to work, but from below 16 MB only.

Parameters

Required Parameters:

DWORDS=

is the number of doublewords of storage to be released.

n

specifies the number of doublewords directly.

(0)

indicates that register 0 contains the number of doublewords being released. Do not specify any register other than register 0. Also, do not express register 0 as an equated symbol.

LOC=

is the address of the block of storage being released.

laddr

specifies the address directly. *laddr* is any address that can be referred to in an LA (LOAD ADDRESS) instruction.

(1)

indicates the address is in register 1. Do not specify any register other than register 1.

Optional Parameters:

label

is an optional statement label.

ERR=

is the return address if any error occurs. The error return is taken if there is a macro coding error or if there is a problem returning the storage. **There is no default for this operand. If it is omitted and an error occurs, the system abends.**

laddr

is any address that can be referred to by an LA (LOAD ADDRESS) instruction.

(*)

passes control to the next sequential instruction.

TYPCALL=

indicates how control is passed to DMSFRET.

SVC

uses SVC linkage to branch to DMSFRET. Routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC). This is the default.

BALR

branches directly to DMSFRET. Because DMSFRET is a nucleus-resident routine, other nucleus-resident routines can branch directly to it.

Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code**Meaning****2**

User storage pointers destroyed.

3

Nucleus storage pointers destroyed.

5

An invalid size was passed to the DMSFRET macro. This error exit is taken if the specified length is not positive.

6

The block of storage that is being released was never allocated by DMSFREE. Such an error is detected if one of the following errors is found:

- The block crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS type storage.
- The block overlaps another block already on the free storage chain.

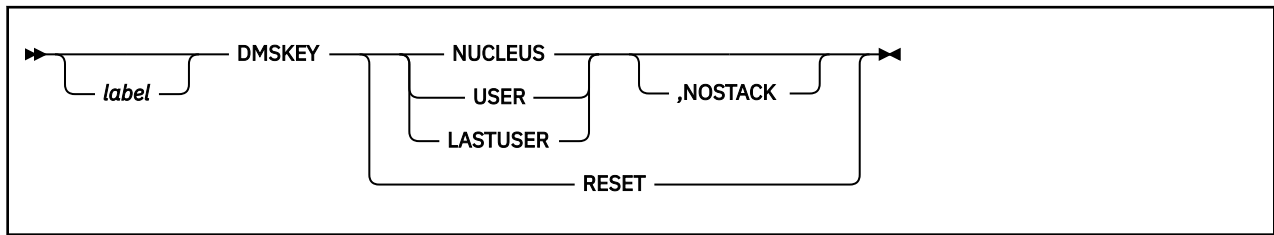
7

The address given for the block being released is not on a doubleword boundary.

8 or greater

An unexpected and unexplained error has occurred in the free storage management routine.

DMSKEY



Purpose



Attention: The use of this macro is not encouraged because it allows modification of internal data areas.

Use the DMSKEY macro to set nucleus protection on or off. DMSKEY works from below the 16 MB line only.

Parameters

Required Parameters:

NUCLEUS

places the nucleus storage protection key in the PSW and saves the old contents of the second byte of the PSW in a stack. This option allows the program to store data into system storage, which is ordinarily protected.

USER

places the user storage protection key in the PSW and saves the old contents of the second byte of the PSW in a stack. This option prevents the program from inadvertently modifying nucleus storage, which is protected.

LASTUSER

causes the SVC handler to trace back through its system save areas for the active user routine closest to the top of the stack. The storage key in effect for that routine is placed in the PSW. The old contents of the second byte of the PSW are saved in a stack.

This option should be used only by system routines that should enter a user exit routine. (OS/MVS macro simulation routines use this option when they want to enter a user-supplied exit routine. The exit routine is entered with the PSW key of the last user routine on the SVC system save area stack.)

RESET

changes the second byte of the PSW to the value at the top of the DMSKEY stack and removes it from the stack. This reverses the effect of the last DMSKEY NUCLEUS, DMSKEY USER, or DMSKEY LASTUSER request.

Note:

1. CMS requires that the DMSKEY stack be empty when a routine terminates. Therefore, for each DMSKEY NUCLEUS, DMSKEY USER, or DMSKEY LASTUSER macro that did not specify the NOSTACK option, you must issue a DMSKEY RESET macro. Otherwise, your program abnormally terminates.
2. Do not use the RESET option if you have **previously** specified NOSTACK.

Optional Parameters:

label

is an optional assembler label for the statement.

NOSTACK

this option can be used with NUCLEUS, USER, or LASTUSER to prevent the system from saving the second byte of the current PSW in a stack. If you specify NOSTACK, you do not need to issue DMSKEY RESET later.

Usage Notes

1. The DMSKEY key stack has a current maximum depth of seven for each routine. In this context, a routine is anything invoked by an SVC call.

►► IO ◄◄

Purpose

This macro maps OPSECT DSECT, which describes the fields that several programs use as parameter lists for I/O operations. The IO macro can only be used to map the area whose address is returned in general purpose register 8 after issuing the FILEDEF command with the AUXPROC option. For more information, see [z/VM: CMS Application Development Guide for Assembler](#).

Usage Notes

1. This macro is contained in DMSOM MACLIB. The CMSCB and IO macros map internal CMS data areas and are used with the TEOVEXIT macro and FILEDEF AUXPROC facility to monitor or modify I/O operations in CMS. For more information on FILEDEF AUXPROC, see [z/VM: CMS Application Development Guide for Assembler](#).
2. The IO macroinstruction maps the OPSECT DSECT as follows:

```

OPSECT    DS    0D
          ENTRY OPSECT

*
*   COMMANDER-IN-CHIEF OF ALL I/O OPERATION LISTS
*
PLIST     DS    0D
CMSOP     DS    CL8           I/O OPERATION COMMAND WORD
FILENAME  DS    CL8           FILE NAME
FILETYPE  DS    CL8           FILE TYPE
FILEMODE  DS    CL2           FILE MODE
          DS    H             NOT USED
FILEBUFF  DS    F             INPUT-OUTPUT BUFFER
FILEBYTE  DS    F             DATA COUNT
FILEFORM  DS    CL2           FILE FORMAT: FIXED/VARIABLE RECORDS
          DS    H             NOT USED
FILEREAD  DS    F             READ DATA COUNT
FILEITEM  DS    F             ITEM NUMBER
FILECOUT  DS    F             NUMBER OF ITEMS
FILEWPTR  DS    F             WRITE POINTER
FILERPTR  DS    F             READ POINTER
POINTERS  EQU    FILEITEM
AFST      EQU    FILEBUFF

*
IOAREA    EQU    FILEBUFF      BUFFER AREA LOCATION
IOLENGTH  EQU    FILEBYTE      BUFFER LENGTH

*
*   IMMEDIATE REGISTER SAVE ARE
*
SAVER14   DC    F'0'           TEMP R14 SAVE
SAVER15   DC    F'0'           TEMP R15 SAVE
SAVER0    DC    F'0'           TEMP R0 SAVE
SAVER1    DC    F'0'           TEMP R1 SAVE

*
CMSNAME    DC    CL8'FILE'      "DEFAULT FILE NAME"

*
*   CONSOLE PARAMETER LISTS
*
          DS    0D
*   READ CONSOLE
CONREAD    DC    CL8'WAITRD'     TERMINAL READ
CONRDBUF   DC    &T.(CMNDLINE)   ADDRESS OF INPUT BUFFER
CONRDCOD   DC    C'U'           TRANSLATE CODE
          DC    X'00'
CONRDCNT   DC    AL2(CBUFMAX)     DATA BYTE COUNT
          DC    F'0'             RESERVED

*
*   CONSOLE WAIT LIST
WAITLIST   DS    0F

```



```

      DC      CL8'CONWAIT'
*
*   WRITE CONSOLE
CONWRITE DS    0F
      DC      CL8'TYPLIN'
CONWRBUF DC     A(0)          LOCATION OF MESSAGE TEXT
CONWRCOD DC     C'B'          COLOR CODE
      DC      X'00'
CONWRCNT DC     AL2(0)        LENGTH OF MESSAGE TEXT
*
*   WAIT PARAMETER LIST
*
WAITLST DS     0F
      DC      CL8'WAIT'
WAITDEV DC     CL4'CON1'
      DC      F'0'
      DC      F'0'
*
*   INTERACTIVE CONSOLE COMMUNICATION CHANNEL PROGRAM
*
CONPCCW CCW     1,0,X'60',0    WRITE FOR APL ASCII PROMPT
CONCCWS CCW     0,0,X'60',0    NORMAL READ OR WRITE
      CCW      3,0,X'20',1    NOP TO GET CE AND DE TOGETHER
*
*   READER PARAMETER LIST
*
      DS      0F
READLST DC     CL8'CARDRD'
RDFLAG  DC     X'00'          FLAG BYTE
      DC     XL3'000000'      OLD BUFFER FIELD
RDCCW   DC     H'0'          CCW BYTE COUNT
RDCOUNT DC     H'0'          BYTES ACTUALLY READ
RDBUFF  DC     A(0)          BUFFER ADDRESS
      DC     XL4'00000000'    RESERVED
RDFENCE DC     8X'FF'        FENCE
*
*   CARD PUNCH PARAMETER LIST
*
PUNCHLST DS    0F
      DC     CL8'CARDPH'
PUNFLAG  DC     X'00'          FLAG BYTE
      DC     XL3'000000'      OLD BUFFER FIELD
PUNCOUNT DC     A(0)          PUNCH CCW COUT
PUNBUFF  DC     A(0)          PUNCH BUFFER ADDRESS
      DC     XL4'00000000'    RESERVED
PUNFENCE DC     8X'FF'        FENCE
*
*   PRINTER PARAMETER LIST
*
PRINTLST DS    0F
      DC     CL8'PRINTR'
PRBUF    DC     A(0)          PRINTER BUFFER ADDRESS
PRTRC    DC     C'0'          TRC BYTE
PRFLGS1  DC     X'0'          PRINT FLAGS
PRXPLIST EQU   X'80'          EXTENDED PLIST IN USE
PR3800   EQU   X'08'          VIRTUAL PRINTER IS A 3800
PRTRCINP EQU   X'04'          PLIST TRC BYTE IS VALID
PRTRCIND EQU   X'02'          TRC IN DATA
PRNOASA  EQU   X'01'          CC BYTE NOT ASA
PRLEN    DC     H'0'          PRINT DATA LENGTH
* ----- EXTENDED PLIST -----
PRFLGS2  DC     X'0'          PRINT FLAGS
PRCCINP  EQU   X'04'          CONTROL CHARACTER IN PLIST
PRCMSDEV EQU   X'02'          CMSDEV INFORMATION IN PLIST
PRFORM   EQU   X'01'          0: FORM=BUFFER, 1: FORM=LIST
PRCC     DC     X'0'          CONTROL CHARACTER
PRDEVC   DC     X'0'          PRINTER DEVICE CLASS
PRDEVT   DC     X'0'          PRINTER DEVICE TYPE
PRCCW    DC     A(0)          CCW BUFFER ADDRESS
PRCNT    DC     H'0'          PRINT RECORD COUNT
      DS      H              RESERVED
PRINTEND EQU   *              END OF PRINTER PLIST
*
*   TAPEIO PARAMETER LIST
*
TAPELIST DS    0F
      DC     CL8'TAPEIO'
TAPEOPER DC     CL8' '          TAPE OPERATION COMMAND
TAPEDEV  DC     CL4'TAP1'       TAPE SYMBOLIC DEVICE
TAPERFMT DC     X'00'          RECORDING FORMAT
TAPEMASK EQU   TAPERFMT,1,C'X'  Old label for TAPERFMT
      DC     XL3'000000'      OLD BUFFER LOCATION

```

```

TAPESIZE DC F'0'
TAPECOUT DC F'0'
TAPEBUFF DC AL4(0)
TAPEMRFT DC XL1'00'
TAPEPORT DC XL1'00'
TAPERESV DC XL2'0000'
TAPFENCE DC 8X'FF'
*
* CLOSE OUT DEVICE DEPENDENT DATA SET ON UNIT RECORD EQUIPMENT
*
CLOSIO DS 0F
DC CL8'CLOSIO' OPERATION
CLOSIO DV DC CL8' ' DEVICE TYPE
DC 4X'FF'
DC 6D'0' - UNUSED
*
*
* STORAGE FOR EXEC BOOTSTRAP:
EXLEVEL DC F'0' EXEC "LEVEL"
EXF1 DC F'1' (FOLLOWS EXLEVEL)
DS F RESERVED
DS F RESERVED
EXGLOBAL DC F'0' ADDRESS OF EXEC GLOBAL AREA
DC F'0' - UNUSED
*
* STORAGE FOR OS MACRO SIMULATION ROUTINES
FCBIO DC A(0) - ADDRESS OF LAST FCB USED DURING I/O
OSIOTYPE DC X'DD' - OS ACCESS METHOD TYPE
*
*
* REGISTER SAVE AREA AND WORK AREA FOR DMSEXQ
EXQWORK DS 0D
EXQSAVE DS 4F SAVEAREA FOR R14-R1
EXQOLD2 DS 11F SAVEAREA FOR R2-R12
EXQOLD13 DS 1F SAVEAREA FOR R13
EXQCMD DC CL8'ESTATE ' USED AS PLIST FOR STATE CMD
EXQNAME DS CL8 EXECNAME PASSED IN PARMLIST
EXQTYPE DS CL8 EXECTYPE PASSED IN PARMLIST
EXQMODE DC CL2'* ' FILE MODE FOR STATE COMMAND
DS CL2
EXQFST DS CL4 FST ADDRESS FROM STATE
EXQEND DC 8X'FF' FENCE FOR STATE
EXQFLAG DS X FLAG FOR OPTIONS
SAVEBYTE DS X SAVE MESSAGE FLAG SETTING
DS 2X UNUSED
EXQPTR DS 1F Data address for STRUCTUR macro
EXQKEYFN DS CL8 Key used for STRUCTUR macro
EXQKEYFT DS CL8 Key used for STRUCTUR macro
EXQSTRCT STRUCTUR OFIND,MF=L Plist area for STRUCTUR macro
* End of DMSEXQ work area
CONQSAVE DS 0D
DS 18F QUEUE MANAGER SAVEAREA
*
* QUEUE MANAGER PARAMETER LIST
*
DMSQPLST DSECT
*
* Console Input Queue
*
CMSQBLK DS 0D
QNXTBLK DC A(0) Fwd ptr - next queue block
QNAME DC CL8'CMS ' Name of this queue
QFLAGS DC X'90' Queue flag byte
QCLFLAG EQU X'80' Queue class - input or output
QCNFLAG EQU X'40' Queue connection specified
QCNCFLAG EQU X'20' Class of the connected queue
QXAFLAG EQU X'10' Queue exit address specified
QMLFLAG EQU X'08' Queue message limit specified
*
DC XL3'00' Reserved
QNAME DC CL8' Connected queue name
QXADDR DC V(DMSCITIM) Exit routine address
QMLIMIT DC F'0' Maximum number of messages
QMCOUNT DC F'0' Number of messages queued
QMHEAD DC A(0) Head of message queue
QMTAIL DC A(0) Tail of message queue
DC XL20'00' Reserved
*
* LINERD PARAMETER LIST
*
DMSLRDP CSECT
*

```

```

*   Fields required by LINERD
*
LNENUM   DC    F'0'           Line number of the data read
COLNUM   DC    F'0'           Column number of the data read
*
*   Console input buffer
*
CONINBLK DS    0D             Reserved
          DC    A(0)
CONINCDE DC    XL1'0A'        Flags and command code
CONRD     EQU   X'0A'         Read command code
CONRDINV  EQU   X'0E'         Special read command code, to
*                               inhibit display of data read
CONATTN   EQU   X'40'         Attention read
CONWRCR   EQU   X'09'         Write with carriage return
CONWRNCR  EQU   X'01'         Write with no carriage return
CBUFMAX   EQU   X'FF'         Maximum console read length
CONINLEN  DC    AL1(255)      Length to be read from console
CONINBUF  DS    CL255         Input line

```


Text specified on the LINEDIT macro is edited so that multiple blanks appear as a single blank, and a period is placed at the end of the line; for example:

```
LINEDIT TEXT='IT ISN'T READY'
```

results in the display:

```
IT ISN'T READY.
```

TEXTA=

specifies the address of the message text. Use the TEXTA operand when you want to display a line that is contained in a buffer. You may specify either a symbolic address or use register notation. Acceptable values are:

label

the symbolic address of the message text.

(reg)

a register containing the address of the message text.

The first byte at the address specified must contain the length of the message text, for example:

```
LINEDIT TEXTA=MESSAGE
      .
      .
MESSAGE DC X'16'
         DC CL22'THIS IS A LINE OF TEXT'
```

If you use register notation with either the standard or list forms of the macro, the code generated is not reentrant. To suppress the MNOTE that informs you that code is not reentrant, use the RENT=NO operand.

DOT=

specifies whether a period is to be placed at the end of the line. Acceptable values are:

YES

specifies that you do want a period to be placed at the end of the line. This is the default value.

NO

specifies that you do not want a period placed at the end of the message text. For example, if you code:

```
LINEDIT TEXT='THE DINOSAUR FAMILY SAYS HI! ',DOT=NO
```

the line is displayed as:

```
THE DINOSAUR FAMILY SAYS HI!
```

COMP=

specifies whether multiple blanks are to be removed from the line. Acceptable values are:

YES

specifies that you want multiple blanks to be removed from the line. This is the default value.

If COMP=YES, not only are all multiple blanks reduced to single blanks, but any leading blanks are removed as well.

NO

specifies that you want to display multiple blanks within your message text.

For example, if you code:

```
LINEDIT TEXT='TOTAL    5 ',COMP=NO
```

the line is displayed as:

```
TOTAL      5.
```

SUB=*sublist*

specifies a substitution list describing the conversions to be performed on the line.

Use the SUB operand to specify the type of substitution to be performed on those portions of the message that contain periods. For each set of periods, you must specify the type of substitution and the value to be substituted or its address. Acceptable values are:

(type, (value,length))

specifies the type of data, its address, and the length of the substitution.

(type,value)

specifies a number used to retrieve the substitution information from the repository.

If you specify a length, you must enclose the value and length in parentheses. Otherwise, do not enclose the value in parentheses.

You can specify both the value and length using register notation. When you specify the length, it is interpreted to be the length of the input field, except when used with the HEX, HEXA, HEX4A, DEC and DECA parameters. For these parameters, the length represents the length of the converted result. Following are the possible values of *type*.

HEX, (reg)

converts the value in the specified register to graphic hexadecimal format and substitutes it in the message text. If you code fewer than 8 consecutive periods in the message text, then leading digits are truncated; leading zeros are not suppressed.

For example, if register 3 contains the value C0031FC8, then the macroinstruction:

```
LINEDIT TEXT='VALUE = ...',SUB=(HEX,(3))
```

results in the display:

```
VALUE = FC8.
```

HEX,expression

converts the given expression to graphic hexadecimal format and substitutes it in the message text. The expression may be a symbolic address or symbol equate; it is evaluated by means of a LOAD ADDRESS (LA) instruction. For example, if your program has a label BUFF1, the line:

```
LINEDIT TEXT='BUFFER IS LOCATED AT .....',SUB=(HEX,BUFF1)
```

might result in the display:

```
BUFFER IS LOCATED AT 0201AC.
```

If you code fewer than 8 periods in the message text, leading digits are truncated; leading zeros are not suppressed.

DEC, (reg)

converts the value in the specified register into graphic decimal format and substitutes it in the message text. Leading zeros are suppressed. If the number is negative, a leading minus sign is inserted. For example, if register 3 contains the decimal value 10,345, then the macroinstruction:

```
LINEDIT TEXT='REG 3 = .....',SUB=(DEC,(3))
```

results in the line:

```
REG 3 = 10345.
```

DEC,expression

converts the given expression to graphic decimal format and substitutes it in the message text. The expression may be a symbolic label in your program or a symbol equate. For example, if your program contains the statement:

```
VALUE EQU 2003
```

then the macroinstruction:

```
LINEDIT TEXT='VALUE IS .....',SUB=(DEC,VALUE+5)
```

results in the display:

```
VALUE IS 2008.
```

HEXA,address

converts the fullword at the specified address to graphic hexadecimal format and substitutes it in the message text. If you code fewer than 8 periods in the message text, leading digits are truncated; leading zeros are not removed. For example, if you code:

```
LINEDIT TEXT='HEX VALUE IS .....',SUB=(HEXA,CODE)
```

then the last 5 hexadecimal digits of the fullword at the label CODE are substituted into the message text.

HEXA,(reg)

converts the fullword at the address indicated in the specified register into graphic hexadecimal format and substitutes it in the message text. For example, if you code:

```
INEDIT TEXT='REGISTER 5 -> .....',SUB=(HEXA,(5))
```

then the last 6 hexadecimal digits of the fullword whose address is in register 5 are substituted in the message text.

If you code fewer than 8 digits, leading digits are truncated; leading zeros are not suppressed.

DECA,address

converts the fullword at the specified address to graphic decimal format. Leading zeros are suppressed; if the number is negative, a minus sign is inserted. For example, if you code:

```
LINEDIT TEXT='COUNT = .....',SUB=(DECA,COUNT)
```

then the fullword at the location COUNT is converted to graphic decimal format and substituted in the message text.

DECA,(reg)

converts the fullword at the address specified in the indicated register into graphic decimal format and substitutes it in the message text. For example:

```
LINEDIT TEXT='SUM = .....',SUB=(DECA,(3))
```

causes the value in the fullword whose address is in register 3 to be displayed in graphic decimal format.

HEX4A,address

converts the data at the specified address into graphic hexadecimal format, and inserts a blank character following every 4 bytes (8 characters of output). The data to be converted does not have to be on a fullword boundary. When you code periods in the message text for substitution, you must code sufficient periods to allow for the blanks. Thus, to display 8 bytes of information (16 hexadecimal digits), you must code 17 periods in the message text.

For example, to display 7 bytes of hexadecimal data beginning at the location STOR in your program, you could code:

```
LINEDIT TEXT='STOR: .....',SUB=(HEX4A,STOR)
```

This might result in a display:

```
STOR: 0A23F115 78ACFE
```

Note that 15 periods were coded in the message text, to allow for the blank following the first 4 bytes displayed.

HEX4A,(reg)

converts the data at the address indicated in the specified register into graphic hexadecimal format and inserts a blank character following every 4 bytes displayed (8 characters of output).

When you code the message text for substitution, you must code sufficient periods to allow for the blank characters to be inserted.

For example, the line:

```
LINEDIT TEXT='BUFFER: .....',SUB=(HEX4A,(6))
```

results in the display of the first 9 bytes at the address in register 6, in the format:

```
hhhhhhhh hhhhhhhh hh
```

CHARA,address

substitutes the character data at the specified address into the message text. For example:

```
LINEDIT TEXT='NAME IS .....',SUB=(CHARA,NAME)
```

causes the 10 characters at location NAME to be substituted into the message text. Multiple blanks are removed.

CHARA,(reg)

substitutes the character data at the address indicated in the specified register into the message text. For example:

```
LINEDIT TEXT='CODE IS ....',SUB=(CHARA,(7))
```

the first 4 characters at the address indicated in register 7 are substituted in the message line.

CHAR8A,address

substitutes the character data at the specified address into the message text, and inserts a blank character following each 8 characters of output.

When you code the message text, you must code enough periods to allow for the blanks that are substituted.

This substitution list is convenient for displaying CMS parameter lists. For example, to display a file ID in an FSCB, you might code

```
LINEDIT TEXT='FILEID IS .....',
SUB=(CHAR8A,OUTFILE+8)
```

where OUTFILE is the label on an FSCB macro. If the file ID for this file were TEST OUTPUT A1, then the LINEDIT macroinstruction would result in the display:

```
FILEID IS TEST OUTPUT A1.
```

In the final edited line, multiple blanks are reduced to a single blank.

CHAR8A,(reg)

substitutes the character data at the address indicated in the specified register and inserts a blank character following each 8 characters of output.

When you code the message text, you must include sufficient periods to allow for the blanks. For example:


```
LINEDIT TEXT='PLIST: .....',
      SUB=(CHAR8A,(7))
```

results in a display of 4 doublewords of character data, beginning at the address indicated in register 7.

DISP=

specifies how the edited line is to be used. When DISP is not coded, the message text is displayed at the terminal. Specify DISP as:

TYPE

specifies that the message is to be displayed on the terminal. This is the default disposition.

NONE

specifies that no output occurs. This option is useful with the BUFFA operand.

SIO

specifies that the message is to be displayed, at the terminal, using Start I/O instead of the usual CMS I/O services. When this option is used, HT (Halt Type) has no effect and the text may be displayed out of chronological order since lines are not stacked in the console buffer.

This option is not intended for routine use. It should be used only when severe errors occur (such as destroyed free storage pointers) because the path through CMS is kept to a minimum and additional storage is not required.

PRINT

specifies that the line is to be printed on the virtual printer. The first character of the line is interpreted as a carriage control character and as such does not appear on the printed output. (See the discussion of the PRINTL macro for a list of valid ASA control characters.) The maximum line size is 130 characters including the ASA character.

When the macro completes, register 15 contains a 2 if a channel 12 punch was sensed, or a 3 if a channel 9 punch was sensed. The location on the page being printed and the corresponding channel punch is defined by the current forms control buffer image being used. For more information on how to specify the forms control buffer image for a virtual spooled printer, see the LOADVFCB and SPOOL commands in the *z/VM: CP Commands and Utilities Reference*. If you use a virtual spooled 3800, refer to the CMS command SETPRT.

When the channel 9 or channel 12 punch is sensed, the write operation terminates after carriage spacing, but before writing the line. If you want to write the line without additional space, you must modify the carriage control character in the buffer to a code that writes without spacing (ASA code + or machine code 01).

You must issue the CP CLOSE command or the CP SPOOL PRT CLOSE command to close the virtual printer file. Issue the command either from your program (using an SVC 202 instruction or a LINEDIT macroinstruction) or from the CMS environment after your program completes execution. The printer is automatically closed when you log off or when you use the CMS PRINT command.

Note: If an error occurs and DISP=PRINT is specified, register 15 contains one of the return codes specified in the Return Codes section of the PRINTL macro.

CPCOMM

specifies that the line is to be passed to CP to be executed as a CP command. For example:

```
LINEDIT TEXT='QUERY USERS',DISP=CPCOMM,DOT=NO
```

results in the CP command line being passed to CP and executed. On return, register 15 contains the return code from the CP command that was executed.

Note: When using the DISP=CPCOMM operand, specify DOT=NO (the default is YES).

ERRMSG

specifies that the line is to be checked to see if it qualifies for error message editing. If it does, it is displayed as an error message rather than as a regular line.

The standard header format of VM error messages is:

```
xxxmmnnns
```

where:

- xxxmmm is the name of the module issuing the message
- nnn is the message number
- s is the severity code

You can code whatever you want for the first 9 characters of the code when you write error messages for your programs, but the tenth character must specify one of the following VM message types:

Code
Message Type

I
Information

W
Warning

E
Error

The line is displayed according to the CP EMSG setting. If EMSG is set to:

- ON - the entire message is displayed
- TEXT - only the message portion is displayed
- CODE - only the 10-character code is displayed.

BUFFA=

specifies the address of the buffer to which the line is to be copied. The message is copied into the indicated buffer, and is used as specified in the DISP operand. Acceptable values are:

addr
specifies the address of the buffer to which the line is to be copied.

(reg)
specifies the register containing the address of the buffer to which the line is to be copied.

If you use register notation to indicate the buffer address, the code generated is not reentrant. To suppress the MNOTE that informs you that code is not reentrant, use the RENT=NO operand.

When the text is copied into the buffer, the length of the message text is inserted into the first byte of the buffer, and the remainder of the text is inserted in subsequent bytes.

MAXSUBS=number

specifies the maximum number of substitutions (MAXSUBS is used with the list form of the macro).

Use the MAXSUBS operand when you code the list format (MF=L) of the LINEDIT macroinstruction. *number* specifies the maximum number of substitutions that is made when the execute form of the macro is used.

MF=

specifies the macro format when you want to code list and execute forms when you write reentrant programs. Acceptable values are:

I (Standard Format)

generates an inline operand list for the LINEDIT macroinstruction, and calls the routine that displays the message. This is the default. It generates reentrant code, except under the following circumstances:

- When you specify more than one substitution list

- When you use register notation with the TEXTA or BUFFA operands.

L (List Format)

generates a parameter list to be filled in when the execute form of the macro is used.

The size of the area reserved depends upon the number of substitutions to be made, which you can specify with the MAXSUBS operand. For example:

```
LINEDIT MF=L,MAXSUBS=5
```

reserves space for a parameter list that may hold up to five substitution lists. This list may be used by several LINEDIT macroinstructions.

(E,addr) (Execute Format)

generates code to fill in the parameter list at the specified address, and calls the routine that displays the message text.

The address specified (either a symbolic address or in register notation) indicates the location of the list form of the macro. The following example shows how you might use the list and execute formats of the LINEDIT macro to write reentrant code:

```
WRITETOT LINEDIT TEXT='SUBTOTAL ..... TOTAL .....',
          SUB=(DEC,(4),DEC,(5)),MF=(E,LINELIST)
:
:
LINELIST LINEDIT MF=L,MAXSUBS=6
```

When the execute format of the LINEDIT macroinstruction is used, the parameter list for the message is built at label LINELIST, where the list form of the macro was coded.

RENT=

specifies whether reentrant code must be generated.

Use the RENT operand when you are going to use the standard format of the LINEDIT macroinstruction and you do not care whether the code that is generated is reentrant. Acceptable values are:

YES

specifies that reentrant code must be generated. This is the default value.

When RENT=YES is in effect, the LINEDIT macro expansion issues an MNOTE message indicating that nonreentrant code is being generated. This occurs when you use the standard format of the macroinstruction and you specify one of the following:

- TEXTA=(reg)
- BUFFA=(reg)
- More than one substitution pair.

NO

specifies that reentrant code is not generated.

If you do not care whether the code is reentrant, and you do not wish to have the MNOTE appear, code RENT=NO. The RENT=NO coding merely suppresses the MNOTE statement; it has no effect on the expansion of the LINEDIT macroinstruction.

Usage Notes

1. You should never use registers 0, 1, or 15 as address registers when you code the LINEDIT macroinstruction; these registers are used by the macro.
2. When message text for the LINEDIT macroinstruction contains two or more consecutive periods, it indicates that a substitution is to be performed on that portion of the message. The number of periods

you code indicates the number of characters that you want to appear as output. To indicate what values are to replace the periods, code a substitution list using the SUB operand.

3. When you use the standard (default) form of the LINEDIT macroinstruction, reentrant code is produced, except when you specify more than one substitution list, or when you use register notation to indicate an address on the TEXTA or BUFFA operands. When any of these conditions occur, an MNOTE message is produced, indicating that the code is not reentrant.

If you do not care whether the code is reentrant, you can specify the RENT=NO operand to suppress the MNOTE message. Otherwise, you can use the list and execute forms of the macro to write reentrant code (see *MF parameter*).

4. When the macro completes, register 15 may contain a return code of 2 or 3, indicating that a channel 9 or channel 12 punch was sensed. You can use these codes to determine whether the end of the page is near (channel 9), or if the end of the page has been reached (channel 12). You might want to check for these codes if you want particular information at the bottom of each page being printed.
5. The length of the argument being substituted is determined by the number of periods in the message text. The number of periods indicates the size of the output field, and indirectly determines the size of the input data area.

For hexadecimal and decimal substitutions, the input data is truncated on the left. To ensure that a decimal number is never truncated, you can code 10 periods (11 for negative numbers) in the message text where it is substituted. For hexadecimal data, code 8 periods to ensure that no characters are truncated when a fullword is substituted.

When you are coding substitution lists with the CHARA, CHAR8A, and HEX4A options, however, you can specify the length of the input data field. You must code the SUB operand as follows:

```
SUB=(type,(address,length))
```

Both address and length may be specified using register notation. For example:

```
SUB=(HEX4A,(LOC,(4)))
```

shows that the characters at location LOC are substituted into the message text; the number of characters is determined by the value contained in register 4, but it cannot be larger than the number of periods coded in the message text.

You can use this method in the special case where only one character is to be substituted. Because you must always code at least two periods to indicate that substitution is to be performed, you can code two periods and specify a length of one, as follows:

```
LINEDIT TEXT='INVALID MODE LETTER ..',SUB=(CHARA,(PLIST+24,1))
```

6. When you want to make several substitutions in the same line, you must enter a substitution list for each set of periods in the message text. For example:

```
LINEDIT TEXT='VALUES ARE ..... and .....',
SUB=(DEC,(3),HEXA,LOC)
```

might generate a line as follows:

```
VALUES ARE -45 AND FFE3C2.
```

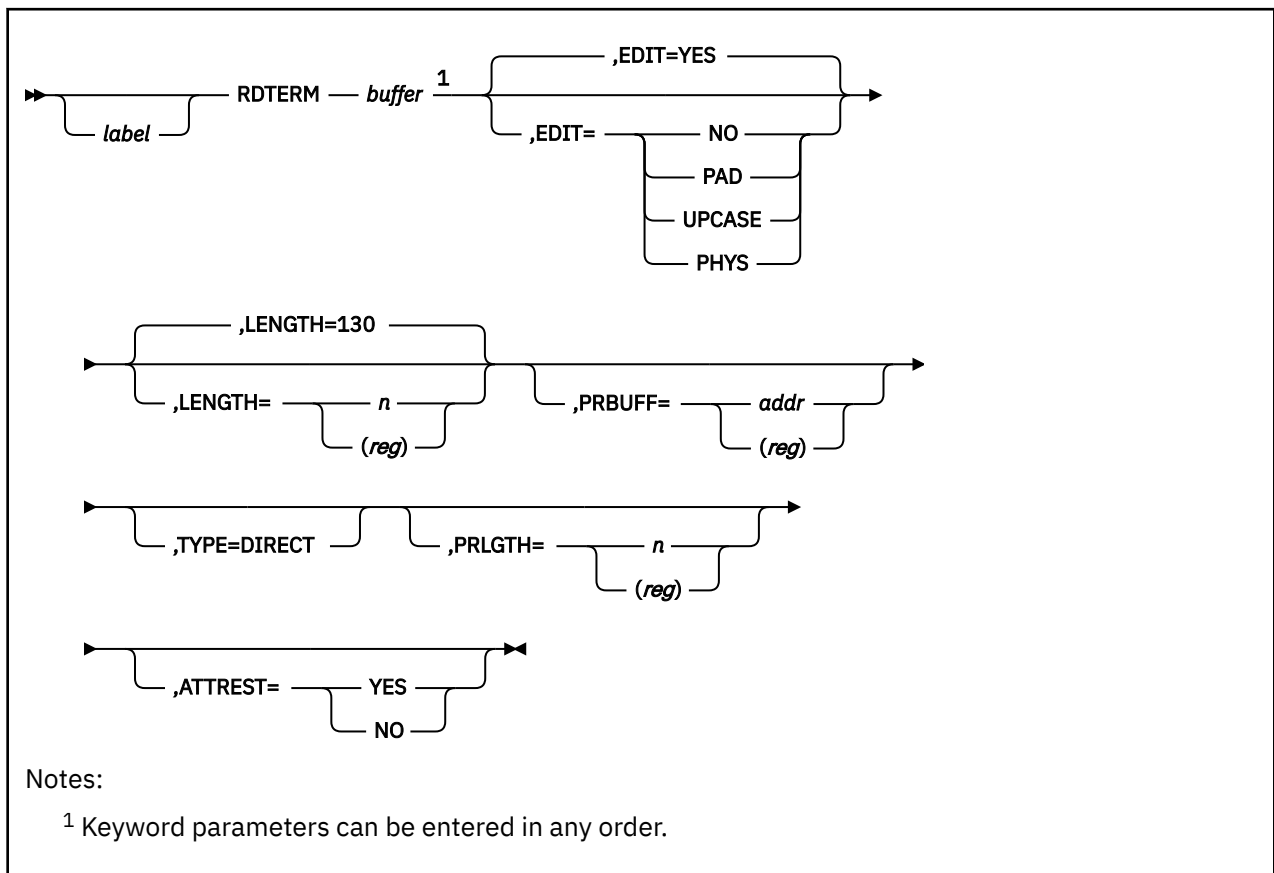
You should remember that if you are using the standard form of the macroinstruction, and you want to perform more than one substitution in a single line, the LINEDIT macro does not generate reentrant code. If you code RENT=NO on the macro line, then you do not receive the MNOTE message indicating that the code is not reentrant. If you want reentrant code, you must use the list and execute forms of the macroinstruction.

Return Codes

If an error occurs and DISP=PRINT is specified, register 15 contains one of the return codes specified in [“Return Codes” on page 348](#) the PRINTL macro.

If an error occurs DISP=CPCOMM is specified, register 15 contains the return code from the CP command executed.

RDTERM



Purpose

Use the RDTERM macroinstruction to read a line from the terminal into an I/O buffer. For new programs and programs that support 31-bit addressing, use the LINERD macro. RDTERM continues to work, but only below 16 MB.

Parameters

Required Parameters:

buffer

specifies the address of a buffer where the line is read. The buffer is assumed to be 130 bytes long, unless EDIT=PHYS is specified. Specify the buffer address as:

bufaddr

the symbolic address of the buffer.

(reg)

a register (2-12) containing the address of the buffer.

Optional Parameters:

label

is an optional assembler label for the statement.

EDIT=

specifies the type of editing, if any, to be performed on the input line.

YES

indicates both padding and translation to uppercase. YES is the default.

NO

indicates that a logical line is to be read and no editing is to be done.

PAD

requests that the input line be padded with blanks to the length specified.

UPCASE

requests that the line be translated to uppercase.

PHYS

indicates that a physical line is to be read. When you specify EDIT=PHYS, you may also enter the LENGTH and ATTREST=NO operands. This option causes the input line to be translated using the user translation table.

LENGTH=

specifies the length of the buffer. If not specified, 130 is assumed. The maximum length is 2030 bytes. The length may be specified only if EDIT=PHYS (see Usage Note 2).

n

specifies a self-defining term indicating the length of the buffer.

(reg)

specifies a register (2-12) containing the length of the buffer.

PRBUFF=

specifies the address of a buffer containing the prompt data. The length of the prompt data to be written is specified by the PRLGTH parameter. If the PRLGTH parameter is specified, but the PRBUFF parameter is not, the prompt information is assumed to reside in the read buffer. Specify PRBUFF as follows:

addr

the symbolic address of the buffer.

(reg)

a register (2-12) containing the length of the buffer.

TYPE=DIRECT

indicates that the input line is to be read directly from the virtual machine console. The terminal input buffer and the program stack are bypassed.

PRLGTH=

specifies the length of the prompt information to be written before the read. The prompt information is written with no carriage return. The prompt information is written from the user's read data buffer or from the buffer specified by the PRBUFF parameter. Specify PRLGTH as:

n

specifies a self-defining term indicating the length of the buffer.

(reg)

specifies a register (2-12) containing the length of the buffer.

ATTREST=YES|NO

specifies whether an attention interrupt during a read should result in a restart of the read operation. (See Usage Note 2.)

Usage Notes

1. When the macro completes, register 0 contains the number of characters read.

2. Use the ATTREST=NO and LENGTH operands only when you are reading physical lines (EDIT=PHYS). When ATTREST=NO, an attention interrupt during a read operation signals the end of the line and does not result in a restart of the read. These operands are used primarily in writing VS APL programs.
Note: If you are using a typewriter terminal, and specify ATTREST=NO, CMS restarts a read when an attention is generated on a null line. The only way to end the read is by pressing the carriage return.
3. The PRBUFF and PRLGTH operands are intended for use with TTY type devices. The maximum PRLGTH is 1760 characters. If the PRBUFF option is used, an 'XON' control character will not be transmitted to TTY devices.
4. If the prompt parameters are used with EDIT=PHYS, the read buffer may not be used for the prompt data because the read buffer is cleared prior to the execution of the function.
5. In CMS fullscreen, when a part of a field from the CMS virtual screen is modified, the entire field is returned as a modified field. For more information on CMS fullscreen, see the LINERD macro.
6. Any translation done on the input buffer that contains both SBCS and DBCS data will only occur on the SBCS portions of the data provided that the display is capable of supporting mixed DBCS.
7. If truncation occurs because the data being read in is longer than the input buffer, and the truncation occurs within a mixed DBCS string, then adjustments will be made to validate the truncated string.

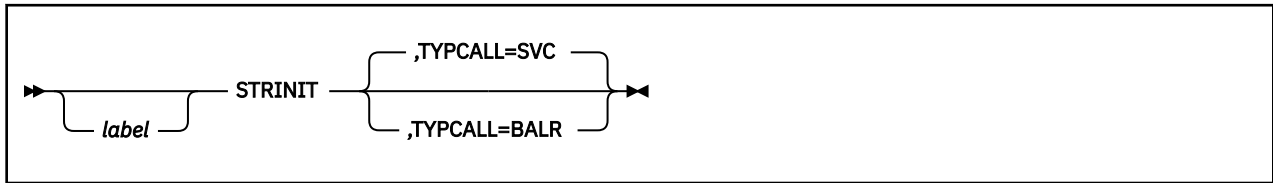
Return Codes

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
------	---------

2	Invalid parameter.
4	Read was terminated by an attention signal (possible only when ATTREST=NO).

STRINIT



Purpose

Use the STRINIT macro to release free storage obtained by the GETMAIN macro.

Parameters

Optional Parameters:

label

is an optional assembler label for the statement.

TYPCALL=

indicates how control is passed to the STRINIT macro.

SVC

uses SVC linkage to branch to the STRINIT routine. Routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC). If no operands are specified, the default is TYPCALL=SVC.

BALR

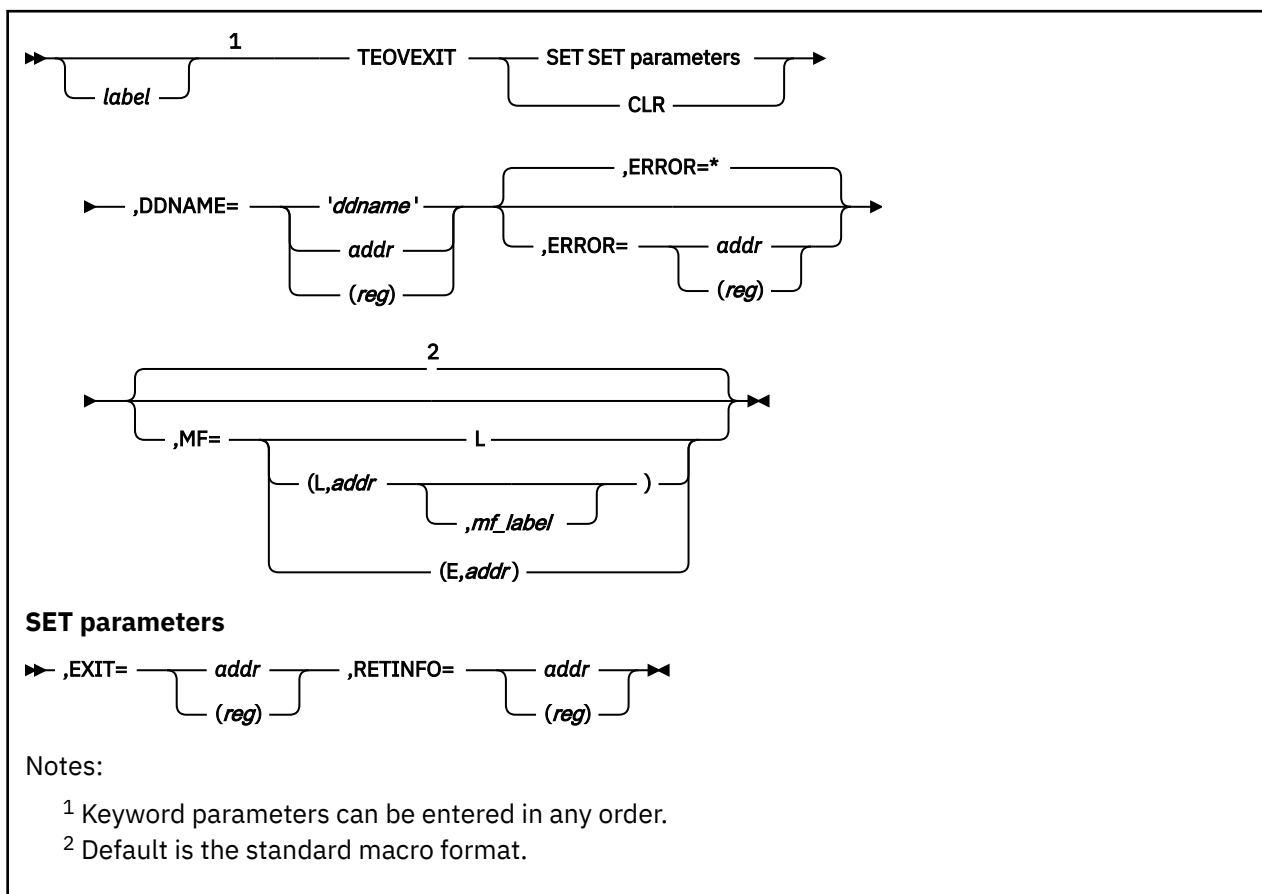
branches directly to the STRINIT routine. Because the STRINIT routine is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR).

Usage Notes

1. CMS releases GETMAIN storage at SVC 202/CMSCALL termination. This means that the program being returned to does not need to issue the STRINIT macro.

Unless you specify otherwise, CMS treats STRINIT as a no-op. If a program depends upon another program's ability to obtain and return GETMAIN storage to it, you can use the SET STORECLR command to instruct CMS to handle GETMAIN storage the way GETMAIN storage was handled in previous releases. In this case you would specify SET STORECLR ENDCMD.

TEOVEXIT



Purpose

Use the TEOVEXIT macroinstruction to set up and clear a CMS tape end-of-volume exit.

This macro is contained in DMSOM MACLIB. The CMSCB and IO macros map internal CMS data areas and are used with the TEOVEXIT macro and FILEDEF AUXPROC facility to monitor or modify I/O operations in CMS. For more information on FILEDEF AUXPROC, see [z/VM: CMS Application Development Guide for Assembler](#).

Parameters

Required Parameters:

SET

establishes an exit.

CLR

clears an exit.

DDNAME=

is the data definition name for which you set the tape end-of-volume exit. Acceptable values are:

'ddname'

specifies the name as 1 to 8 alphanumeric characters enclosed in quotation marks.

addr

specifies the name as an assembler label.

(reg)

specifies a general register containing the address of the name.

EXIT=

specifies the address of the program's end-of-volume processing routine.

addr

is the symbolic address of the program's end-of-volume processing routine.

(reg)

specifies a general register that contains the address of the program's end-of-volume processing routine.

The exit routine receives control after trailer labels have been processed and the tape has been rewound and unloaded. It receives control with the same PSW key as the call to CMS QSAM. The registers passed to the exit are the same as they were at the call to QSAM except that (a) register 0 points to the data control block (DCB), (b) register 1 points to the file control block (FCB), and (c) register 14 contains the address the routine branches to when it completes. If the exit does not return control to the address in register 14, future operations are unpredictable for that file. Register 15 contains the address of the user exit routine.

(This attribute is required for SET. If you specify the EXIT attribute on CLR, it is ignored. No MNOTE is issued.)

Note: The exit routine should not alter the registers of the program that issues the QSAM call.

RETINFO=

specifies the address of a 20-byte halfword aligned area to contain return information.

addr

is the symbolic address of a 20-byte halfword aligned area.

(reg)

specifies a general register that contains the address of a 20-byte halfword aligned area.

The RETINFO parameter is required for TEOVEXIT SET; it is ignored on TEOVEXIT CLR (no MNOTE is issued).

Optional Parameters:

label

is an optional assembler label for the statement.

ERROR=

specifies an action to be taken if an error occurs. If you do not specify the ERROR= parameter, control passes to the next sequential instruction. Acceptable values are:

passes control to the next sequential instruction. This is the default value.

addr

passes control to the specified address.

(reg)

passes control to the address in the specified register.

Note: Do not specify the ERROR= parameter with the list (MF=L) or complex list (MF=(L,addr,mf_label)) macro forms.

MF=

specifies the macro form. Omitting the MF parameter specifies the standard format. For more information about the MF parameter, see [“CMS Macro Formats” on page 15](#). Acceptable values are:

L

specifies the list format.

(L,addr,mf_label)

specifies the complex list format. Specify *addr* as an assembler expression or as a register enclosed in parentheses. The *mf_label* parameter is optional.

(E, *addr*)

specifies the execute format. Specify *addr* as an assembler expression or as a register enclosed in parentheses.

CMS QSAM Tape End-of-Volume Exit. A program working with CMS simulation of OS QSAM can set up an exit that could be entered on the end-of-volume condition on IBM standard label tapes. This exit should not be confused with the OS/MVS DCB end-of-volume exit. The OS/MVS DCB end-of-volume exit continues to be unsupported.

Restrictions:

1. Tape end-of-volume exits apply only to CMS OS QSAM simulation.
2. Only IBM standard label tapes are supported. If you use labels other than standard labels, you receive a return code of 16 from TEOVEXIT.
3. The LEAVE option of the FILEDEF command is invalid. If it is used, you receive a return code of 20 from TEOVEXIT.
4. The NOEOV processing option of the FILEDEF command is invalid. If it is used, you receive a return code of 28 from TEOVEXIT.
5. You cannot read backward. If it is attempted, the results are unpredictable.
6. The tape end-of-volume exit is not entered if either an OPEN or a CLOSE is in progress.
7. The exit must not issue I/O requests that might result in the tape end-of-volume exit being invoked. If it is attempted, the results are unpredictable.
8. The exit must not issue additional QSAM requests to the file. If it is attempted, the results are unpredictable.
9. The exit must not modify or clear the FCB of the file the end-of-volume condition was encountered on.
10. TEOVEXITs are cleared whenever a CLOSE or a CLOSE type T is issued for the file.

Return Codes

If any errors occur during the processing of the TEOVEXIT macro, register 15 contains the error return codes.

SET Function:

Code**Meaning****0**

Normal completion, an end-of-volume exit is established for the specified DDNAME.

4

The DDNAME specified is not found. (No FILEDEF was found with the given DDNAME.)

8

The device specified in the FILEDEF is not a tape device.

12

The tape identification is invalid; it must be TAP0-TAPF.

16

The tape label type is other than SL.

20

LEAVE is specified in the FILEDEF (FCB).

24

An invalid parameter list was specified.

28

NOEOV is specified in the FILEDEF (FCB).

32

The exit address or RETINFO address is zero.

CLR Function:

Code**Meaning****0**

Normal completion--the end-of-volume exit is cleared for the specified DDNAME or the end-of-volume exit was not in effect, but was still cleared.

4

The DDNAME specified is not found. (No FILEDEF was found with the given DDNAME.)

24

An invalid parameter list was specified.

Successful Completion:

On successful completion of TEOVEXIT SET (register 15=0), the RETINFO attribute contains:

Word**Meaning****0**

The symbolic tape number associated with the given DDNAME (character TAP0-TAPF).

1

The address of the FCB of the given DDNAME.

2

RESERVED

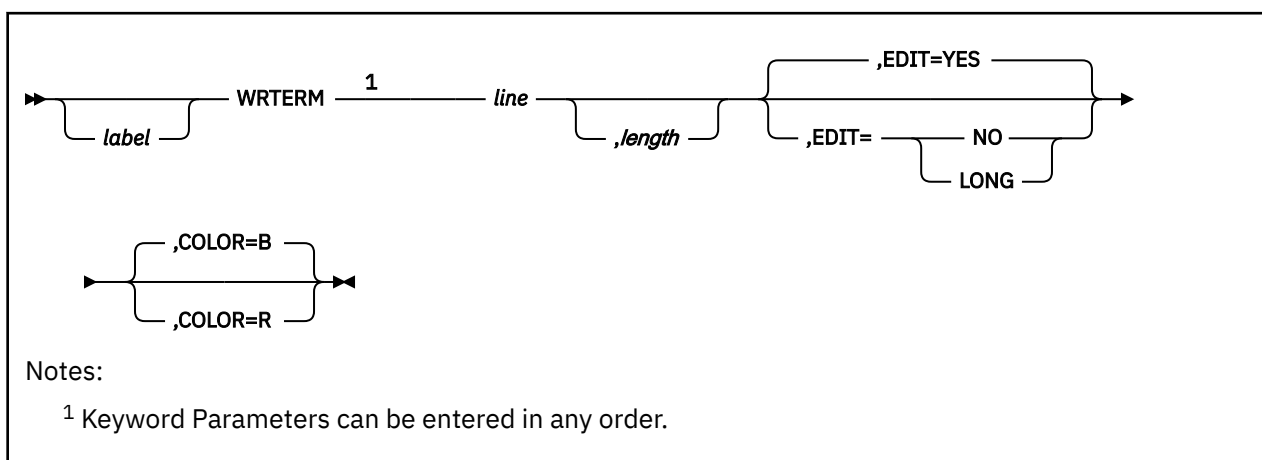
3

RESERVED

4

RESERVED

WRTERM



Purpose

Use the WRTERM macroinstruction to display a line of text at the terminal. New programs and programs that support 31-bit addresses should use the LINEWRT or APPLMSG macros rather than WRTERM. (Use APPLMSG if you want to specify the line of text on the macro call itself; otherwise, use LINEWRT.) WRTERM is maintained for compatibility to previous releases and does not work above the 16 MB line.

Parameters

Required Parameters:

line

specifies the line to be displayed:

'linetext'

the actual text line enclosed in single quotation marks.

lineaddr

the label on the statement containing the line.

(reg)

a register containing the address of the line.

Optional Parameters:

label

is an optional assembler label for the statement.

length

specifies the length of the line. If you specify the line within quotation marks, you can omit the length operand. Otherwise, specify length as:

n

a self-defining term indicating the length.

(reg)

a register containing the length.

EDIT=

specifies whether CMS edits the line:

YES

removes trailing blanks and adds a carriage return to the end of the line. YES is the default value.

NO

trailing blanks are not removed and no carriage return is added.

LONG

indicates the line may exceed 130 bytes. No editing is performed.

COLOR=

indicates the color of the line typed, if the typewriter terminal has a 2-color ribbon:

B

types the line in black. This is the default.

R

types the line in red.

Usage Notes

1. The maximum line length is 130 characters for a black line and 126 characters for a red line.
2. If EDIT=LONG, you must specify COLOR as *B*. In this case, you may write as many as 1760 bytes with a single WRTERM macroinstruction. You are responsible for embedding the proper terminal control characters in the data. (This operand is for use primarily with VS APL programs.)
3. Use the WAITT macroinstruction if you need to make sure that terminal I/O is complete before continuing program execution.
4. If you specify EDIT=NO or EDIT=LONG, differences in device characteristics may cause identical output to appear inconsistent.
5. If truncation occurs because the data is longer than the maximum line length, and the truncation occurs within a mixed DBCS string, then adjustments will be made to validate the truncated string.
6. Any translation done on the input buffer that contains both SBCS and DBCS data will only occur on the SBCS portions of the data provided that the display is capable of supporting mixed DBCS.

Chapter 5. CMS Compatibility Functions

This chapter describes the assembler language functions CMS supports for compatibility only.

All of these functions run in an ESA, XA, or XC virtual machine although they are not capable of supporting 31-bit addressing. To execute CMS functions from application programs, set up a parameter list and then issue the CMSCALL macro.

IBM does not recommend compatibility group functions in new programs.

The following CMS functions are described in this section:

- ATTN
- NUCEXT
- SUBCOM
- TODACCNT
- WAITRD.

ATTN

Purpose

Note: The CMSSTACK macro supersedes the ATTN function. ATTN continues to work, but from below the 16 MB line only. It does not support 31-bit addressing.

The ATTN function inserts a line of input into the program stack. ATTN may be executed from an assembler language program through SVC 202 with the following parameter list:

PLIST	DS	OD	
	DC	CL8'ATTN'	
	DC	CL4'order'	where order may be LIFO or FIFO.
*			FIFO is the default
	DC	AL1(length)	length of line to be stacked
	DC	AL3(addr)	address of line to be stacked

Usage Notes

- 1. The line that ATTN stacks is extracted from the program stack when WAITRD is executed to read a line of input. (See the WAITRD function description for details of WAITRD function.)
- 2. ATTN stacks lines of up to 255 characters.

Return Codes

Code

Meaning

0

Normal completion.

25

No more storage.

NUCEXT

Purpose

Note: The preferred interface to the NUCEXT function is the NUCEXT macro. The following information is provided as a convenience for programmers whose current programs use the NUCEXT function.

The nucleus extension function (NUCEXT) lets you identify command entry points in programs established in free storage, so that they can be called by an SVC 202 as if they were nucleus commands. Thus, they become nucleus extensions. You can also create your own Immediate commands with the NUCEXT function.

NUCEXT builds a chain of SCBLOCKs in storage for nucleus extensions. The chain of nucleus extensions is reordered each time a command is found on the chain. The reordering puts the most frequently used commands at the beginning of the chain.

NUCEXT is a name given to a group of commands that all use an internal function named NUCEXT. The actual commands provided for manipulation of nucleus extensions are:

NUCXLOAD

Loads an ADCON-free or relocatable module into free storage and installs it as a nucleus extension.

NUCXDROP

Cancels a nucleus extension and releases the corresponding storage.

NUCXMAP

Prints on the console or stacks a list of the nucleus extensions.

Use NUCEXT to access user-written programs without having to do disk read operations (as would be required for modules) or to avoid thrashing in the transient or user areas when several programs are used repeatedly (the same programs are loaded many times).

Use NUCEXT for gathering statistics, filtering commands for various purposes, creating anchors for data kept in free storage until the next CMS IPL, and special operations during CMS abnormal end processing.

Nucleus extensions with the IMMCMD option can receive control as user-defined Immediate commands or as regular commands. Nucleus extensions with the ENDCMD option receive control at normal end-of-command processing. The ENDCMD nucleus extensions only receive control after a command is entered from the virtual console. They do not receive control if the command was issued from an EXEC, a user program, or CMS SUBSET mode. Unlike transient routines or user programs, nucleus extensions are retained until they are explicitly unloaded, or as a side effect of abnormal end cleanup for those using free storage of type 'user' (that is reclaimed during an abnormal end) or are not designated as system routines to survive abnormal end. Nucleus extensions can have the same name as existing CMS nucleus commands or functions. If they do have the same name, the extensions override the existing nucleus commands or functions. Only nucleus functions invoked through SVC 202 can be overridden. Two existing nucleus functions, RDBUF and WRBUF, however, cannot be overridden. It is possible to create a nucleus extension that can call another nucleus extension having the same name. This allows a nucleus extension to *frontend* another nucleus extension. The techniques necessary to perform this call are complex and require assembler language programming. This override process may not be possible in all cases.

The last nucleus extension to be established receives control first. This is the first nucleus extension on the SCBLOCK chain with a name that matches the requested name.

The nucleus extension may perform whatever processing it requires. To pass control to another nucleus extension having the same name, you must first use the NUCEXT RENAME macro to change the name field of the original SCBLOCK to a unique name.

The original nucleus extension can now issue an SVC 202 for the nucleus extension control that is to be passed. The original nucleus extension can restore the original contents of general registers 0 and 1 before this call.

Control is passed to the next nucleus extension with the same name on the SCBLOCK chain. The nucleus extension receives the PLIST that was pointed to by registers 0 and 1 when the SVC 202 was issued on the first nucleus extension.

On return from the second nucleus extension, the original nucleus extension must now issue an SVC 202 for itself. The name used for this SVC 202 must be the unique name that was placed in the SCBLOCK earlier. This call reorders the SCBLOCK chain, placing the original nucleus extension at the head of the SCBLOCK chain. The nucleus extension must be designed to recognize these special reorder calls. Reorder calls can be determined by checking the parameter list that is pointed to by register 1 upon entry. If the unique name is the first token in the PLIST, then this is a reorder call. Control should only be returned to the caller; typically, no processing should be performed.

The original nucleus extension should now restore the name field of its SCBLOCK to its original name. Control can now be returned to the original caller.

Nucleus Extensions and Abnormal Ends. There are two types of nucleus extensions.

Types of Nucleus Extensions. The types of nucleus extensions, *system* and *user*, differentiated by their behavior during a CMS abnormal end. The system nucleus extension will survive an abnormal termination of a user program (abnormal end), whereas the user nucleus extension will not.

Note: Because CMS reclaims all storage of type *user* during the abnormal end cleanup phase, any nucleus extension in user storage is deleted during abnormal end, regardless of its system attribute. The storage obtained for user type nucleus extensions code must be doubleword aligned to the next doubleword or CMSSTOR errors will occur during ABEND processing.

Because of this storage reclamation during abnormal end, programs which build data structures in free storage of type 'user' but keep pointers in storage of type 'system' need to know when abnormal end cleanup occurs (for example, after HX).

Service Calls: PURGE and RESET. A program's need to know about abnormal end cleanup is supported by the idea of a service call. When a nucleus extension is declared (through NUCEXT), it may request that it receive a service call under appropriate circumstances. There are two standard service calls supported by NUCEXT. The PURGE service call is issued during CMS abnormal end cleanup. The RESET service call is issued by the NUCXDROP program when a nucleus extension is explicitly unloaded. It is the responsibility of the unloaded program to cancel any secondary nucleus extension entry points by reacting to the RESET service call issued by NUCXDROP before the main entry point is canceled and the program is unloaded. The RESET call allows programs with several entry points to cancel these at the same time, or to free static storage areas obtained from free storage.

A note on service calls during an abnormal end. Do not stack during a service call. This causes the system to allocate storage that is not accounted for during abnormal end.

The SYSTEM and SERVICE Attributes. Nucleus extensions may or may not have the *SYSTEM* attribute and the *SERVICE* attribute. These attributes determine the handling of a nucleus extension during abnormal end processing.

If a nucleus extension has the *SYSTEM* attribute, it remains active after an abnormal end. It is your responsibility to see that such a nucleus extension is loaded into nucleus storage, not user storage (which is recovered after an abnormal end).

If a nucleus extension has the *SERVICE* attribute, it is called during abnormal end processing with the parameter list:

DS	0F
DS	CL8'nucleus extension name'
DC	CL8'PURGE'
DC	8X'FF'

The high-order byte in register 1 is set to X'FF'. A nucleus extension may have the *SYSTEM* and *SERVICE* attributes in any combination.

Nucleus Storage. Remember that during abnormal end recovery,

- When a nucleus extension has the *SYSTEM* attribute, it should be in nucleus storage and the length word is used by abnormal end recovery to account for the amount of storage used by that program.
- If a nucleus extension does not have the *SYSTEM* attribute but is in nucleus storage anyway, that storage will be recovered during abnormal end.

When a nucleus extension obtains nucleus-type free storage other than what is accounted for by the origin and length fields in the SCBLOCK, it should:

1. Use the *SERVICE* flag so that it is called with the PURGE parameter list during abnormal end, at which time it returns any nucleus-type storage it obtained (but not that described in its SCBLOCK).
2. If it has the *SYSTEM* attribute, account for any extra nucleus storage which is to be kept through an abnormal end by adding the length in doublewords of such storage into the NUCXFRES field in NUCON. It is a good idea to update this field as soon as the storage is obtained. This is required if the nucleus extension does not have the *SERVICE* attribute.

Nucleus extensions remain in effect until canceled, either explicitly or implicitly. Implicit cancellation normally occurs only for nucleus extensions of the *user* type (during an abnormal end cleanup time when all storage of *user* type is reclaimed). Explicit cancellation does not release the storage (if any) occupied by the nucleus extension: that is the responsibility of the program that issues the cancellation (usually the program NUCXDROP).

Using the NUCEXT function affects the command resolution strategy of CMS when a SVC 202 is processed. Nucleus extensions are sought before functions in the real CMS nucleus. This gives the user the ability to intercept, filter, augment, and so forth, the *real* nucleus functions.

ENDCMD ATTRIBUTE. A nucleus extension with the ENDCMD option receives control at normal end-of-command processing. At end-of-command processing the CMS console handler invokes all nucleus extensions having the ENDCMD option. The nucleus extensions are invoked by a SVC 202. Register 1 points to the PLIST, the high-order byte of register 1 is set to X'FE' to indicate an end-of-command call. The PLIST used to invoke an ENDCMD extension is:

```
DS    0F
DS    CL8'nucleus extension name'
DC    CL8'ENDCMD'
DS    F'return code'
DC    8X'FF'
```

where:

return code is the value returned to CMS in register 15 by the terminating command.

Linkage Conventions. When a nucleus extension is declared, the following information must be provided:

- The NAME of the command implemented by the nucleus extension.
- The PSW to be used when passing control to the nucleus extension.
- The address and length of the storage area occupied by the program. The length must be rounded up to doubleword alignment.
- Flag bits to indicate either *user* or *system* type, and indicate whether service calls are desired.
- Flag bits should be used to indicate if the ENDCMD or IMMCMD options are desired.

Secondary entry points are declared by indicating a storage size of zero. The PSW specifies the system mask, the PSW key to be used, the program mask (and initial condition code), and the starting address for execution. The problem-state bit and machine-check bit may be set. The machine-check bit has no effect in CMS under CP. The EC-mode bit and the wait-state bit cannot be set (they are always forced to zero). The flag bits are encoded in the third byte of the PSW. Also, one byte of user defined flags and one 4-byte user-defined word can be associated with the nucleus extension, and referred to when the entry point is subsequently called.

Entry into a Nucleus Extension. On entry to a nucleus extension, the register contents are:

R0	Address of extended parameter list (if one was provided by the caller).
R1	Address of the command name (and the tokenized parameter list).
R2	Address of SCBLOCK with NUCEXT extension.
R12	Entry point address.
R13	24-word save area address.
R14	Return address (CMSRET).
R15	Entry point address.

This is the standard entry point convention except that R2 points to the SCBLOCK.

The NUCEXT function queries, declares, or cancels user nucleus extensions. NUCEXT can be issued as a command only for its query function. With one argument, 'name,' it returns either:

```
0    'name' is a user nucleus extension (found it).
```

or

```
1    'name' not found.
```

PLISTs: As a function (called from a program), NUCEXT takes the following PLIST:

Declare PLIST:

NUCX	DS	0F	PLIST TO DECLARE NUCLEUS EXTENSION
	DC	CL8'NUCEXT'	
NUCXNAME	DC	CL8'name'	COMMAND NAME
NUCXPSW	DC	XL2'0000',AL2(0)	SYSTEM MASK, STORAGE KEY, ETC
NUCXADDR	DC	A(*-*)	ENTRY ADDRESS, -1 for QUERY
	DC	A(0)	USER WORD
NUCXORG	DC	A(*-*)	LOAD ADDRESS
NUCXLEN	DC	A(*-*)	SIZE, IN BYTES ROUNDED TO THE NEXT DOUBLEWORD.

This declares 'name' as a nucleus extension and puts an SCBLOCK at the head of the NUCEXT chain. The name may already be defined, in which case the previous declaration will be hidden until the present one is canceled. Return code 25 means not enough storage was available to allocate the necessary SCBLOCK.

The third and fourth bytes of the start-up PSW (interrupt code) are used as flag bytes. The format of the PSW is:

AL1(system mask)	(EC-mode bit forced to 0)
BL4(storage key)	
BL4'0MWP'	
AL1(NUCEXT flags)	System=X'80', Service=X'40', End of command=X'10', Immediate=X'04'
AL1(user flag)	May be used for private purpose.
A(entry point)	

Cancel PLIST:

```
CL8'NUCEXT'
CL8'name'
XL4'irrelevant'
A(0) identifies the cancel function
```

This cancels the nucleus extension or gives a return code of 1 if 'name' is not found. The storage occupied by the program calling for this nucleus extension is not freed. This is the responsibility of the canceling program.

Query PLIST:

```
CL8'NUCEXT'
CL8'name'
```

XL4'irrelevant'	Receives A(SCBLOCK).
XL4'FFFFFFFF'	Identifies the query function

This form returns the address of the SCBLOCK if 'name' is found, otherwise it changes nothing and gives a return code of 1.

Note that if 'NUCEXT name' is called from a command level or from an EXEC file, the Query PLIST is the form of PLIST which will be issued.

Get Anchor PLIST:

CL8'NUCEXT'	
CL8'irrelevant'	
A(*-*)	Receives value (not address) of NUCEXT list anchor or 0 if there are no nucleus extensions.
A(1)	Indicates request for anchor.

Nucleus Extensions as Immediate Commands. When a nucleus extension is established with the IMMCMD option, it can be invoked as a regular command or as an Immediate command. In addition to having an SCBLOCK, a nucleus extension with IMMCMD attribute also has a similar control block, called an IMMBLOK, associated with it.

Nucleus extensions with the IMMCMD attribute are entered as Immediate commands when they are invoked by the CMS console interrupt handler. This occurs when a particular command that has been established as an Immediate command is entered by the terminal user while CMS is busy.

Nucleus extensions with the IMMCMD attribute can be overridden by an identically named nucleus extension (for example, NUCXLOAD with the PUSH option). If the new nucleus extension does not have the IMMCMD attribute but does have the same name as an existing nucleus extension with the IMMCMD attribute, the nucleus extension with the IMMCMD attribute is disabled as an Immediate command.

Entry conditions to a nucleus extension as an Immediate command are identical with entry conditions that occur when a nucleus extension is invoked through SVC 202, except for the following conditions:

- The high-order byte of register 1 contains X'06'. This indicates that the nucleus extension was invoked as an Immediate command. When invoked through SVC 202, the high-order byte of register 1 is normally X'01' or X'0B'.
- Register 2 contains the address of an IMMBLOK.
- Register 14 contains the return address that is located in the CMS console interrupt handler.

With respect to common information (for example, command name and user word), displacements within the IMMBLOK are identical with those in a SCBLOCK. These displacements are as follows:

Displacement	Offset Information
0	Pointer to next IMMBLOK
4	User word
8	Command name
20	Entry point address

For more information on the Immediate commands in CMS, see [z/VM: CMS Application Development Guide](#).

SUBCOM

Purpose

Note: The preferred interface to the SUBCOM function is the SUBCOM macro. The following information is provided as a convenience for programmers whose current programs use the SUBCOM function.

Dynamic Linkage/SUBCOMM: It is possible for a program that is already loaded from disk to become dynamically known by name to CMS for the duration of the current command; such a program can be called through SVC 202. In addition, this program can also make other programs dynamically known if the first program can supply the entry points of the other programs.

To become known dynamically to CMS, a program or routine invokes the create function of SUBCOM. To invoke SUBCOM, issue the following calling sequence from an assembler language program:

```

                LA    R1,PLIST
                SVC   202
                DC    AL4(ERROR)
                .
                .
                .
PLIST          DS    0F
                DC    CL8'SUBCOM'
SUBCNAME       DC    CL8'name'      COMMAND NAME
SUBCPSW        DC    XL2'0000'      SYSTEM MASK, STORAGE KEY, ETC.
                DC    AL2(0)        RESERVED
SUBCADDR       DC    A(0)          ENTRY ADDRESS, -1 FOR QUERY PLIST
                DC    A(0)          USER WORD

```

SUBCOM creates an SCBLOCK control block containing the information specified in the SUBCOM parameter list. SVC 202 uses this control block to locate the specified routine. All nonsystem SUBCOM SCBLOCKS are released at the completion of a command (that is, when CMS displays the ready message). A SUBCOM environment may be defined as a system SUBCOM by setting a X'80' in the first byte of the interruption code field of the PLIST. See page [“SCBLOCK” on page 376](#) for a description of the SCBLOCK control block.

When a program issues an SVC 202 call to a program that has become known to CMS through SUBCOM, it places X'02' in the high-order byte of register 1. Control passes to the called program at the address specified by the called program when it invoked SUBCOM.

The PSW in the SCBLOCK specifies the system mask, the PSW key to be used, the program mask (and initial condition code), and the starting address for execution. The problem-state bit and machine-check bit may be set. The machine-check bit has no effect in CMS under CP. The EC-mode bit and wait-state bit cannot be set. They are always forced to zero. Also, one 4-byte, user-defined word can be associated with the SUBCOM entry point and referred to when the entry point is subsequently called.

When control passes to the specified entry point, the register contents are:

R0

Same as caller.

R1

Same as caller.

R2

Address of SCBLOCK for this entry point.

R12

Entry point address.

R13

24-word save area address.

R14

Return address (CMSRET).

R15

Entry point address.

You can also use SUBCOM to delete the potential linkage to a program or routine's SCBLOCK, or you can use SUBCOM to determine if an SCBLOCK exists for a program or routine.

To delete a program or routine's SCBLOCK, issue:

```
DC CL8'SUBCOM'
DC CL8'program or routine name'
DC 8X'00'
```

To determine if an SCBLOCK exists for a program or routine, issue:

```
DC CL8'SUBCOM'
DC CL8'program or routine name'
DC A(0) SCBLOCK addressed as a returned value
DC 4X'FF'
```

If 'SUBCOM name' is called from an EXEC file, the QUERY PLIST is the form of PLIST that is issued.

To query the chain anchor, issue:

```
DC CL8'SUBCOM'
DS CL8 (contents not relevant)
DS AL4 Will receive chain anchor
        contents from NUCSCBLK
DC AL4(1) Indicates request for anchor
```

Note that the anchor is equal to F'0' if there are no SCBLOCKs on the chain.

Note: If you create SCBLOCKs for several programs or routines with the same name, they are all remembered, but SUBCOM uses the last one created. A SUBCOM delete request for that name eliminates only the most recently created SCBLOCK making active the next most recently created SCBLOCK with the same name.

When control returns to CMS after a console input command has terminated, the entire SUBCOM chain of SCBLOCKs is released. None of the subcommands established during that command are carried forward to be available during execution of the next console command.

Return Codes

Return codes from the SUBCOM function are:

0

Successful completion. A new SCBLOCK was created, the specified SCBLOCK was deleted, or the specified program or routine has an SCBLOCK.

1

No SCBLOCK exists for the specified program or routine. This is the return code for a delete or a query.

20

The name specified on the SUBCOM macro contains an invalid character. The following characters are invalid: =, *, (,) and X'FF'.

25

No more free storage available. SCBLOCK cannot be created for the specified program or routine.

CMS SUBCOMM Environment

A function is provided that lets you invoke a command (from a program) that is resolved according to the CMS command search hierarchy. For example, the command is resolved just as though the command was entered from the terminal. This SUBCOM function is named CMS. This command search function checks the IMPEX and IMPCP settings of CMS SET.

SUBCOM

The CMS SUBCOM function is defined during system initialization at IPL and remains defined during the entire CMS session.

To pass a command to the CMS SUBCOM function, the user should define PLISTs as follows:

```
PLIST    DS    0F
         DC    CL8'CMS'
EXPLIST  DS    0F
         DC    A(PLIST)
         DC    A(BEGARGS)
         DC    A(ENDARGS)
         DC    A(0)           (or address of CSFCB)
BEGARGS  DS    0F
         DC    C'command to be invoked'
ENDARGS  EQU    *
```

Register 1 must contain the address of PLIST and a high-order byte of X'02'. Register 0 must contain the address of the extended PLIST. Having established the PLIST and register information the user issues an SVC 202. The X'02' in the high-order byte of register 1 indicates that this is a call to a previously defined SUBCOM.

The fourth word of the extended plist is used for recursion control as follows:

- To inhibit the recursion of execs, the word must be the address of a CSFCB containing a pointer to the 8-byte name of the current exec.
- To allow recursion of execs, the word must be zero (that is, there is no CSFCB).

This function can also be invoked using the CMSCALL macro specifying CALLTYP=CMS.

TODACCNT

Purpose

Use the TODACCNT function to issue a DIAGNOSE code X'70' for activating the time-of-day clock accounting interface. Using the TODACCNT function helps to avoid DIAGNOSE code X'70' calls (a specification exception). For more information on the DIAGNOSE code X'70' call, refer to [z/VM: CP Programming Services](#).

The TODACCNT function has two subfunctions, ENABLE and QUERY.

- ENABLE tells CMS to issue a DIAGNOSE code X'70' instruction to indicate to CP that the virtual machine wishes to receive timing information. Each time the virtual machine is dispatched, CP provides the accumulated processor time the virtual machine has used and the time-of-day that the virtual machine was dispatched. This information is stored in a 16-byte area in page zero. Refer to [Table 25 on page 503](#).
- QUERY function returns the 16 bytes of timing information supplied by CP as a result of the enable function.

TODACCNT ENABLE is executed from a program through an SVC 202 with the following parameter list:

```

PLISTE   DS    OD
          DC    CL8'TODACCNT'
          DC    CL8'ENABLE'      ENABLE function
          DC    F'0'             Address of timing information in
*                                     page zero is returned here
*                                     (provided the return code is 0 or 4).
```

TODACCNT QUERY is executed from a program through an SVC 202 with the following parameter list:

```

PLISTQ   DS    OD
          DC    CL8'TODACCNT'
          DC    CL8'QUERY'      QUERY function
          DC    2D'0'           Timing information (16-bytes) will
*                                     be transferred from page zero
*                                     (provided the return code is 0).
```

TODACCNT 16 byte output area contains the following information:

Table 25. TODACCNT 16-byte timing information	
8-bytes	8-bytes
TOTCPU	TOD CLOCK

Total processor time (*TOTCPU*) is in TOD clock units. For more information on the TOD clock and its unit of measurement, see [Enterprise Systems Architecture/390 Principles of Operation \(publibfp.dhe.ibm.com/epubs/pdf/dz9ar008.pdf\)](#).

Usage Notes

1. The parameter list must be on a double-word boundary.
2. An error return address must be supplied in the 4 bytes immediately following the SVC 202 instruction. If the return code (register 15) contains a nonzero value after returning from the SVC call, control passes to the address specified unless the address is equal to 1. If the address is 1, return is made to the next instruction after the DC AL4(1) instruction.

Return Codes

Register 15 contains one of the following codes.

Return codes for the ENABLE subfunction:

Code

Meaning

0

ENABLE function successfully completed. The address of the 16-byte area in page zero is returned in the parameter list.

4

ENABLE function has already been issued. The address of the 16-byte area in page zero is returned in the parameter list.

20

DIAGNOSE code X'70' has already been issued. CMS is not able to return the timer area address.

Return codes for the QUERY subfunction:

Code

Meaning

0

QUERY function successfully completed. The 16 bytes of timer information has been transferred from page zero to the parameter list.

12

ENABLE function has not been issued.

Return codes for the ENABLE and QUERY functions:

Code

Meaning

16

Invalid function specified. Valid functions are 'ENABLE ' or 'QUERY '. This should be an 8-byte field.

WAITRD

Purpose

Note: The LINERD macro supersedes the WAITRD function. WAITRD continues to work, but from below the 16 MB line only. It does not support 31-bit addressing.

Use the WAITRD function to read a line of input from the virtual machine console, the program stack or the terminal input buffer. Use CMSCALL with the following parameter list to execute WAITRD from an assembler language program:

```
PLIST    DS    0F
         DC    CL8'WAITRD'
         DC    AL1(1)
         DC    AL3(input buffer address)
         DC    CL1'code1'
         DC    CL1'code2'
         DC    AL2(length of buffer)
         DC    AL4(prompt buffer address)
         DC    AL4(prompt buffer length)
```

WAITRD first reads from the program stack. If the program stack is empty, WAITRD reads from the terminal input buffer. If the terminal input buffer is empty, WAITRD reads from the virtual machine console. However, if you desire, WAITRD can bypass the contents of the program stack and the terminal input buffer and read directly from the virtual machine console.

After WAITRD reads a line of input, the line is stored in your input buffer. The input buffer address specifies the address of this buffer.

The prompt buffer address and prompt buffer length are optional parameters. If they are used, the prompt information is written from either the buffer specified by the prompt buffer address or your input buffer (if the prompt buffer address is not specified). The prompt buffer length specifies the length of the prompt information to be written prior to the read. Prompt information is written with no carriage return and is used with TTY type devices.

Note: If the prompt parameters are used with code1 = W, Z, *, or \$, the read buffer may not be used for the prompt data because the read buffer is cleared prior to the execution of the function.

code1

The following codes specify what kind of processing WAITRD performs on lines read from the terminal input buffer. For codes U, V, S, T, and X, you must specify a buffer length of 130 bytes in the 'length of buffer' field in the WAITRD parameter list. For code Y, a buffer length of 255 must be specified.

Code

Meaning

U

Reads a logical line, pads it with blanks, and translates it to uppercase.

V

Reads a logical line and translates it to upper case; does not pad with blanks.

S

Reads a logical line and pads it with blanks.

T

Reads a logical line; does not pad with blanks.

X

Reads a physical line.

Y

Reads a logical line, pads with blanks to 255, does no uppercase translation and does not do SET INPUT translation.

The following codes specify what kind of processing WAITRD performs on lines read from the program stack. The length of the input buffer may be up to 255 bytes.

Code

Meaning

W

Reads a physical line; performs no uppercase translation or padding with blanks.

Z

Reads a physical line and translates it to upper case; does not pad with blanks.

Use the following codes when you use APL under CMS. The length of the buffer may be up to 2030 bytes.

Code

Meaning

*

Reads a physical line into the caller's buffer. (See Usage Note 4.)

\$

Reads a physical line into the caller's buffer. (See Usage Note 4.)

code2

Code

Meaning

B

Write the prompt information before the read, and read a line of input directly from the virtual machine console.

D

Read a line of input directly from the virtual machine console.

P

Write prompt information before the read.

binary zeros

There is no prompt information, and do not read a line of input directly from the virtual machine console.

The prompt buffer address and the prompt buffer length are specified only if *code2* is B or P.

Usage Notes

1. Specify the input buffer length as the last parameter in the WAITRD parameter list. Upon completion of the WAITRD function, the 'number of bytes' field contains the number of bytes read.
2. WAITRD does not perform logical line editing when reading a physical line.

WAITRD performs line editing on lines read from the terminal input buffer (lines typed at the terminal), unless code X is specified; WAITRD does not perform logical line editing when you specify code X. WAITRD does not perform line editing (except uppercase translation, if requested) on lines read from the program stack.
3. Lines typed at the terminal (and stacked in the terminal input buffer) are scanned by CP for logical line editing characters. Logical line editing characters are set by the CP TERMINAL command. The line editing characters may be set for:

```
Chardel
Linedel
Linend
Escape
```

In addition, CMS scans the lines for the following two hexadecimal characters:

X'00' -

interpreted as the end of the physical line. Any character(s) to the right of this hexadecimal character is ignored.

X'15' -

interpreted as the end of the logical line. Any character(s) to the right of this hexadecimal character is interpreted as a new line.

4. For code \$, an attention interrupt during a read operation signals the end of the line and does not result in a restart of the read. For code *, an attention interrupt during a read results in a restart of the read operation.

Return Codes

Code	Meaning
0	Function completed successfully.
2	Invalid code. Read not completed.
4	Code=\$. An attention interruption ended the read operation.

Appendix A. Simplified RACROUTE Macro Functions

CMS applications can access the following RACROUTE macro functions, which support REXX, assembler, and C callers. The external interfaces (services) described in this section provide a subset of each macro function. Not every RACROUTE macro parameter is available, and not every value of a particular parameter is supported.

The following RACROUTE functions are available to the external interfaces:

REQUEST=STAT

Checks whether the FACILITY class is active.

REQUEST=AUTH

Checks whether a user has access to a given resource.

REQUEST=AUDIT

Creates GENERAL audit records.

For each function, the calling application is responsible for setting up the necessary ESM connection – for example by calling RPIUCMS INIT – and for documenting the necessary ESM-specific configuration actions. For example, if you are using RACF as the ESM, the caller must give REQUEST=STAT users READ authority to the FACILITY class ICHCONN profile, which is required to successfully invoke the RACROUTE function, and must give REQUEST=AUTH and REQUEST=AUDIT users UPDATE authority to ICHCONN.

For a complete description of these RACROUTE macro functions and the requirements for calling them, see *z/VM: Security Server RACROUTE Macro Reference*.

External Interfaces Supported for REXX Callers

For REXX callers, IBM provides the following supported external interfaces:

- The DMSWRAUD EXEC constructs a string containing the service module parameters and metadata used by the DMSWRRAC EXEC to add an entry to the ESM's security audit log. See [“DMSWRAUD” on page 510](#).
- The DMSWRAUT EXEC constructs a string containing the service module parameters and metadata used by the DMSWRRAC EXEC to test a user's authorization to a resource at a given access level. [“DMSWRAUT” on page 511](#).
- The DMSWRESM EXEC constructs a string containing the service module parameters and metadata used by the DMSWRRAC EXEC to verify an ESM resource class is active. See [“DMSWRESM” on page 510](#).
- The DMSWRRAC EXEC is a service wrapper. You can either call it or write your own equivalent. Given an argument string, it manages all the CMS event interactions, calls the service module, and makes the service module's output available in a convenient format. [“DMSWRRAC” on page 512](#).
- DMSWBRAC COPY, a binding file that allows application callers to use symbolic names. See [“DMSWBRAC COPY” on page 510](#).

IBM recommends that you use these external interfaces in order to reduce your implementation effort. While you are allowed to call the service modules without these interfaces, doing so means that you will have to write and debug more code than would otherwise be necessary. IBM installs these EXECs and binding files on the MAINT 193 disk.

IBM-Provided Binding Files

DMSWBRAC COPY

IBM provides the DMSWBRAC COPY binding file to define REXX variables that your REXX EXECs can re-use. All variables defined in this file are part of the stem variable "dmswbrac.". Before a REXX EXEC can access these variables, it must call the CMS APILOAD service, passing DMSWBRAC as the argument.

Example

```
Call APILOAD "DMSWBRAC" /* Bind constants into dmswbrac. stem */
```

IBM-Provided REXX EXECs

DMSWRESM

The DMSWRESM EXEC constructs a string containing the service module parameters and metadata used by the DMSWRRAC EXEC to verify an External Security Manager (ESM) resource class is active.

Input

DMSWRESM EXEC input consists of:

- Event type: "VMRACROUTE STAT" or dmswbrac.0EventNameReqStat
- Subsystem name
- Requestor name
- Class: only the FACILITY resource class is supported by the DMSWRESM EXEC. You specify this class by passing a value of "FACILITY" or by using the variable defined in the DMSWBRAC COPY binding file (dmswbrac.0ClassNameFacility).

DMSWRESM's input parameters exactly match the input parameters of the RACROUTE REQUEST=STAT module, DMSWSESM.

Output

The resulting value is a blank-delimited data string suitable for the DMSWRRAC EXEC parameter "VMRACROUTE STAT DMSWSESM *plist_data*".

Example

For an example of using the DMSWRESM EXEC in conjunction with the DMSWRRAC EXEC, see [“Calling Using the IBM-Provided REXX EXECs”](#) on page 519.

DMSWRAUD

The DMSWRAUD EXEC constructs a string containing the service module parameters and metadata used by the DMSWRRAC EXEC to add an entry to the ESM's security audit log.

Input

The DMSWRAUD input parameters are:

- Event name: "VMRACROUTE AUDIT" or dmswbrac.0EventNameReqAudit
- Subsystem name
- Requestor name

- Audit event type: only general events are currently supported by the DMSWRAUD EXEC. You specify general events by passing a value of "GENERAL" or by using the variable you defined in the DMSWBRAC COPY binding file (dmswbrac.0AuditEventName).
- Event qualifier code: RACROUTE treats this value as installation-defined, but only supports a limited set of values. It is the caller's responsibility to ensure uniqueness across callers if your goal is to be able to map an audit log entry back to an application based on the entry's event qualifier value.

If you are using SMAPI, when its authorization policy allows the use of both ESM and the SMAPI authorization list for request authorization, SMAPI can call this service with the following event qualifier codes:

```
00000001x or dmswbrac.0AuditEventQualifierAuthListSuccess
00000002x or dmswbrac.0AuditEventQualifierAuthListUnauthorized
```

- Class: only the FACILITY resource class is supported by the DMSWRAUD EXEC. You specify this class by passing a value of "FACILITY" or by using the variable defined in the DMSWBRAC COPY binding file (dmswbrac.0ClassNameFacility).
- Resource name: the resource name passed to the RACROUTE ENTITYX keyword. This can be any value that the ENTITYX parameter accepts as an entity name.
- Log string (optional): any string up to 255 bytes.

Output

The DMSWRAUD output consists of a blank delimited data string suitable for the DMSWRRAC EXEC parameter "VMRACROUTEAUDIT DMSWSAUD *plist_data*". (See [“Parameter List \(plist\) Layout for Input to DMSWRRAC”](#) on page 512.)

Usage Information

See [“Creating an Audit Log Entry for a Resource in the FACILITY Class”](#) on page 515.

DMSWRAUT

The DMSWRAUT EXEC constructs a string containing the service module parameters and metadata used by the DMSWRRAC EXEC to test a user's authorization to a resource at a given access level.

Input

The DMSWRAUT input parameters are:

- Event name: "VMRACROUTEAUTH" or dmswbrac.0EventNameReqAuth
- Subsystem name
- Requestor name
- Requesting user ID: user ID whose authorization is to be tested.
- Class: only the FACILITY resource class is supported by the DMSWRAUT EXEC. You specify this class by passing a value of "FACILITY" or by using the variable defined in the DMSWBRAC COPY binding file (dmswbrac.0ClassNameFacility).
- Resource name: the resource name passed to the RACROUTE ENTITYX keyword. This can be any value that the ENTITYX parameter accepts as an entity name.
- Application name: the application name passed to the RACROUTE APPL keyword. This can be any value that the APPL parameter accepts as an application name.
- The installation exit parameters passed to the RACROUTE INSTLN keyword. Must be valid according to the rules for that keyword.
- Log string: any string (up to 255 bytes)
- Authority level (optional). Can be "READ" (the default), "UPDATE", "CONTROL", or "ALTER".

Output

The DMSWRAUT output consists of a blank delimited data string suitable for the DMSWRRAC EXEC parameter "VMRACROUTEAUTH DMSWSAUT *plist_data*". (See [“Parameter List \(plist\) Layout for Input to DMSWRRAC”](#) on page 512.)

Usage Information

See [“Testing a User's Authority to Access a Resource in the FACILITY Class”](#) on page 517.

DMSWRRAC

The DMSWRRAC EXEC is a call interface to RACROUTE functions. DMSWRRAC calls a function (a module or an EXEC) by using as its input the output string produced by the DMSWRESM EXEC argument string constructor.

Input

DMSWRRAC EXEC input consists of:

- The name of a stem variable where its output will be stored. This variable is set in the caller's variable pool.
- An environment flag (1 for SMAPI, 0 otherwise).
- An event monitor token (an integer value or, to have this EXEC create a new monitor, two double quotes ("").
- The argument string constructor's output.

Output

DMSWRRAC EXEC output includes:

- A return code, as a REXX function result that can be explicitly assigned to a variable as shown in [“DMSWRRAC EXEC Stable Results”](#) on page 519, or that will be implicitly assigned to the "result" variable. The binding file DMSWBRAC COPY provides symbolic names beginning with "dmswbrac.0dmswrrac" for values that the DMSWRRAC EXEC return code can consist of, and (for the case when its return code is zero) names to index into the stem variable outputs.
- Output data that is also stored in the stem variable provided on input.

Usage Information

For an example with output and return codes, see [“Calling Using the IBM-Provided REXX EXECs”](#) on page 519.

Parameter List (plist) Layout for Input to DMSWRRAC

The parameter list (plist) used by the DMSWRRAC EXEC is a "direct format" plist. (See [“Direct Format Argument String: Eye Catcher VALU”](#) on page 522). The entire plist is contained in one section of storage. The DMSWRRAC EXEC performs verification of the input *plist_data*.

Data	Size in Bytes	Description
"VALU"	4	Key for direct format argument string
01x	1	Version
00x	1	Reserved
parm_offset_start	2	Offset to parameter offset list
num	4	Count of parameters

Data	Size in Bytes	Description
parm_values offset start	2	Offset to parameter values
0000x	2	Reserved
offset 1	4	Offset to the start of the first value
offset 1b	4	Offset to the byte past the end of the first value
...	...	
offset n	4	Offset to the start of the last value
offset nb	4	Offset to the byte past the end of the last value
parm value 1	...	Data for first parameter
parm value 2	...	Data for last parameter
... parm value n	...	Data for last parameter

Return Codes

A return code is returned along with diagnostics in the "*.0diag." variables, where the asterisk is a wildcard character. "*.0diag." represents any variable ending with ".0diag." Examples of errors are the following:

- Parameter list eye catcher invalid
- Parameter list version invalid
- Parameter list reserved bits non-zero
- Parameter list too short
- Parameter value too short
- Parameter value ends past the end of the parameter list

For more information, see [“Return Codes and Reason Codes”](#) on page 524.

External Interfaces supported for REXX, Assembler, and C Callers

Testing Whether a Class is Active with DMSWSESM

The DMSWSESM module provides CMS application access to the RACROUTE REQUEST=STAT (Determine RACF Status) macro function. For a complete description of this macro function, see [z/VM: Security Server RACROUTE Macro Reference](#).

Input

The input to this module can be in direct or in indirect format. For more information, see [“Calling Without Using the IBM-Provided REXX EXECs”](#) on page 522.

Usage Information

Use the following information to determine whether a class is active. Note that only the FACILITY class is currently supported. If you have a requirement to test other classes, please provide that feedback to IBM.

Input Parameter	Description
Result event name	Contains the RACROUTE macro call results. Its value <i>must</i> be the EBCDIC string VMRACROUTE _{STAT} .

Input Parameter	Description
Subsystem name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE SUBSYS= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the calling application for diagnostic purposes.
Requestor name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE REQSTOR= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the point of call for diagnostic purposes.
Resource class name	Passed through to the ESM. Its value <i>must</i> be the EBCDIC string FACILITY.
Argument String Constructors	
DMSWRESM EXEC on MAINT 193	For REXX callers
Service Wrappers	
DMSWRRAC EXEC on MAINT 193	For REXX callers
Binding Files	
DMSWBRAC COPY on MAINT 193	For REXX callers
Samples	
DMSWSSMI EXEC	For REXX callers
Module	
DMSWSESM	

Example

The following is an example of the RACROUTE parameters used by this module:

```
CALLESM RACROUTE REQUEST=STAT,RELEASE=1.9.2,MF=(E,StatPL),
        DECOUPL=YES,
        CLASS=ClassInput,
        SUBSYS=SubsysInput,
        REQSTOR=ReqstorInput,
        WORKA=RACROUTE_WorkArea
```

Testing Whether the FACILITY Class is Active

Use the following information to determine whether a class is active. Note that only the FACILITY class is currently supported. If you have a requirement to test other classes, please provide that feedback to IBM.

Input Parameter	Description
Result event name	Contains the RACROUTE macro call results. Its value <i>must</i> be the EBCDIC string VMRACTESTAT.
Subsystem name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE SUBSYS= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the

Input Parameter	Description
	RACROUTE macro. Its value should identify the calling application for diagnostic purposes.
Requestor name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE REQSTOR= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the point of call for diagnostic purposes.
Resource class name	Passed through to the ESM. Its value <i>must</i> be the EBCDIC string FACILITY.
Argument String Constructors	
DMSWRESM EXEC on MAINT 193	For REXX callers
Service Wrappers	
DMSWRRAC EXEC on MAINT 193	For REXX callers
Binding Files	
DMSWBRAC COPY on MAINT 193	For REXX callers
Samples	
DMSWSSMI EXEC	For REXX callers
Module	
DMSWSESM	

Example

The following is an example of the RACROUTE parameters used by this module:

```
CALLESM RACROUTE REQUEST=STAT,RELEASE=1.9.2,MF=(E,StatPL),
        DECOUPL=YES,
        CLASS=ClassInput,
        SUBSYS=SubsysInput,
        REQSTOR=ReqstorInput,
        WORKA=RACROUTE_WorkArea
```

Creating an Audit Log Entry with DMSWSAUD

The DMSWSAUD module provides CMS application access to the RACROUTE REQUEST=AUDIT (General-Purpose Security-Audit) macro function. For a complete description of this macro function, see [z/VM: Security Server RACROUTE Macro Reference](#).

Input

The input to this module can be in direct or in indirect format. For more information, see [“Calling Without Using the IBM-Provided REXX EXECs”](#) on page 522.

Creating an Audit Log Entry for a Resource in the FACILITY Class

Use the following information to create an audit log entry. Note that only the FACILITY class is currently supported. If you have a requirement to test other classes, please provide that feedback to IBM.

Input Parameter	Description
Result event name	Contains the RACROUTE macro call results. Its value <i>must</i> be the EBCDIC string VMRACROUTEAUDIT.
Subsystem name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE SUBSYS= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the calling application for diagnostic purposes.
Requestor name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE REQSTOR= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the point of call for diagnostic purposes.
Audit log event name	Passed through to the ESM. Its value <i>must</i> be the EBCDIC string GENERAL.
Audit log event qualifier	<p>Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE EQVAL= keyword. In addition, it <i>must</i> be a 4-byte binary value.</p> <p>Note: When SMAPI logs authorization decisions mediated by the SMAPI authorization process, it specifies an event qualifier of 1. Other CMS applications should use a different value if you want their audit records to be differentiated from the ones generated by SMAPI calls. For more information, see the section "Configuring an ESM for SMAPI Authorization Decisions" in z/VM: Systems Management Application Programming.</p>
Resource class name	Passed through to the ESM. Its value <i>must</i> be the EBCDIC string FACILITY.
Resource name	Passed through to the ESM. Its value <i>must</i> be an EBCDIC string. The service module creates the length-value structure that the RACROUTE macro requires for the RACROUTE ENTITYX= keyword.
Log string	Passed through to the ESM. Its value <i>must</i> be an EBCDIC string. The service module creates the length-value structure that the RACROUTE macro requires for the RACROUTE LOGSTR= keyword. Its length <i>can</i> be zero, but it <i>must not</i> be omitted.
Output return and reason codes	
When using the service wrapper	See “Calling Using the IBM-Provided REXX EXECs” on page 519. If the service wrapper's result is 0, the ESM's results are shown in “Calling Without Using the IBM-Provided REXX EXECs” on page 522.
When calling the service module directly	The service module's outputs are shown in “Calling Without Using the IBM-Provided REXX EXECs” on page 522.
Argument String Constructors	
DMSWRAUD EXEC on MAINT 193	For REXX callers

Input Parameter	Description
Service Wrappers	
DMSWRRAC EXEC on MAINT 193	For REXX callers
Binding Files	
DMSWBRAC COPY on MAINT 193	For REXX callers
Samples	
DMSWSXIA EXEC	For REXX callers
Module	
DMSWSAUD	

Example

The following is an example of the RACROUTE macro invocation parameters:

```
CALLESM RACROUTE REQUEST=AUDIT,RELEASE=1.9.2,MF=(E,AuditPL),
          DECOUPL=YES,
          EVENT=EventNameInput,
          EVQUAL=(2),
          CLASS=ClassInput,
          ENTITYX=ResourceNameInput,
          SUBSYS=SubsysInput,
          REQSTOR=ReqstorInput,
          LOGSTR=LogStringInput,
          WORKA=RACROUTE_WorkArea
```

For more information, see [“Calling Using the IBM-Provided REXX EXECs” on page 519](#).

Testing a User's Authority to Access a Resource with DMSWSAUT

The DMSWSAUT module provides CMS application access to the RACROUTE REQUEST=AUTH (Check RACF Authorization) macro function. For a complete description of this macro function, see [z/VM: Security Server RACROUTE Macro Reference](#).

Input

The input to this module can be in direct or in indirect format. For more information, see [“Calling Without Using the IBM-Provided REXX EXECs” on page 522](#).

Testing a User's Authority to Access a Resource in the FACILITY Class

Use the following information to test a user's authority to access a resource. Note that only the FACILITY class is currently supported. If you have a requirement to test other classes, please provide that feedback to IBM.

Input Parameter	Description
Result event name	Contains the RACROUTE macro call results. Its value <i>must</i> be the EBCDIC string VMRACROUTEAUTH.
Subsystem name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE SUBSYS= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the calling application for diagnostic purposes.

Input Parameter	Description
Requestor name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE REQSTOR= keyword. This value <i>will not</i> be used for ESM request routing; the service module always specifies DECOUPL=YES on the RACROUTE macro. Its value should identify the point of call for diagnostic purposes.
User ID	User ID whose access to the resource will be tested. The service module passes this value through to the ESM. Its value must comply with all rules documented for the RACROUTE USERID= keyword.
Resource class name	Passed through to the ESM. Its value <i>must</i> be the EBCDIC string FACILITY.
Resource name	Passed through to the ESM. Its value <i>must</i> be an EBCDIC string. The service module creates the length-value structure that the RACROUTE macro requires for the RACROUTE ENTITYX= keyword.
Application name	Passed through to the ESM. Its value must comply with all rules documented for the RACROUTE APPL= keyword.
Installation exit parameters	Passed through to the ESM. Its value <i>must</i> be an EBCDIC byte sequence. The service module creates the length-value structure that the RACROUTE macro requires for the RACROUTE INSTLN= keyword. Its length <i>can</i> be zero, but it <i>must not</i> be omitted.
Log string	Passed through to the ESM. Its value <i>must</i> be an EBCDIC string. The service module creates the length-value structure that the RACROUTE macro requires for the RACROUTE LOGSTR= keyword. Its length <i>can</i> be zero, but it <i>must not</i> be omitted.
Authority level	Passed through to the ESM. Its value <i>must</i> be an EBCDIC string. The service module translates the value to the corresponding value that the RACROUTE macro requires for the RACROUTE ATTR= keyword. It <i>can</i> be omitted; if it is omitted, the default is READ. If it is specified, then its value must be one of the string values permitted on ATTR= (READ, UPDATE, CONTROL, or ALTER). It cannot be a register.
Argument String Constructors	
DMSWRAUT EXEC on MAINT 193	For REXX callers
Service Wrappers	
DMSWRRAC EXEC on MAINT 193	For REXX callers
Binding Files	
DMSWBRAC COPY on MAINT 193	For REXX callers
Samples	
DMSWSXIA EXEC	For REXX callers
Module	

Input Parameter	Description
DMSWSAUT	

Example

The following is an example of the RACROUTE macro invocation parameters:

```
CALLESM RACROUTE REQUEST=AUTH,RELEASE=1.9.2,MF=(E,AuthPL),DECOUPL=YES,
          USERID=UserIdInput,
          CLASS=ClassInput,
          ENTITYX=ResourceNameInput,
          ATTR=(3),
          LOG=ASIS,LOGSTR=LogStringInput,
          SUBSYS=SubsysInput,
          REQSTOR=ReqstorInput,
          APPL=ApplNameInput,
          INSTLN=InstallationExitParmsInput,
          WORKA=RACROUTE_WorkArea
```

For more information, see [“Calling Using the IBM-Provided REXX EXECs”](#) on page 519.

Calling Using the IBM-Provided REXX EXECs

Using the external interfaces described in [“External Interfaces Supported for REXX Callers”](#) on page 509 reduces the amount of application code necessary to invoke the services. To use these interfaces, the process you need to follow is:

1. Be aware of the application specific parameters you want to use.
2. Call the appropriate argument string constructor to serialize the parameters.
3. Call the appropriate service wrapper to invoke the RACROUTE macro.
4. Check the output.

For example, the following REXX code tests whether the FACILITY class is active:

```
/* Sample REXX code to test whether the FACILITY class is active */
/* Parameters can be hard-coded on the following line */
statParms = DMSWRESM "VMRACROUTESTAT" , "MYSUBSYS" , "MYREQSTR" , "FACILITY"
wrracResult = DMSWRRAC "stat." , "0" , "" , statParms
```

DMSWRESM constructs a string containing the service module parameters and metadata used by DMSWRRAC EXEC to call the correct service module. DMSWRESM's input parameters exactly match the RACROUTE REQ=STAT module DMSWSESM's input parameters. (See [“Calling Without Using the IBM-Provided REXX EXECs”](#) on page 522.) All the argument string constructors follow this same pattern. They have no return codes, and their only output is the result string that you pass into the service wrapper DMSWRRAC.

DMSWRRAC EXEC takes several inputs: the name of a stem variable where its outputs should be stored, an environment flag, and the argument string constructor's output. When calling DMSWRRAC EXEC, always pass "0" for the environment flag. If you specify other values, DMSWRRAC will make incorrect assumptions about the run-time environment.

Here are two distinct classes of output from DMSWRRAC EXEC: the stable results of the call, and detailed diagnostics intended to help when things go wrong.

DMSWRRAC EXEC Stable Results

The contents of the stable results *values* stem variables are an intended programming interfaces, meaning future changes will be compatible. The DMSWBRAC COPY file contains variables to help further isolate application code from future changes. The contents of the stable results *labels* stem variables are *not* considered a stable interface; they may change without warning. The following shows sample code to

format the output of DMSWRRAC EXEC. The stat stem variable used in the Do loop must match the name of the stem variable passed as the first argument to DMSWRRAC EXEC.

```
if wrracResult \= dmswbrac.0dmswrracResultSuccess then do
  say "Output"
  do i = 1 to stat.0label.0
    say "      " stat.0label.i ":" stat.0value.i
  end
end
```

The stat parameter in the last line of the previous example ("Sample REXX code to test whether the FACILITY class is active") causes the DMSWRRAC EXEC to write its output lines to that REXX stem variable, which the above Do loop is printing out. These results are valid only if DMSWRRAC's result is 0.

The complete set of its results is listed below, and each one has a binding in DMSWBRAC COPY. Diagnostic results are usually available regardless of the return code.

The code example above does not have to change if, for example, in the future DMSWRRAC EXEC adds a new output. However, most programmatic uses need to know where in the stem variable particular information is found, and the corresponding code should depend only on stable interfaces so that it continues to work in the future. Consider the example of finding the SAF return code. The intended method for doing that, since it depends only on stable interfaces, is to use the bindings in DMSWBRAC EXEC, as shown in the following example:

```
/* REXX */
Call APILOAD "DMSWBRAC"
tailModRc = dmswbrac.0dmswrracOutputModRcIndex
tailSafRc = dmswbrac.0dmswrracOutputSafRcIndex

/* 'Gather parameters here means hardcoding them on the following line */
statParms = DMSWRESM "VMRACROUTESTAT" , "MYSUBSYS" , "MYREQSTR" , "FACILITY"
wrracResult = DMSWRRAC "stat." , "0" , "" , statParms
if wrracResult \= dmswbrac.0dmswrracResultSuccess then do
  ...trace and return to caller
end
if stat.0value.tailModRc \= dmswbrac.0dmswrracOutputModRcSuccess then do
  ...trace and return to caller
end
safrc = stat.0value.tailSafRc
```

Instead of using DMSWBRAC-defined variables, the code could have tested the value of the diagnostic output labels. IBM does not recommend doing this, however, because those labels might change at any time. The following example replaces DMSWBRAC-defined variables with other techniques that rely on unstable parts of the results, and these techniques should be avoided.

```
/* REXX */
/* 'Gather parameters' here is just hard-coding them on the following line */
statParms = DMSWRESM "VMRACROUTESTAT" , "MYSUBSYS" , "MYREQSTR" , "FACILITY"
wrracResult = DMSWRRAC "stat." , "0" , "" , statParms
if wrracResult \= 0 then do /* should use constant */
  ...trace and return to caller
end
modrc = -1
safrc = -1
do i = 1 to stat.0label.0
  say "      " stat.0label.i ":" stat.0value.i
  if stat.0label.i = "DMSWSESM rc (decimal)" then modrc = stat.0value.i /* label could change! */
  if stat.0label.i = "SAF rc (decimal)" then safrc = stat.0value.i /* label could change! */
end
if modrc \= 0 then do
  ...trace and return to caller
end
```

Return Code (Decimal)	Description
0	Normal completion. Results are available in the _input-stem_.0value. stem variable.

Return Code (Decimal)	Description
16	Failed trying to call the service module.
50	An unrecognized event name prefix was provided in the input parameters. Only limited validation is done; not all such cases are detected.
60	An unrecognized service module name prefix was provided in the input parameters. Only limited validation is done; not all cases are detected.
100	Unable to create an event monitor. The associated return/reason codes are available in the diagnostic results.
104	EventWait failure. The associated return/reason codes are available in the diagnostic results.
108	EventRetrieve failure. The associated return/reason codes are available in the diagnostic results.
112	Event monitor reset failure. The associated return/reason codes are available in the diagnostic results.
116	Event monitor delete failure. The associated return/reason codes are available in the diagnostic results.
200	Unable to copy results to the caller's variable pool.
204	Unable to copy result labels to the caller's variable pool.
208	Unable to copy diagnostic results to the caller's variable pool.

IBM recommends that you use the DMSWBRAC bindings. The previous example and the DMSWSSMI EXEC on the MAINT193 disk show the use of these bindings.

DMSWRRAC EXEC makes the following service module results available in the stem variable you supply in the first parameter when you call DMSWRRAC EXEC. DMSWBRAC COPY contains bindings for each of them, isolating your client code from future changes to the bound values. Results are valid only if DMSWRRAC's result is 0.

Table 26. Service Module Results	
Item	Description
Service module return code	The return code from the service module. These are listed in “Return Codes and Reason Codes” on page 524, primarily for diagnostic purposes. Most code will only need to verify that this value is zero before using the SAF/ESM outputs below. When the service module return code is non-zero, the SAF/ESM outputs in this table are undefined.
Service module reason code	The reason code from the service module. These are listed in “Return Codes and Reason Codes” on page 524, primarily for diagnostic purposes. Most code will use this only for diagnostic purposes, and only when the corresponding return code is non-zero.
SAF return code	Values are documented in the ESM publication's description of the RACROUTE macro with a REQ= value corresponding to the service module. That is, REQ=STAT for class-active tests, REQ=AUTH for authorization tests, and REQ=AUDIT when creating audit log records

Table 26. Service Module Results (continued)	
Item	Description
ESM return code	Values are documented in the same place as the SAF return code.
ESM reason code	Values are documented in the same place as the SAF return code.

DMSWRRAC EXEC Diagnostic Results

The contents of the diagnostic strings are not considered a stable result. The following shows sample code to format the output of DMSWRRAC EXEC:

```
if datatype( o.0diag.0 ) = "NUM" then do
  say "Diagnostics"
  do i = 1 to o.0diag.0
    say "      " o.0diag.i
  end
end
```

Calling Without Using the IBM-Provided REXX EXECs

If the external interfaces do not meet your needs, or are not available for your calling language, you can call the service modules directly using the information in this section. All of the modules follow the same pattern for inputs and outputs.

Each module uses standard CMS linkage, and requires an EPLIST as input. The EPLIST argument string, delimited by EPLARGBG and EPLARGND, consists of a header, offsets to each parameter's starting and ending positions within the argument string, and the value of each parameter. An eye catcher within the header determines the format used to interpret the rest of the argument string, due to different supported-language string termination conventions.

Direct Format Argument String: Eye Catcher VALU

REXX callers must use this format. Assembler callers can use this format or the method described in “Indirect Format Argument String: Eye Catcher ADRL” on page 523. C callers can build this format in a private storage buffer, then build a separate indirect format argument string that points to the direct one.

Note: If you want confirmation that your parameter list is built properly, you can call the service's argument string constructors with the same values and compare its result against your results.

Header

The header consists of the following:

- 4-byte parameter list eye catcher containing the EBCDIC string "VALU"
- 1-byte binary version number. Currently this value must be 01x.
- 1-byte binary field reserved for future use. Its value must be binary zeros.
- 2-byte zero-based offset from the first byte of the eye catcher to the first byte of the parameter offsets structure below.
- 4-byte signed binary number containing the number of parameters supplied in the argument string. There must be one parameter offset entry for each parameter, even if the corresponding parameter value's length is zero.
- 2-byte zero-based offset from the first byte of the eye catcher to the first byte of the parameter values structure. See “Parameter Values” on page 523.
- 2-byte binary field reserved for future use. Its value must be binary zeros.

Parameter Offsets

For each parameter supplied, one entry is produced in the following format. The entries are contiguous in storage, and ordered. Entries are implicitly numbered starting with 1; that is, the first entry describes the first parameter's value.

- 4-byte unsigned zero-based offset from the first byte of the parameter values structure to the first byte of the corresponding parameter's value. See “Parameter Values” on page 523.
- 4-byte unsigned zero-based offset from the first byte of the parameter values structure to the first byte past the corresponding parameter's value. See “Parameter Values” on page 523.

Note that a parameter offset entry whose two offsets are equal describes a parameter value of length zero.

Parameter Values

There is no internal structure to parameter values. The values can occur in the same order as the parameter offset entries, but this is not required. Multiple parameter offset entries can describe (have identical offsets to) a single value, allowing reuse. Values can overlap.

A single parameter's value is described as follows:

- An effective starting address – the address of the argument string, plus the header's offset to the parameter values structure, plus the first byte offset from the corresponding parameter offset entry.
- An effective length – its effective ending address (formed in the same way as the effective starting address, but using the past-last-byte offset), minus its effective starting address.

Note that the value's effective length is zero when the corresponding parameter offset entry contains two equal values.

Indirect Format Argument String: Eye Catcher ADRL

C callers must build a separate direct format argument string in a private storage buffer, then build this indirect argument string pointing to the direct one, and pass the indirect format into `syscall()`. REXX callers cannot use this format, because it requires the user code to know storage addresses. Assembler callers should use the simpler direct format instead.

Descriptor

The descriptor consists of the following:

- 4-byte parameter list eye catcher containing the EBCDIC string "ADRL".
- 8-byte printable address of a buffer containing the direct format argument string.
- 8-byte printable length of the direct format argument string.

For example, if the 31-bit address of the direct format argument string is 0x00000001, then the address field will contain the eight characters 00000001, or F0F0F0F0F0F0F0F1 in hexadecimal. `rAcaddr.C` provides a mapping of both parameter list formats along with the necessary code to call `RACROUTE REQUEST=STAT` for the FACILITY class. To download `rAcaddr.C`, go to [z/VM Downloads](https://www.vm.ibm.com/download/) (<https://www.vm.ibm.com/download/>).

Example

In the following example, output from DMSWRESM is used as input to DMSWSESM. Suppose the DMSWSESM result is the following:

```
VMRACROUTE STAT DMSWSESM VALU          ;      ;      VMRACROUTE STAT TSTSTAT
TSTSTAT
FACILITY
```

In hexadecimal, the information above is:

The section above that is input to DMSWSESM (printed normally, 1 character per byte) is:

The same information (printed as hexadecimal, with two visible characters per byte) is:

Return Codes and Reason Codes

Return Code	Reason Code	Description
0	0	The module invoked RACROUTE, and RACROUTE's results must be retrieved using EventRetrieve. Their format is documented later.
4	<i>reason</i>	The module received return code vm_evn_warning from the EventSignal service. <i>reason</i> is the corresponding EventSignal reason code. The VMREXMTR COPY file on the z/VM system CMS disk documents the reason code values.
8	<i>reason</i>	The module received return code vm_evn_error from the EventSignal service. <i>reason</i> is the corresponding EventSignal reason code. The VMREXMTR COPY file on the z/VM system CMS disk documents the reason code values.
90	0	No ESM is installed on the system.
100	<i>reason</i>	The header portion of the argument string is not valid. The <i>reason</i> values are shown below.
<i>n</i> 000	<i>reason</i>	Parameter number <i>n</i> is not valid. <i>n</i> =1000 for parameter 1, 2000 for parameter 2, and so on. The <i>reason</i> values are shown below.

Header Not Valid		
Return Code	Reason Code	Description
100	1	The argument string is too small to be valid as a header.
100	2	The header's version number is smaller than the minimum that the module can process.
100	3	The header's version number is larger than the maximum that the module can process.
100	4	The header's reserved fields are non-zero.

Header Not Valid <i>(continued)</i>		
Return Code	Reason Code	Description
100	5	The caller supplied too few parameters, based on the parameter list's version number.
100	6	The header's "offset to first parameter offset" field is too large, based on the size of the argument string and the number of parameters expected.
100	7	The header's "offset to first value offset" field is too large, based on the size of the argument string and the number of parameters expected.
100	8	The header's eye catcher is not a valid value.

Return Codes and Reason Codes when a Parameter is Not Valid

Divide the return code by 1000 decimal to determine which parameter is not valid; $n=1000$ for parameter 1, 2000 for parameter 2, and so on.

Parameter Not Valid		
Return Code	Reason Code	Description
$n000$	1	Parameter is missing, and it is required.
$n000$	2	Parameter value extends past the end of the argument string.
$n000$	3	Parameter value length is negative.
$n000$	4	Parameter value is shorter than the minimum required.
$n000$	5	Parameter value is longer than the maximum permitted.
$n000$	6	Parameter value must be one of a required list, but it is not.
$n000$	7	Parameter value begins before the start of the argument string

RACROUTE Macro Request Outputs

When the RACROUTE module's return code is zero, additional outputs from the RACROUTE request are available, as shown in Table 26 on page 521. The caller retrieves them by invoking the CMS multitasking API EventRetrieve using the event name that IBM documents for that module. For example, use VMRACTESTAT when testing whether or not a class is active via RACROUTE REQ=STAT.

Offset (Decimal)	Offset (Hexadecimal)	Length (Decimal Bytes)	Description
00	00	4	Size of the stable output area, in bytes
04	04	4	Module return code; see “Return Codes and Reason Codes” on page 524
08	08	4	Module return code; see “Return Codes and Reason Codes” on page 524
12	0C	40	Printable diagnostic message

Offset (Decimal)	Offset (Hexadecimal)	Length (Decimal Bytes)	Description
52	34	4	SAF return code
56	38	4	ESM return code
60	3C	4	ESM return code
64	40	4	Size in bytes of the ESM log entry, if any
68	44	varies	Diagnostic data that might be requested by IBM service. Its format is undocumented.

You can use the DMSWRRAC EXEC on MAINT 193 to see an example of parsing the EventRetrieve output. The following sample shows an outline of the necessary code, in REXX. Observe DMSWRRAC EXEC on a live system to be sure that you see changes due to service. You can also invoke DMSWRRAC EXEC yourself and print its diagnostic output to compare against your code, if necessary.

```

data_buffer_length = 4096
eventNumber = 1
do i = 1 to event_flag.0 /* set by EventWait */
  if event_flag.i >= 0 then do
    data_buffer_length = event_flag.i
    eventNumber = i
  end
end

call CSL 'EventRetrieve retcode reascode g.0eventMonitor',
        'eventNumber data_buffer data_buffer_length event_data_length'
if retcode > 0 then do
  signal ReturnToCaller
end

g.0eventData = substr(data_buffer,1,event_data_length)
parse var g.0eventData mob      +4 modrc +4 modrs +4 moddiag +40 ,
          safrc +4 esmrc +4 esmrs +4 loglen +4 extra

```

Appendix B. VSE Macros

CMS simulates programming interfaces defined by the VSE operating system. The material that follows documents CMS simulation of these programming interfaces. For information about these interfaces, see the appropriate VSE book.

This appendix lists the VSE macros that CMS supports, including:

- VSE assembler language macros
- VSE supervisor macros
- VSE declarative macros
- VSE imperative macros.

For more information on VSE macros, see the [z/VM: CMS Application Development Guide for Assembler](#).

VSE Assembler Language Macros Supported

Table 27 on page 527 lists the VSE assembler language macros supported by CMS/DOS. You can assemble source programs that contain these macros under CMS/DOS, provided that you have the macros available in either your own or a shared CMS macro library. The macros whose functions are described in the *Function* column with the term *no-op* are supported for assembly only; when you run programs that contain these macros, the VSE functions are not performed. To accomplish the macro function you must run the program on a real VSE system.

Table 27. VSE Macros Supported by CMS		
Macro Name	SVC Number	Function
CALL		Pass control to another program
CANCEL	06	End processing
CDLOAD	65	Load a VSAM phase
CHECK		Verify completion of a read or write operation
CLOSE/ CLOSER		Deactivate a data file
CNTRL		Control a physical device
COMRG	33	Return address of background partition communication region
DEQ	41	no-op
DTFxx		Establish file definitions
DUMP		Dump storage and registers and end processing
ENQ	42	no-op
EOJ	14	End processing normally
ERET		Provide an error routine
EXCP	00	Process a channel program
EXIT PC	17	Return from program check routine
EXIT AB	95	Return from abnormal termination routine
EXTRACT	98	Retrieve PUB, storage boundaries, or CPUID information
FCEPGOUT	86	no-op
FETCH	01	Load and pass control to a phase
FETCH	02	Load and pass control to a logical transient

Table 27. VSE Macros Supported by CMS (continued)

Macro Name	SVC Number	Function
FREE	36	no-op
FREEVIS	62	Release user free storage
GENL		Generate a phase directory list
GET		Access a sequential file
GETFLD/ MODFLD	107	Provide macro interface support for system information retrieval.
GETVCE	99	Return requested device information to output area.
GETVIS	61	Obtain user free storage
GETIME	34	Get the time of day
JDUMP		Dump storage and registers and end processing
LOAD	04	Load a phase into storage
LOCK/ UNLOCK	110	Resource control
MVCOM	05	Modify bytes in the partition communication region
NOTE		Manage data set access
OPEN/ OPENR		Activate a data file
PAGEIN	87	no-op
PDUMP		Dump storage and registers and continue processing
PFIX	67	no-op
PFREE	68	no-op
POINTR		Position a file for reading
POINTS		Reposition a file to its beginning
POINTW		Position a file for writing
POST	40	Post the event control block
PRTOV		Control printer overflow
PUT		Write to a sequential file
PUTR		Communicate with the system operator
READ		Access a sequential file
RELPAG	85	Simulates releasing pages by setting them to binary zeros.
RELSE		Skip to begin reading next block
RETURN		Return control to calling program
RUNMODE	66	Check if program is running real or virtual
SECTVAL	75	Obtain a sector number
SETIME	10/24	no-op
SETPFA	71	no-op
STXIT AB	37	Provide or end linkage to abnormal ending routine
STXIT PC	16	Provide or end linkage to program check routine
STXIT IT	18	no-op
STXIT OC	20	no-op
SUBSID	105	Retrieve information on supervisor subsystem
TRUNC		Skip to begin writing next block

Table 27. VSE Macros Supported by CMS (continued)		
Macro Name	SVC Number	Function
TTIMER	52	Return a 0 in Register 0 (effectively a no-op)
WAIT	07	Wait for the event completion
WRITE		Write to a sequential file
xxMOD		Create Logical IOCS routine inline

VSE Supervisor and I/O Macros Supported by CMS/DOS

CMS/DOS supports the VSE Supervisor macros and the SAM and VSAM I/O macros to the extent necessary to run the DOS/VS COBOL Compiler, the DOS PL/I Optimizing Compiler, and DOS/VS RPG II Compiler under CMS/DOS. CMS/DOS supports VSE Supervisor macros described in the publication *VSE Macro Reference*.

Because CMS is a single-user system executing in a virtual machine with virtual storage, VSE operations, such as multitasking, that cannot be simulated in CMS are ignored.

The following information deals with the type of support that CMS/DOS provides in the simulation of VSE Supervisor and Sequential Access Method I/O macros.

Supervisor Macros

CMS/DOS supports physical IOCS macros and control program function macros for VSE. [Table 28 on page 529](#) lists the physical IOCS macros and describes their support. [Table 29 on page 529](#) lists the control program function macros and their support.

Table 28. Physical IOCS Macros Supported by CMS/DOS	
Macro	Support
CCB (command control block)	The CCB is generated. CCW=FORMAT1 is supported only for I/O to the console or to OS or DOS formatted DASD.
IORB (input/output request block)	The IORB is generated. IOFLAG=FORMAT1 is supported only for I/O to the console or to OS or DOS formatted DASD.
EXCP (process channel program)	The REAL operand is not supported. All other operands are supported.
WAIT	Supported. Issued whenever your program requires an I/O operation (started by an EXCP macro) to be completed before execution of program continues.
SECTVAL (sector value)	Supported for VSAM.
OPEN/OPENR	Supported. Activates a data file.
LBRET (label processing return)	Return to the \$\$\$B-transient after an SVC 8 was issued to give control to the problem program.
FEOV (forced end of volume)	Not supported.
SEOV (system end of volume)	Not supported.
CLOSE/CLOSER	Supported. Deactivates a data file.

Table 29. SVC Support Routines and Their Operation		
Function/Macro	SVC No. Dec Hex	Support
EXCP	0 0	Used to start an I/O operation on a device in the CMS/DOS environment.

<i>Table 29. SVC Support Routines and Their Operation (continued)</i>		
Function/Macro	SVC No. Dec Hex	Support
FETCH	1 1	Used to bring a problem program phase into user storage and to start execution of the phase if the phase was found. Operand SYS=YES is not supported.
FETCH	2 2	Used to bring a \$\$B-transient phase into the CMS transient area (or if the phase is in the CMSDOS segment, not to load it), and start execution of the phase if the phase was found. Operand SYS=YES is not supported.
FORCE DEQUEUE	3 3	Not supported. See note “2” on page 537.
LOAD	4 4	Used to bring a problem program phase into user storage, and return the caller the entry point address of the phase just loaded. Operand SYS=YES is not supported.
MVCOM	5 5	Provides the user with a means of altering positions 12 through 23 of the partition communications region (BGCOM).
CANCEL	6 6	Cancels a VSE session either by a VSE program request or by a request from any of the CMS routines handling CMS/DOS.
WAIT	7 7	Used to wait on a CCB, IORB, ECB, or TECB. (Note that CMS/DOS does not support ECBs or TECBs). CCBs are always posted by the DMSXCP routine before returning to the caller. The WAIT support under CMS/DOS will effectively be a branch to the CMS/DOS POST routine.
CONTROL	8 8	Temporarily return control from a \$\$B-transient to the problem program.
LBRET	9 9	Return to the \$\$B-transient after an SVC 8 was issued to give control to the problem program.
SET TIMER	10 A	No operation. Successful return code of 0 is given in R15. See “1” on page 537.
TRANS. RETURN	11 B	Return from a \$\$B-transient to the calling problem program.
JOB CONTROL ‘AND’	12 C	Resets flags to 0 in the linkage control byte in BGCOM (communication region). If R1 = 0, bit 5 of JCSW4 (COMREG byte 59) is turned off.
JC FLAGS	13 D	Not supported. See note 2.

<i>Table 29. SVC Support Routines and Their Operation (continued)</i>		
Function/Macro	SVC No. Dec Hex	Support
EOJ	14 E	Normally terminates execution of a problem program.
SYSIO	15 F	Not supported. See note 2.
PC STXIT	16 10	Establish or end linkage to a user's program check routine.
PC EXIT	17 11	Used to provide supervisory support for the EXIT macro. SVC 17 provides a return from the user's PC routine to the next sequential instruction in the program that was interrupted because of a program check.
IT STXIT	18 12	No operation. Successful return code of 0 is given in R15. See note 1.
IT EXIT	19 13	Not supported. See note 2.
OC STXIT	20 14	No operation. Successful return code of 0 is given in R15. See note 1.
OC EXIT	21 15	Not supported. See note 2.
SEIZE	22 16	No operation. Successful return code of 0 is given in R15. See note 1.
LOAD HEADER	23 17	Not supported. See note 2.
SETIME	24 18	No operation. Successful return code of 0 is given in R15. See note 1.
HALT I/O	25 19	Not supported. See note 2.
Validate address limits.	26 1A	The upper address must be specified in general register 2 and the lower address must be specified in general purpose register 1.
TP HALT I/O	27 1B	Not supported. See note 2.
MR EXIT	28 1C	Not supported. See note 2.
WAITM	29 1D	Not supported. See note 2.
QWAIT	30 1E	Not supported. See note 2.
QPOST	31 1F	Not supported. See note 2.
	32 20	Reserved
COMRG	33 21	Used to provide the caller with the address of the partition communications region. DMSDOS provides the caller with the address of the partition communications region, in the user's register 1.

Table 29. SVC Support Routines and Their Operation (continued)		
Function/Macro	SVC No. Dec Hex	Support
GETIME	34 24	Provides support for the GETIME macro. SVC 34 updates the date field in the communications region. The GMT operand is not supported.
HOLD	35 23	No operation. Successful return code of 0 is given in R15. See note 1.
FREE	36 24	No operation. Successful return code of 0 is given in R15. See note 1.
AB STXIT	37 25	Establish or end linkage to a user's abnormal end routine. Supported for OPTION=DUMP or NODUMP.
ATTACH	38 26	Not supported. See note 2.
DETACH	39 27	Not supported. See note 2.
POST	40 28	Used to post an ECB, IORB, TECB, or CCB. Byte 2, bit 0 of the specified control block is turned 'on' by DMSDOS.
DEQ	41 29	No operation. Successful return code of 0 is given in R15. See note 1.
ENQ	42 2A	No operation. Successful return code of 0 is given in R15. See note 1.
	43 2B	Reserved
UNIT CHECKS	44 2C	Not supported. See note 2.
EMULATOR INTERF.	45 2D	Not supported. See note 2.
OLTEP	46 2E	Not supported. See note 2.
WAITF	47 2F	Not supported. See note 2.
CRT TRANS	48 30	Not supported. See note 2.
CHANNEL PROG.	49 31	Not supported. See note 2.
LIOCS DIAG.	50 32	Issued by a logical IOCS routine when the LIOCS is called to perform an operation that the LIOCS was not generated to perform. The error message <i>unsupported function in a LIOCS routine</i> is issued, and the session is then terminated.
RETURN HEADER	51 33	Not supported. See note 2.
TTIMER	52 34	No operation. Successful return code of 0 is given in R15. See note 1. R0 is also cleared.
VTAM® EXIT	53 35	Not supported. See note 2.
FREEREL	54 36	Not supported. See note 2.
GETREAL	55 37	Not supported. See note 2.

<i>Table 29. SVC Support Routines and Their Operation (continued)</i>		
Function/Macro	SVC No. Dec Hex	Support
POWER	56 38	Not supported. See note 2.
POWER	57 39	Not supported. See note 2.
SUPVR. INTERF.	58 3A	Not supported. See note 2.
EOJ INTERF.	59 3B	Not supported. See note 2.
GETADR	60 3C	Not supported. See note 2.
GETVIS	61 3D	Used to obtain free storage for scratch use or for obtaining an area where a relocatable program may be loaded. The POOL and SVA GETVIS options are ignored. The PAGE option is ignored for requests of less than or equal to 2K bytes of storage. LOC=RES is treated the same as LOC=BELOW.
FREEVIS	62 3E	Used to return the free storage obtained through an earlier GETVIS call.
USE	63 3F	The USE/RELEASE function has been replaced by SVC 110 (LOCK/UNLOCK) for serially controlling system resources. All SVC 63 and 64 requests are mapped into SVC 110 requests respectively. Return codes previously associated with USE/RELEASE under CMS/DOS are maintained.
RELEASE	64 40	Reference SVC 63.
CDLOAD	65 41	Used to load a relocatable VSAM phase into storage, unless the program has already been loaded.
RUNMODE	66 42	Used by a problem program to find out if the program is running in real or virtual mode. The caller's register 0 is zeroed to indicate that the program is running in virtual mode.
PFIX	67 43	No operation. Successful return code of 0 is given in R15. See note 1.
PFREE	68 44	No operation. Successful return code of 0 is given in R15. See note 1.
REALAD	69 45	Not supported. See note 2.
VIRTAD	70 46	Not supported. See note 2.
SETPFA	71 47	No operation. Successful return code of 0 is given in R15. See note 1.
GETCBUF/ FREECBUF	72 48	Not supported. See note 2.
SETAPP	73 49	Not supported. See note 2.
PAGE FIX	74 4A	Not supported. See note 2.

<i>Table 29. SVC Support Routines and Their Operation (continued)</i>		
Function/Macro	SVC No. Dec Hex	Support
SECTVAL	75 4B	Used by I/O routines to obtain a sector number for a CKD or ECKD™ device.
SYSREC	76 4C	Not supported. See note 2.
TRANSCCW	77 4D	Not supported. See note 2.
CHAP	78 4E	Not supported. See note 2.
SYNCH	79 4F	Not supported. See note 2.
SETT	80 50	Not supported. See note 2.
TESTT	81 51	Not supported. See note 2.
LINKAGE	82 52	Not supported. See note 2.
ALLOCATE	83 53	Not supported. See note 2.
SET LIMIT	84 54	Not supported. See note 2.
RELPAG	85 55	<p>Provides support for the RELPAG macro. At entry register 1 points to a list of 8-byte storage description area. Each entry contains the beginning address and the length-1 of an area to be released. A nonzero byte following an entry indicates the end of the list. An area is released only if it contains at least a full CP page (4K bytes). CMS simulates the release of pages by setting them to binary zeros when the virtual machine calls CP with DIAGNOSE code X'10'. On return, R15 holds return code as follows:</p> <p>R15 = 0 all areas have been released.</p> <p>R15 = 2 one or more negative area lengths were specified.</p> <p>R15 = 4 one or more pages to be released were outside the user storage area.</p> <p>R15 =16 at least one entry contains a beginning address outside the user storage area.</p>
FCEPGOUT	86 56	No operation. Successful return code of 0 is given in R15. See note 1.
PAGEIN	87 57	No operation. Successful return code of 0 is given in R15. See note 1.
TPIN	88 58	Not supported. See note 2.
TPOUT	89 59	Not supported. See note 2.
PUTACCT	90 5A	Not supported. See note 2.
POWER	91 5B	Not supported. See note 2.

<i>Table 29. SVC Support Routines and Their Operation (continued)</i>		
Function/Macro	SVC No. Dec Hex	Support
XECBTAB	92 5C	Not supported. See note 2.
XPOST	93 5D	Not supported. See note 2.
XWAIT	94 5E	Not supported. See note 2.
AB EXIT	95 5F	Exit from abnormal task termination routine and continue the task.
TT EXIT	96 60	Not supported. See note 2.
TT STXIT	97 61	Not supported. See note 2.
EXTRACT	98 62	Support for EXTRACT macro of VSE. The caller requests PUB information, CPUID, or storage boundary information. Register 1 on entry points to a parameter list. Output is placed in an area provided by caller.
GETVCE	99 63	Caller requests device information about specific DASD. Information is returned in an output area pointed to from the parameter list. Register 1 contains a pointer to the parameter list on entry.
	100 64	Reserved
MODVCE	101 65	No operation. Successful return code of 0 is given in R15. See note 1.
	102 66	Reserved.
SYSFIL	103 67	Not supported. See note 2.
EXTENT	104 68	No operation. Successful return code of 0 is given in R15. See note 1.
SUBSID	105 69	SUBSID.. the 'INQUIRY' function is supported for the supervisor subsystem. Information returned is described by the SUPSSID control block. The SUBSID 'NOTIFY' and 'REMOVE' functions are not supported.
LINKAGE	106 6A	Not supported. See note 2.

Table 29. SVC Support Routines and Their Operation (continued)

Function/Macro	SVC No. Dec Hex	Support
TASK INTERF.	107 6B	<p>Provides macro interface support for system information retrieval. The parameters supported are:</p> <p>GETFLD:</p> <p>field=ppsavar returns problem program save area address.</p> <p>=savar returns current save area address.</p> <p>=maintask returns maintask TID in R1.</p> <p>=aclose returns in R1: 1 if in process, 0 if not.</p> <p>=pcexit returns the pcexit routine address and save area in R0 and R1 respectively. If the exit routine is currently active, bit 0 in R0 is set ON. If no exit is defined, it returns a 0 in both R0 and R1.</p> <p>MODFLD:</p> <p>field=vsamopen set bit X'08' in tcbflags byte if R1=0</p> <p>=aclose set bit X'10' in tcbflags byte if R1=0</p> <p>The MODFLD requests for fields CNCLALL and OPENSVA are treated as a NOP with a return code of 0.</p> <p>All other SVC 107 macro calls are unsupported. The error message DMSGMF121S is issued and the request is canceled. See note 2.</p>
DATA SECURE	108 6C	Not supported. See note 2.
PAGESTAT	109 6D	Not supported. See note 2.
LOCK/UNLOCK	110 6E	<p>Used by VSAM to control access to resources. Access is maintained in either a 'shared' or 'exclusive' control environment. When DOS is SET ON, counters are maintained as well as the type of control for each resource in a table (LOCKTAB) built in free storage. All entries not unlocked by the program are cleared at both normal and abnormal end-of-job. All requests for resource control are passed to SVC 110 through the DTL macro (define the lock). SVC 63 and 64 requests are mapped into a dummy DTL and processed by SVC 110.</p>

Note:

1. No operation: In each case, register 15 is cleared to simulate successful operation, and all other registers are returned unchanged, unless otherwise noted.
2. Not supported: For unsupported SVCs, an error message is given, and the SVC is treated as a *cancel*.

Declarative Macros (Sequential Access Method I/O Macros)

CMS/DOS supports the following declarative macros:

- DTFCF - Types X'02' and X'04'
- DTFCN - Types X'03'
- DTFDI - Types X'33'
- DTFMT - Types X'10', X'11', X'12', and X'14'
- DTFPR - Types X'08'
- DTFSD - Types X'20'.

The CDMOD, DIMOD, MTMOD, and PRMOD macros generate the logical IOCS routines that correspond to the declarative macros. For files on disk, the logical IOCS routines used during program execution reside in the CMSBAM DCSS and are not generated within the program. The operands that CMS/DOS supports for the DTF are also supported for the xxMOD macro. In addition, CMS/DOS supports three internal macros that the COBOL and PL/I compilers require: DTFCP (types X'31' and X'32'), CPMOD, and DTFSL.

DTFCD Macro - Defines the File for a Card Reader

CMS/DOS does not support the ASOCFLE, FUNC, TYPEFILE=CMBND, and OUBLKSZ operands of the DTFCD macro. CMS/DOS ignores the SSELECT operand and any mode other than MODE=E. [Table 30 on page 537](#) describes the DTFCD macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Table 30. CMS/DOS Support of DTFCD Macro		
Operand	Status	Description
DEVADDR=SYSxxx		Symbolic unit for reader-punch used for this file.
IOAREA1=xxxxxxxx	*	Name of the first I/O area.
ASOCFLE=xxxxxxxx	*	Not supported.
BLKSIZE=nnn	*	Length of one I/O area, in bytes. If omitted, 80 is assumed. If CTLCHR=YES is specified, BLKSIZE defaults to 81.
CONTROL=YES		CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501.
CRDERR=RETRY	*	Retry if punching error is detected. Applies to 2520 and 2540 only. However, this situation is never encountered under CMS/DOS because hardware errors are not passed to the LIOCS module.
CTLCHR=xxx		(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file.
DEVICE=nnnn	*	(2501, 2520, 2540, 3505, or 3525). If omitted, 2540 is default.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.

Table 30. CMS/DOS Support of DTFCN Macro (continued)		
Operand	Status	Description
ERROPT=xxxxxx	*	IGNORE, SKIP, or name. Applies to 3505 and 3525 only.
FUNC=xxx	*	Not supported.
IOAREA2=xxxxxxxx	*	If two output areas are used, name of second area.
IOREG=(nn)		Register number if two I/O areas are used and GET or PUT does not specify a work area. Omit WORKA.
MODE=xx	*	Only MODE=E is supported.
MODNAME=xxxxxxxx		Name of the logic module that is used with the DTF table to process the file.
OUBLKSZ=nn	*	Not supported.
RDONLY=YES	*	Causes a read-only module to be generated.
RECFORM=xxxxxx		(FIXUNB, VARUNB, UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)	*	Register number if RECFORM=UNDEF.
SEPASMB=YES		DTFCN is to be assembled separately.
SSELECT=n	*	Ignored.
TYPEFLE=	*	Input or output.
WORKA=YES		I/O records are processed in work areas instead of the I/O areas.

DTFCN Macro - Defines the File for a Console

CMS/DOS supports all of the operands of the DTFCN macro. Table 31 on page 538 describes the operands of the DTFCN macro and their support under CMS/DOS. The status column is blank because the CMS/DOS and VSE support of DTFCN are the same.

Table 31. CMS/DOS Support of DTFCN macro		
Operand	Status	Description
DEVADDR=SYSxxx		Symbolic unit for the console used for this file.
IOAREA1=xxxxxxxx		Name of I/O area.
BLKSIZE=nnn		Length in bytes of I/O area (for PUTR macro usage, length of output part of I/O area). If RECFORM=UNDEF, maximum is 256. If omitted, 80 is default.
INPSIZE=nnn		Length in bytes for input part of I/O area for PUTR macro usage.
MODNAME=xxxxxxxx		Logic module name for this DTF. If omitted, IOCS generates a standard name. The logic module is generated as part of the DTF.
RECFORM=xxxxxx		(FIXUNB or UNDEF). If omitted, FIXUNB is default.

Table 31. CMS/DOS Support of DTFCN macro (continued)		
Operand	Status	Description
RECSIZE=(nn)		Register number if RECFORM=UNDEF. General purpose registers 2 through 12, enclosed in parentheses.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or CMBND). Input processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is default.
WORKA=YES		GET or PUT specifies work area.

DTFDI MACRO - Defines the File for Device Independence for System Logical Units

CMS/DOS supports most operands of the DTFDI macro. Table 32 on page 539 describes the operands of the DTFDI macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Table 32. CMS/DOS Support of DTFDI Macro		
Operand	Status	Description
DEVADDR=SYSxxx		(SYSIPT, SYSLST, SYSPCH, or SYSRDR). System logical unit. CMS/DOS issues an error message if the logical unit specified on the DTF does not match the logical unit specified on the corresponding DLBL command.
IOAREA1=xxxxxxxx		Name of the first I/O area.
CISIZE=n	*	This operand specifies the control interval size for a DOS formatted FB-512 device assigned to a nonsystem file logical unit. This operand is ignored for count-key-data devices and CMS formatted disks.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.
FBA=YES		This operand is not required and is ignored if specified.
ERROPT=xxxxxxxx		(IGNORE, SKIP, or name of your error routine). Prevents termination on errors.
IOAREA2=xxxxxxxx		If two I/O areas are used, name of second area.
IOREG2=(nn)		Register number. If omitted and two I/O areas are used, register 2 is default. General purpose registers 2 through 12, enclosed in parentheses.
MODNAME=xxxxxxxx		DIMOD name for this DTF. If omitted, IOCS generates a standard name. This operand is ignored with DASD. The SAM OPEN routines within the CMSBAM DCSS always load an IBM supplied logic module and link it to the DTF.
RDONLY=YES		Generates a read-only module. Requires a module save area for each routine using the module.
RECSIZE=nnn		Number of characters in record. Default values: 121 (SYSLST), 81 (SYSPCH), 80 (other).
SEPASMB=YES		DTFDI to be assembled separately.

Table 32. CMS/DOS Support of DTFDI Macro (continued)		
Operand	Status	Description
TRC=YES	*	Not supported.
WLRERR=xxxxxxx		Name of your wrong-length record routine.

DTFMT Macro - Defines the File for a Magnetic Tape

CMS/DOS does not support the ASCII, BUFOFF, HDRINFO, LENCHK, and READ=BACK operands of the DTFMT macro. Tape I/O operations are limited to reading in the forward direction.

You may use the FILABL operand in the DTFMT macro to specify that you have a standard tape label file, a nonstandard tape label file, or an unlabeled tape. The type of tape label processing depends on the option selected.

Table 33 on page 540 describes the DTFMT macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

Table 33. CMS/DOS Support of DTFMT Macro		
Operand	Status	Description
BLKSIZE=nnnnn		Length of one I/O area in bytes (maximum = 32,767.
DEVADDR=SYSxxx		Symbolic unit for tape drive used for this file.
EOFADDR=xxxxxxx		Name of your end-of-file routine.
FILABL=xxxx		(NO, STD, or NSTD). If NSTD specified, include LABADDR.
IOAREA1=xxxxxxx		Name of first I/O area.
ASCII=YES	*	Not supported.
BUFOFF=nn	*	Not supported.
CKPTREC=YES		Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records.
ERREXT=YES		Additional errors and ERET are desired.
ERROPT=xxxxxxx		(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
HDRINFO=YES	*	Not supported.
IOAREA2=xxxxxxx		If two I/O areas are used, the name of the second area.
IOREG=(nn)		Register number. Use only if GET or PUT does not specify a work area or if two I/O areas are used. Omit WORKA. General purpose registers 2 through 12, enclosed in parentheses.
LABADDR=xxxxxxx		Name of your label routine if FILABL=NSTD or if FILABL=STD and user-standard labels are processed.
LENCHK=YES	*	Not supported.
MODNAME=xxxxxxx		Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name.

<i>Table 33. CMS/DOS Support of DTFMT Macro (continued)</i>		
Operand	Status	Description
NOTEPNT=xxxxxx		(YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro is used. POINTS if only POINTS macro is used.
RDONLY=YES		Generate read-only module. Requires a module save area for each routine using the module.
READ=xxxxxxx	*	CMS/DOS only supports READ=FORWARD.
RECFORM=xxxxxx		(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed.
RECSIZE=nnnn		If RECFORM=FIXBLK, number of characters in the record. If RECFORM=UNDEF, register number. Not required for other records. General purpose registers 2 through 12, enclosed in parentheses.
REWIND=xxxxxx		(UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only.
SEPASMB=YES		DTFMT is to be assembled separately.
TPMARK=NO		Prevent writing a tapemark ahead of data records if FILABL=NSTD or NO.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or WORK). If omitted, INPUT is default.
VARBLD=(nn)		Register number, if RECFORM=VARBLK and records are built in the output area. General purpose registers 2 through 12 are enclosed in parentheses.
WLRERR=xxxxxxxx		Name of wrong-length record routine.
WORKA=YES		GET or PUT specifies a work area. Omit IOREG.

DTFPR Macro - Defines the File for a Printer

CMS/DOS does not support the ASOCFLE, ERROPT=IGNORE, and FUNC operands of the DTFPR macro. Table 34 on page 541 describes the operands of the DTFPR macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

<i>Table 34. CMS/DOS Support of DTFPR Macro</i>		
Operand	Status	Description
DEVADDR=SYSxxx		Symbolic unit for the printer used for this file.
IOAREA1=xxxxxxxx		Name for the first output area.
ASOCFLE=xxxxxxxx	*	Not supported.
BLKSIZE=nnn	*	Length of one output area, in bytes. If omitted, 121 is default.
CONTROL=YES		CNTRL macro used for this file. Omit CTLCHR for this file.

<i>Table 34. CMS/DOS Support of DTFPR Macro (continued)</i>		
Operand	Status	Description
CTLCHR=xxx		(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file.
DEVICE=nnnn	*	(1403, 1443, 3203, or 3211). If omitted, 1403 is default.
ERROPT=xxxxxxxx	*	RETRY or the name of your error routine for 3211. Not allowed for other devices. IGNORE is not supported.
FUNC=xxxx	*	Not supported.
IOAREA2=xxxxxxxx		If two output areas are used, name of second area.
IOREG=(nn)		Register number; if two output areas used and GET or PUT does not specify a work area. Omit WORKA.
MODNAME=xxxxxxxx		Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name.
PRINTOV=YES		PRTOV macro used for this file.
RDONLY=YES		Generate a read-only module. Requires a module save area for each routine using the module.
RECFORM=xxxxxx		(FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)		Register number if RECFORM=UNDEF.
SEPASMB=YES		DTFPR is to be assembled separately.
STLIST=YES		Use 1403 selective tape listing feature.
TRC=YES	*	Not supported.
UCS=xxx		(ON) process data checks. (OFF) ignores data checks. Only for printers with the UCS feature or 3203 or 3211. If omitted, OFF is default.
WORKA=YES		PUT specifies work area. Omit IOREG.

DTFSD Macro - Defines the File for a Sequential DASD

CMS/DOS does not support the FEOVD, HOLD, and LABADDR operands of the DTFSD macro. [Table 35 on page 542](#) describes the operands of the DTFSD macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from VSE support.

<i>Table 35. CMS/DOS Support of DTFSD Macro</i>		
Operand	Status	Description
BLKSIZE=nnnn		Length of one I/O area, in bytes.
CISIZE=n	*	This operand specifies the control interval size for a DOS formatted FB-512 device assigned to a nonsystem file logical unit. This operand is ignored for count-key-data devices and CMS formatted disks.
EOfADDR=xxxxxxxx		Name of your end-of-file routine.

Table 35. CMS/DOS Support of DTFSD Macro (continued)

Operand	Status	Description
IOAREA1=xxxxxxxx		Name of first I/O area.
CONTROL=YES		This operand is ignored. CONTROL=YES is always included.
DELETFL=NO	*	If DELETFL=NO is specified, the work file is not erased. Otherwise, when the work file is closed, CMS/DOS erases it.
DEVADDR=SYSnnn	*	Symbolic unit. This operand is optional. If DEVADDR is not specified, all I/O requests are directed to the logical unit identified on the corresponding CMS/DOS DLBL command. If a valid logical unit is specified with the DEVADDR operand of the DTF and a different, but also valid, logical unit is specified on the DLBL command, the unit specified on the DLBL command overrides the unit specified in the DTF. However, CMS/DOS issues an error message if a valid logical unit is specified in the DTF and no logical unit is specified on the corresponding DLBL command.
DEVICE=nnnn	*	This operand is ignored. The actual device type is determined by OPEN.
ERREXT=YES		Additional error facilities and ERET are desired. This operand is ignored. ERREXT=YES is always included.
ERROPT=xxxxxxxx		(IGNORE, SKIP, or name of error routine.) Prevents job termination on error records. Do not use SKIP for output files.
FEOVD=YES	*	Not supported.
HOLD=YES	*	Not supported. HOLD=YES is specified for DTFSD update or work files to provide a track hold capability. However, the CMS/DOS open routine sets the track hold bit off and bypasses track hold processing.
IOAREA2=xxxxxxxx		If two I/O areas are used, name of second area.
IOREG=(nn)		Register number. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA.
LABADDR=xxxxxxxx	*	Not supported.
MODNAME=xxxxxxxx		This operand is not required. If specified, it is ignored. The SAM OPEN routines within the CMSBAM DCSS always load an IBM supplied logic module and link it to the DTF.
NOTEPNT=xxxxxxxx		Indicates that NOTE, POINTR, POINTW, and POINTS are used. This operand is ignored. NOTEPNT=YES is always included.
RDONLY=YES		This operand is not required and is ignored if specified. RDONLY=YES is always included.

Table 35. CMS/DOS Support of DTFSD Macro (continued)		
Operand	Status	Description
PWRITE=YES	*	For a DOS formatted FB-512 disk, this operand specifies that for output operations a physical write occurs for every logical block. This operand is ignored for count-key-data devices and CMS formatted disks. DOS formatted FB-512 disks are not supported for output.
RECFORM=xxxxxx		(FIXUNB, FIXBLK, VARUNB, SPNUNB, SPNBLK, VARBLK, or UNDEF). If omitted, FIXUNB is assumed. For work files, use FIXUNB or UNDEF. Although work files contain fixed-length unblocked records, the CMS file system handles work UNDEF files as variable-length record files. If you specify FIXBLK, VARBLK, or UNDEF when creating a CMS file on a CMS disk, CMS writes the file in variable-length format. The LISTFILE command would show the file as V format. If you specify FIXUNB when creating a CMS file on a CMS disk, CMS writes the file in fixed-length format.
RECSIZE=nnnnn		If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records.
SEPASMB=YES		DTFSD is to be assembled separately.
TRUNCS=YES		RECFORM=FIXBLK or TRUNC macro used for this file.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed.
UPDATE=YES		Input file or work file is to be updated.
VARBLD=(nn)		Register number if RECFORM=VARBLK and records are built in the output area. Omit if WORKA=YES.
VERIFY=YES		Check disk records after they are written.
WLRERR=xxxxxxxx		Name of your wrong-length record routine.
WORKA=YES		GET or PUT specifies work area. Omit IOREG. Required for RECFORM=SPNUNB or SPNBLK.

Imperative Macros (Sequential Access Method I/O Macros)

CMS/DOS supports the following imperative macros:

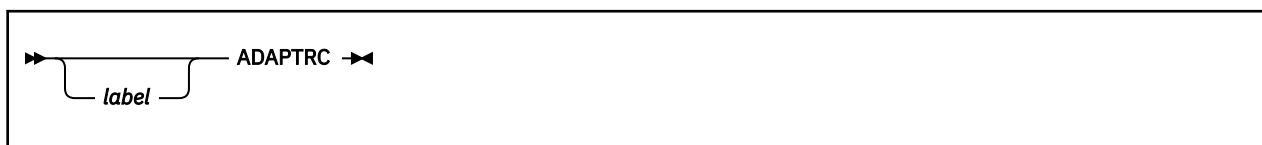
- **Initialization macros:** OPEN and OPENR
- **Processing macros:** GET, PUT, PUTR, RELSE, TRUNC, CNTRL, ERET, and PRTOV.
Note: No code is generated for the CHNG macro.
- **Work file macros for tape and disk:** READ, WRITE, CHECK, NOTE, POINTR, POINTW, and POINTS.
- **Completion macros:** CLOSE and CLOSER.

CMS/DOS supports workfiles containing fixed-length unblocked records and undefined records. Disk work files are supported as single volume, single pack files. Normal extents and split extents are both supported.

Appendix C. CRR Participation Macros

The macros contained in this appendix are intended for the programmer with product development responsibility who is writing code to enable an IBM or non-IBM resource manager to participate in Coordinated Resource Recovery (CRR).

ADAPTRC



Purpose

Use the ADAPTRC macro in a resource adapter to define all the constants needed for functions, actions, response codes, and return codes in the exit interface with the synchronization point manager (SPM). The constants are defined as a series of equates. The names of the constants begin with the letters ADA. For information about writing the exit routines, see [z/VM: CMS Application Development Guide](#).

Actions and Responses

Name	Value	Description	Response	Action
ADANULL	0	Null. No response code, used to initialize.	Reserved for use by CMS	Reserved for use by CMS
ADABOUT	4	Backout.	Resource manager has backed out its changes.	Resource manager is to back out its changes.
ADARF	8	Resource manager failure.	Resource manager failed, and did not give any information about the state of its changes.	Reserved for use by CMS
ADAALLOC	12	Allocate. Conversation allocation	Reserved for use by CMS	Reserved for use by CMS
ADADSYNC	16	Deallocate synclevel. Conversation deallocation, synclevel	Reserved for use by CMS	Reserved for use by CMS
ADADA	20	Deallocate abend.	Resource manager backed out its changes and is unavailable for any more work.	Resource manager should back out all changes. Protected conversations should be ended.
ADAPREP	24	Prepare.	Reserved for use by CMS	Prepare all changes to be committed.
ADARQCMT	28	Request Commit.	Resource manager has prepared the changes.	Commit all changes (Simple commit action).
ADAPDS	32	Prepare deallocate synclevel.	Reserved for use by CMS	Reserved for use by CMS
ADARQCDS	36	Request Commit deallocate synclevel.	Reserved for use by CMS	Reserved for use by CMS
ADANNEWL	40	No new LUWID. Cannot accept a new LUWID for the next sync point.	Reserved for use by CMS	Reserved for use by CMS
ADACMTDL	44	Committed with new LUWID.		Commit all changes, use the new LUWID for the next sync point.
ADAPVPRT	48	Partner detected and reported our protocol violation	Partner detected protocol violation and severed its link.	
ADADAE	52	Deallocate allocation error	Reserved for use by CMS	Reserved for use by CMS
ADACMTD	56	Committed.	Resource manager has successfully committed its changes.	Commit all changes.
ADAFGET	60	Forget.	Resource manager has completed work for this commit.	

Name	Value	Description	Response	Action
ADAFRIP	64	Forget, resynchronization in progress.	Resynchronization is in progress for some resource, trying to commit changes.	
ADABORIP	68	Backout, resynchronization in progress.	Resynchronization is in progress, trying to back out the changes.	
ADAKBO	72	OK backout.	Resource manager has backed out its changes, in response to a backout request.	Confirm to the resource manager that its prior action of backout was successfully received and processed.
ADACCHK	76	Commit state check.	Resource manager cannot start a commit because of the application state.	
ADABCHK	80	Backout state check.	Resource manager cannot start a backout because of the application state.	
ADACPERR	84	Commit product error.	Resource manager cannot start a commit because of some condition other than the application state.	
ADAHMIX	88	Heuristic mixed.	A heuristic decision was made which was inconsistent with other resources.	
ADAPV	92	Protocol violation.	Resource adapter detected a protocol violation and severed its link.	
ADANEWL	96	New LUWID.		Store this LUWID for use during next sync point.
ADAOK	100	Successful.	Action completed successfully.	
ADABOUT2	104	Backout, second phase.		Backout the changes during the second phase of the sync point.
ADAPTRS	108	Prepare to resynchronize.		Sever the connection to resource manager causing the resource manager to backout. Prepare the resource for resynchronization processing.
ADAIRCMT	112	Initiator request commit.		Inform the initiator that all changes were successfully prepared during phase one.
ADAIRCNL	116	Initiator request commit new LUWID.		Inform the initiator that all changes were successfully prepared during phase one, and that all downstream resources can accept a new LUWID, if necessary.
ADAIFFGET	120	Initiator forget.		Inform the initiator that the sync point has successfully completed.
ADAIFFRIP	124	Initiator forget resynchronization in progress.		Inform the initiator that the sync point is successful so far, and will be completed by resynchronization.
ADAIKBO	128	Initiator OK backout.		Inform the initiator that the sync point has been backed out as requested.
ADAICMT	132	Initiator committed.		Inform the initiator that the sync point has completed successfully.

Name	Value	Description	Response	Action
ADAICRIP	136	Initiator committed resynchronization in progress.		Inform the initiator that the sync point is successful so far, and will be completed by resynchronization.
ADAIABOUT	140	Initiator backout.		Inform the initiator that the sync point has been backed out.
ADAIBRIP	144	Initiator backout resynchronization in progress.		Inform the initiator that the sync point has been backed out, and will be completed by resynchronization.
ADAIHMIX	148	Initiator heuristic mixed.		Inform the initiator that the sync point has ended with heuristic damage.
ADAPRCOM	152	Precoordination commit.		Perform precoordination processing for a commit.
ADAPRBCK	156	Precoordination backout.		Perform precoordination processing for a backout.
ADAPSABN	160	Postcoordination abnormal termination.		Perform postcoordination processing for an abnormal termination of the sync point.
ADAPSSC	164	Postcoordination state check.		Perform postcoordination processing for a state check.
ADAPSCOM	168	Postcoordination commit.		Perform postcoordination processing for a sync point which ended in a commit.
ADAPSBCK	172	Postcoordination backout.		Perform postcoordination processing for a sync point which ended in a backout.
ADAEWPUR	176	End of work unit purge.		Perform end of work unit processing because the workunitid is being purged.
ADAEWRET	180	End of work unit return.		Perform end of work unit processing because the workunitid is being returned.
ADAEWEOC	184	End of work unit end of command.		Perform end of work unit processing because of end of command.
ADAEWABN	188	End of work unit ABEND.		Perform end of work unit processing because an ABEND is ending the workunitid.
ADAEWESS	192	End of work unit end of subset.		Perform end of work unit processing because of end of subset mode.
ADABRQBO	196	Backout required, backout.		Perform the processing to put this resource in backout required state, because some other resource has to back out.
ADABRQRF	200	Backout required, resource failed.		Perform the processing to put this resource in backout required state, because some other resource has a severe error.
ADABRQDA	204	Backout required, deallocateabend.		Perform the processing to put this resource in backout required state, because a protected APPC conversation has terminated.

Name	Value	Description	Response	Action
ADAPERR	208	Program error.	Application error, changes have neither been prepared nor backed out.	
ADAAERR	212	Resource adapter error.	Adapter failed, and did not give any information about the state of its changes.	
ADAAWARN	216	Resource adapter warning.	Adapter performed the requested function, but processing environment has changed.	

Functions

Functions	Value	Description
ADAPRCF	300	Precoordination function.
ADACORF	304	Coordination function.
ADAPSCF	308	Postcoordination function.
ADAEWUF	312	End of work unit function.
ADABORQF	316	Backout required function.

Return Codes

Return Codes	Value	Description
ADACOMP	0	Completed. The adapter has completed the specified action.
ADAREDRV	4	Redrive. The adapter has started processing for the action, but has more processing to do.
ADACOMPD	8	Completed, Default response. The adapter processing has completed, and the adapter could not recognize the specified action.
ADARCAF	12	Adapter failure. The adapter could not complete processing for the specified action.

Appendix D. NETDATA Format

The format of the data transmitted and received by the NETDATA command is described in this appendix. For information on the NETDATA command, see the [z/VM: CMS Commands and Utilities Reference](#).

Figure 1 on page 551 shows the format of a file sent using the NETDATA command with the SEND option. Several control records begin the file, followed by the data records, followed by a trailer record. An acknowledgement file is composed of only control records.

The data is actually transmitted as 80-byte card images. However, the record descriptions that follow are those of *logical records* of varying length and card boundaries are ignored.

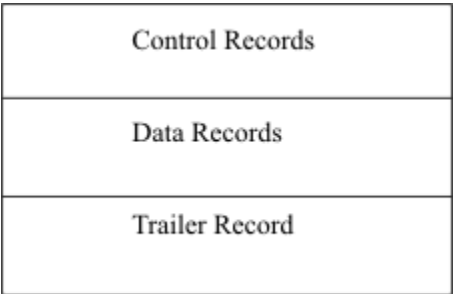


Figure 1. NETDATA File Format

Control records and **data records** have the same format, which is shown in Figure 2 on page 551. Each control record begins in the byte immediately following the end of the previous record. The first data record begins in the byte immediately following the end of the last control record in the header.

The records of the file to be transmitted are broken up into segments, whose format is shown in Figure 2 on page 551. A segment has a maximum length of 255 bytes, including the 2-byte header (length and flags bytes). If the length of a record in the file is greater than 253 bytes, the record is sent as multiple segments.

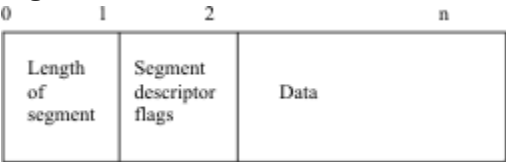


Figure 2. Data and Control Record Formats

Byte	Length	Contents
0	1	Length of segment including two-byte header (length is in the range of 2 to 255)
1	1	Segment descriptor flags: X'80' - First segment of original record. X'40' - Last segment of original record. X'20' - This is (part of) a control record. X'10' - This is record number of next record. X'0F' - Reserved.
2	n-2	Data (n is in the range of 0 to 253). Control records have a control record identifier (for example, INMR01) in bytes 2-7. Text units generally begin in byte 8. Data records begin directly in byte 2.

Exception

In the INMR02 control record, the data portion begins with a 4-byte file number field, which indicates to which file of the transmission this INMR02 refers. This field is then followed by the text units. If this field has a value greater than one, that is, if the transmission contains multiple files, the format is like that shown in [Figure 3 on page 552](#).

A **trailer control record** begins in the byte immediately following the last data byte. The format of this control record is the same as for the other control records.

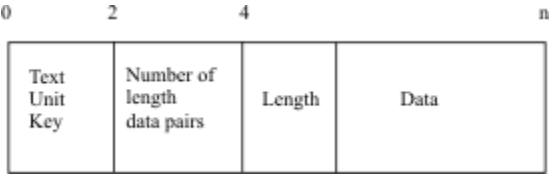


Figure 3. INMR02 Control Record Format

Control Record Formats

A control record contains a length field in byte 0, a flag field in byte 1, and the control record name in bytes 2-7. The control record data begins in byte 8. Following the eight bytes of control record header information, each control record is composed of a varying number of text units, except for the INMR02 control record, which begins with a four-byte file number field.

The NETDATA command recognizes and acts on a subset of the text unit keys that may be contained in the control records. For those text unit keys that are not recognized, NETDATA verifies that the field contains valid values, but it does not act on them. For example, a length field cannot be negative.

Text Units

Text units are the primary way of storing control information in transmission control records. The format of a single text unit is:

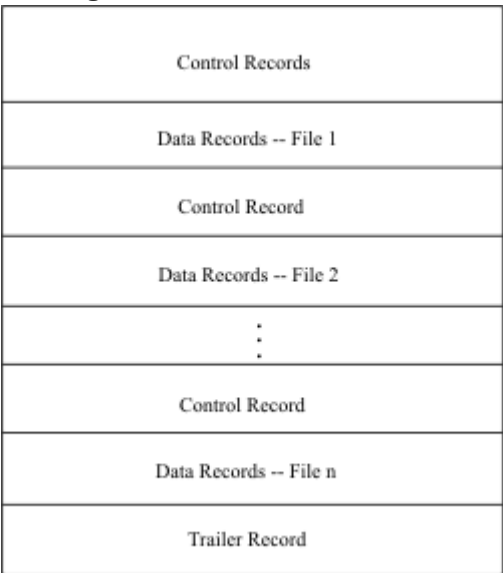


Figure 4. Text Units

Offset	Length	Description
0	2	Text unit key. The key identifies the type of information contained in the text unit. Possible key values are given in “Text Unit Keys” on page 553 .

Offset	Length	Description
2	2	Number. The number field contains the number of length-data pairs that follow. Most of the text units have only one length and one data field.
4	2	Length. The first of perhaps many length fields. The length value includes only the length of the data field immediately following it, and not its own two-byte length.
6	n	Data. The first of perhaps many data fields. The data field contains the parameter information being passed, for example, the target node name. The descriptions of the individual text units, which follow, describe the content of each.
6+n		Second length-data if the number field indicates more than one entry is present.

When text units occur in control records, they are placed together, one after another.

Dates and Times

All dates and times are expressed in Coordinated Universal Time. For all text units where a date or time is specified, the value field will have a standard format, using EBCDIC characters for the

year (4)
month (2)
day (2)
hour (2)
minute (2)
second (2)
fraction of seconds (n).

Only as much as is known of this time value need be specified. For example, if only the year were known, the value is yyyy. If microseconds are known, the value is yyyyymmddhhmmssuuuuuu.

Numeric Values

All numeric values may be specified with a length of 1 to 8 bytes. If the field is longer than 4 bytes, only the low-order 4 bytes are used.

Text Unit Keys

The following text units are recognized and acted on by the NETDATA command:

Text Unit Key (hex)	Mnemonic	Function
0002	INMDSNAM	Name of the file
0028	INMTERM	Note file (VM only)
0030	INMBLKSZ	Block Size
003C	INMDSORG	File organization
0042	INMLRECL	Logical record length
0049	INMRECFM	Record format
1001	INMTNODE	Node to which the transmission is being sent.

Text Unit Key (hex)	Mnemonic	Function
1002	INMTUID	Target user ID
1011	INMFNODE	Origin node name
1012	INMFUID	Origin user ID
1021	INMLCHG	Last change date
1024	INMFTIME	Origin time stamp
1026	INMFACK	Originator requested acknowledgement
1027	INMERRCD	RECEIVE command error code
1028	INMUTILN	Name of utility function
102C	INMSIZE	File size in bytes
102D	INMFFM	File Mode number
102F	INMNUMF	Number of files

The NETDATA command with the RECEIVE option recognizes and acts on the following text unit keys:

INMBLKSZ	INMFUID
INMDSNAM	INMLCHG
INMDSORG	INMLRECL
INMERRCD	INMNUMF
INMFACK	INMRECFM
INMFFM	INMTERM
INMFNODE	INMUTILN
INMFTIME	

The NETDATA command with the SEND option generates the following text unit keys:

INMDSNAM	INMLRECL
INMDSORG	INMNUMF
INMERRCD	INMRECFM
INMFACK	INMSIZE
INMFFM	INMTERM
INMFNODE	INMTUID
INMFTIME	INMUTILN
INMFUID	INMTNODE
INMLCHG	

For more information on the other text units, see the *System Programming Library: TSO Extensions Customization* book.

File Block Size

INMBLKSZ specifies the block size of the file. When this key is specified, NUMBER is 1, LENGTH is 1 to 8, and VALUE contains the block size of the file.

Example

For a block size of 32768, INMBLKSZ contains:

KEY	NUMBER	LENGTH	VALUE
0030	0001	0004	00008000

File Name

INMDSNAM specifies the file name of the file. File names are divided into *fields*.

In CMS, a file name is called a file identifier. It always has three fields that are separated by blanks (file name, file type, and file mode, having a maximum length of 8, 8, and 2 characters, respectively).

When transmitting a CMS file, the file mode number is not specified as part of the file identifier. Instead, it is specified on the INMFFM text unit. The file mode letter (alphabetic character) is considered to be the *highest qualifier* of the file identifier, so it is always transmitted as the first field.

In MVS, a file is called a data set. In its name there are a maximum of 22 fields, and each field has a maximum of 8 characters. The fields are separated by periods; for example,

```
AA.BB.CC.DD
```

has 4 fields. The total length, including periods, must not exceed 44 characters.

When this key is specified, NUMBER is the number of fields in the file name, and the LENGTH and VALUE fields contain the length and name of each field of the file name.

Example

For a CMS file identifier of *ABC EXEC A2*, INMDSNAM contains:

KEY	NUMBER	LENGTH1	VALUE1	LENGTH2	VALUE2	LENGTH3	VALUE3
0002	0003	0001	C1	0003	C1C2C3	0004	C5E7C5C3

Example

For a TSO data set name of *A.B*, INMDSNAM contains:

KEY	NUMBER	LENGTH1	VALUE1	LENGTH2	VALUE2
0002	0002	0001	C1	0001	C2

File Organization

INMDSORG specifies the file organization. When this key is specified, NUMBER is 1, LENGTH is 2, and VALUE contains:

X'0008'

For VSAM

X'0200'

For partitioned organization

X'4000'

For physical sequential

Note: X'0008' and X'0200' are not handled by NETDATA.

Example

For a physical sequential file, INMDSORG contains:

KEY	NUMBER	LENGTH	VALUE
003C	0001	0002	4000

Receive Results

INMERRCD indicates the result of the RECEIVE operation. When this key is specified, NUMBER is 1, LENGTH is 1 or more, and VALUE contains a string indicating the result of the RECEIVE operation.

Example

For a sent file that was received (*RECEIVED*), INMERRCD contains:

KEY	NUMBER	LENGTH	VALUE
1027	0001	0008	D9C5C3C5C9E5C5C4

Example

For a sent file that was deleted (*DELETED*), INMERRCD contains:

KEY	NUMBER	LENGTH	VALUE
1027	0001	0007	C4C5D3C5E3C5C4

Receipt Request

INMFACK indicates that the origin has requested an acknowledgement of receipt of the transmitted file. When this key is specified, NUMBER is 0 or 1. If NUMBER is 1, LENGTH is 1 to 64 and VALUE contains a transmission identifier to be returned with the notification. The return code is also returned using this text unit.

Example

For an acknowledgement request without id, INMFACK contains:

KEY	NUMBER	LENGTH	VALUE
1026	0000		

Example

For an acknowledgement request with id of FRED, INMFACK contains:

KEY	NUMBER	LENGTH	VALUE
1026	0001	0004	C6D9C5C4

File Mode

INMFFM specifies the file mode number of the file. When this key is specified, NUMBER is 1, LENGTH is 1, and VALUE contains the file mode number of the file.

Example

For a file mode number of 0, INMFFM contains:

KEY	NUMBER	LENGTH	VALUE
102D	0001	0001	F0

Node of Originator

INMFNODE specifies the node name of the origin of this transmission. When this key is specified, NUMBER is 1, LENGTH is the length of the node name, and VALUE contains the name of the origin node.

Example

For an origin node of VENICE, INMFNODE contains:

KEY	NUMBER	LENGTH	VALUE
1011	0001	0006	E5C5D5C9C3C5

Time of Transmission

INMFTIME specifies the time at which the transmission was sent (origin time stamp). When this key is specified, NUMBER is 1, LENGTH is 4 or more, and VALUE contains the time the transmission was created. The time is specified in standard format.

Example

For a transmission time of July 19, 1951 at 3:20 PM, INMFTIME contains:

KEY	NUMBER	LENGTH	VALUE
1024	0001	000C	F1F9F5F1F0F7F1F9F1F5F2F0

User ID of Originator

INMFUID specifies the user ID of the originator of this transmission. When this key is specified, NUMBER is 1, LENGTH is the length of the user ID, and VALUE is the user ID of the originator of the transmission.

Example

For a user ID of IBMUSER, INMFUID contains:

KEY	NUMBER	LENGTH	VALUE
1012	0001	0007	C9C2D4E4E2C5D9

Date of Last Change

INMLCHG specifies the last change date of the file. When this key is specified, NUMBER is 1, LENGTH is 4 or more, and VALUE contains the last change date in standard format.

Example

For a last change date (and time) of 04/01/81 8:12 PM, INMLCHG contains:

KEY	NUMBER	LENGTH	VALUE
1021	0001	000E	F1F9F8F1F0F4F0F1F2F0F1F2

Logical Record Length

INMLRECL specifies the actual or maximum length, in bytes, of a logical record in the file. When this key is specified, NUMBER is 1, LENGTH is 1 to 8, and VALUE contains the actual or maximum record length.

Example

For 80-byte records, INMLRECL contains:

KEY	NUMBER	LENGTH	VALUE
0042	0001	0001	50

Number of Files

INMNUMF specifies the number of files that make up this transmission. It is required on the INMR01 control record if there are any files in the transmission. (If this text unit is missing, the number of files is assumed to be zero, which is only true on an acknowledgement.) When this key is specified, NUMBER is 1, LENGTH is 1 to 8, and VALUE contains the number of files that make up this transmission.

Note: For the NETDATA command, VALUE cannot be more than 2 because NETDATA does not support more than one file and one note in the same transmission.

Example

For a transmission with two files, INMNUMF contains:

KEY	NUMBER	LENGTH	VALUE
102F	0001	0004	00000002

Record Format

INMRECFM specifies the record format of the file. When this key is specified, NUMBER is 1, LENGTH is 2, and VALUE is one or more of the following added together:

X'0001'

Shortened VBS format used for transmission records

X'xx02'

Varying length records without the 4-byte header

X'0200'

Data includes machine code printer control characters

X'0400'

Data contains ASA printer control characters

X'0800'

Standard fixed records or spanned variable records

X'1000'

Blocked records

X'2000'

Track overflow or variable ASCII records

X'4000'

Variable length records

X'8000'

Fixed length records

X'C000'

Undefined records.

Example

For fixed block records, INMRECFM contains:

KEY	NUMBER	LENGTH	VALUE
0049	0001	0002	9000

File Size

INMSIZE specifies the size of the file in bytes. When this key is specified, NUMBER is 1, LENGTH is 1 to 8, and VALUE contains the size of the file in bytes.

Note: The INMSIZE text unit for a partitioned data set (PDS) is the size of the PDS, not the size of a member.

Example

For a 1 megabyte file, INMSIZE contains:

KEY	NUMBER	LENGTH	VALUE
102C	0001	0004	000F4240

Note File

INMTERM specifies the file is a note. When this key is specified, NUMBER is 0, and LENGTH and VALUE are omitted.

Example

For a note, INMTERM contains:

KEY	NUMBER	LENGTH	VALUE
0028	0000		

Target Node

INMTNODE specifies the node name or node number provided by the NETDATA SEND command of the target of this transmission. When this key is specified, NUMBER is 1, LENGTH is the length of the node name or number, and VALUE is the name or number of the target node.

Example

For a node of ROME, INMTNODE contains:

KEY	NUMBER	LENGTH	VALUE
1001	0001	0004	D9D6D4C5

Target User ID

INMTUID specifies the user ID of the target of this transmission. When this key is specified, NUMBER is 1, LENGTH is the length of the user ID, and VALUE is the target user ID of the transmission.

Example

For a target user ID of IBMUSER, INMTUID contains:

KEY	NUMBER	LENGTH	VALUE
1002	0001	0007	C9C2D4E4E2C5C9

Program Name

INMUTILN specifies the name of the utility program which is used as part of restoring the transmitted data to its original format. When this key is specified, NUMBER is 1, LENGTH is the length of the utility name, and VALUE contains the name of the utility. NETDATA supports

INMCOPY

Invoke internal utility to convert from the transmission format to a sequential file.

TSO/E also supports

IEBCOPY

Invoke the IEBCOPY utility to reload a partitioned file.

AMSCIPHR

Invoke the Access Method Services REPRO command to decrypt a file.

Example

For the utility program INMCOPY, INMUTILN contains:

KEY	NUMBER	LENGTH	VALUE
1028	0001	0007	C9D5D4C3D6D7E8

Header Record (INMR01)

The INMR01 record is always the first record of a transmission. The identifier of the record is INMR01 in bytes 2-7. The remainder of the record (beginning with byte 8 of the first record) is composed of text units. The text unit keys which are always present are:

INMFNODE	Origin node name
INMFTIME	Origin time stamp
INMFUID	Origin user ID
INMLRECL	Length of physical control record segments
INMTNODE	Target node name
INMTUID	Target user ID

Text unit keys which may be present are:

INMFACK	Receipt notification requested
INMFVERS	Origin version number
INMNUMF	Number of files in this transmission
INMUSERP	User parameter string

For the INMR01 record, NETDATA with the RECEIVE option recognizes and acts on:

INMFNODE
INMFTIME
INMFUID
INMFACK
INMNUMF

For the INMR01 record, NETDATA with the SEND option generates:

INMFACK
INMFNODE
INMFTIME
INMFUID

INMLRECL
 INMNUMF
 INMTNODE
 INMTUID

File Utility Control Record (INMR02)

Each INMR02 record controls a data restoration step. In a given transmission, one or more processes represented by a corresponding number of INMR02 records are required. The only utility operation supported is INMCOPY, which converts sequential files to and from the NETDATA format. (TSO/E also supports IEBCOPY, which converts partitioned files to and from sequential files, and AMSCIPHR, which invokes the Access Method Services REPRO command to encrypt and decrypt files.)

The text units which are on the INMR02 record describe the output of the utility operation. The input with which it must work is described by the INMR03 data description record.

If the transmission contains more than one file, one or more INMR02 records are required for each file in the transmission. The INMR02 records are in the same order as the files in the transmission. The file number field (which precedes the text units in the control record) identifies which of the multiple files in the transmission the control record applies.

Note: NETDATA does not support more than one file and one note in the same transmission.

The identifier for this record is INMR02 in bytes 2-7 of the first record. Bytes 8-11 contain the number of the file in this transmission to which the control record applies. Multiple files in a single transmission are numbered sequentially starting at one.

The text unit keys begin in byte 12. Text unit keys always present are:

INMDSORG	File organization
INMLRECL	Logical record length
INMRECFM	Record format
INMSIZE	Approximate size of file in bytes
INMUTILN	Utility program name

Text units which may be present are:

INMBLKSZ	File block size
INMCREAT	Creation date
INMDIR	Number of directory blocks
INMDSNAM	File name
INMEXPDT	Expiration date
INMFFM	File mode number
INMLCHG	Last change date
INMLREF	Last reference date
INMEMBR	Member name list
INMTERM	Note file
INUSERP	User parameter string.

For the INMR02 record, NETDATA with the RECEIVE option recognizes and acts on:

INMDSORG
INMRECFM
INMUTILN
INMBLKSZ
INMDSNAM
INMFFM
INMLCHG
INMLRECL
INMTERM

For the INMR02 record, NETDATA with the SEND option generates:

INMDSNAM
INMDSORG
INMFFM
INMLCHG
INMLRECL
INMRECFM
INMSIZE
INMTERM
INMUTILN

Data Control Record (INMR03)

The INMR03 record immediately precedes the transmitted data and identifies its format.

The identifier for this record is INMR03 in bytes 2-7 of the first record; text units begin in byte 8. Text unit keys always present are:

INMDSORG	File organization
INMLRECL	Length of physical control record segments
INMRECFM	Record format
INMSIZE	Size of file in bytes

NETDATA does not recognize, validate, or act on any of the text unit keys in INMR03. Because NETDATA does not support multiple files, the information in this record is not used and the corresponding fields from the File Utility Control Record are used. Those fields are required and generated by TSO/E to allow the support of multiple files. NETDATA with the SEND option generates:

INMDSORG
INMLRECL
INMRECFM
INMSIZE

User Control Record (INMR04)

The INMR04 record can appear almost anywhere among the control records. It must not appear before the INMR01 record. When it is found, it is ignored.

The identifier for this record is INMR04 in bytes 2-7 of the first record; text units begin in byte 8. Text unit keys always present are:

INMUSERP	User parameter string
----------	-----------------------

NETDATA does not recognize nor act on any of the text units in INMR04. NETDATA with the SEND option does not create an INMR04 record.

Trailer Control Record (INMR06)

The INMR06 record is always the last record in a transmission. This record verifies that the transmission is complete.

The identifier for this record is INMR06 in bytes 2-7 of the first record; text units begin in byte 8. No text units are defined for this record.

Acknowledgement Control Record (INMR07)

The INMR07 record indicates acknowledgement of a previous transmission. When it appears, the transmission will consist of only the INMR01, INMR07, and INMR06 records.

The identifier for this record is INMR07 in bytes 2-7 of the first record; text units begin in byte 8. Text units always present are:

INMDSNAM	File name
INMFTIME	Origin time stamp
INMTERM	Note file

Either INMDSNAM or INMTERM is present, but not both. Other text units which may be present are:

INMERRCD	Error indication for the RECEIVE operation
INMFACK	Acknowledgement ID
INMFFM	File mode number
INMUSERP	User parameter string

For the INMR07 record, NETDATA with the RECEIVE option recognizes and acts on:

INMDSNAM
INMERRCD
INMFFM
INMFTIME
INMTERM

For the INMR07 record, NETDATA with the SEND option generates:

INMDSNAM
INMERRCD
INMFTIME

INMTERM

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming Interface Information

This reference document contains intended Programming Interfaces that allow the customer to write programs to obtain services of z/VM.

Trademarks

IBM, the IBM logo, and [ibm.com](https://www.ibm.com)[®] are trademarks or registered trademarks of International Business Machines Corp., in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on [IBM Copyright and trademark information](https://www.ibm.com/legal/copytrade) (<https://www.ibm.com/legal/copytrade>).

The registered trademark Linux[®] is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

- The section entitled **IBM Websites** at [IBM Privacy Statement](https://www.ibm.com/privacy) (<https://www.ibm.com/privacy>)
- [Cookies and Similar Technologies](https://www.ibm.com/privacy#Cookies_and_Similar_Technologies) (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see [z/VM: General Information](#).

Where to Get z/VM Information

The current z/VM product documentation is available in [IBM Documentation - z/VM \(https://www.ibm.com/docs/en/zvm\)](https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- [z/VM: License Information](#), GI13-4377
- [z/VM: General Information](#), GC24-6286

Installation, Migration, and Service

- [z/VM: Installation Guide](#), GC24-6292
- [z/VM: Migration Guide](#), GC24-6294
- [z/VM: Service Guide](#), GC24-6325
- [z/VM: VMSES/E Introduction and Reference](#), GC24-6336

Planning and Administration

- [z/VM: CMS File Pool Planning, Administration, and Operation](#), SC24-6261
- [z/VM: CMS Planning and Administration](#), SC24-6264
- [z/VM: Connectivity](#), SC24-6267
- [z/VM: CP Planning and Administration](#), SC24-6271
- [z/VM: Getting Started with Linux on IBM Z](#), SC24-6287
- [z/VM: Group Control System](#), SC24-6289
- [z/VM: I/O Configuration](#), SC24-6291
- [z/VM: Running Guest Operating Systems](#), SC24-6321
- [z/VM: Saved Segments Planning and Administration](#), SC24-6322
- [z/VM: Secure Configuration Guide](#), SC24-6323

Customization and Tuning

- [z/VM: CP Exit Customization](#), SC24-6269
- [z/VM: Performance](#), SC24-6301

Operation and Use

- [z/VM: CMS Commands and Utilities Reference](#), SC24-6260
- [z/VM: CMS Primer](#), SC24-6265
- [z/VM: CMS User's Guide](#), SC24-6266
- [z/VM: CP Commands and Utilities Reference](#), SC24-6268

- [z/VM: System Operation](#), SC24-6326
- [z/VM: Virtual Machine Operation](#), SC24-6334
- [z/VM: XEDIT Commands and Macros Reference](#), SC24-6337
- [z/VM: XEDIT User's Guide](#), SC24-6338

Application Programming

- [z/VM: CMS Application Development Guide](#), SC24-6256
- [z/VM: CMS Application Development Guide for Assembler](#), SC24-6257
- [z/VM: CMS Application Multitasking](#), SC24-6258
- [z/VM: CMS Callable Services Reference](#), SC24-6259
- [z/VM: CMS Macros and Functions Reference](#), SC24-6262
- [z/VM: CMS Pipelines User's Guide and Reference](#), SC24-6252
- [z/VM: CP Programming Services](#), SC24-6272
- [z/VM: CPI Communications User's Guide](#), SC24-6273
- [z/VM: ESA/XC Principles of Operation](#), SC24-6285
- [z/VM: Language Environment User's Guide](#), SC24-6293
- [z/VM: OpenExtensions Advanced Application Programming Tools](#), SC24-6295
- [z/VM: OpenExtensions Callable Services Reference](#), SC24-6296
- [z/VM: OpenExtensions Commands Reference](#), SC24-6297
- [z/VM: OpenExtensions POSIX Conformance Document](#), GC24-6298
- [z/VM: OpenExtensions User's Guide](#), SC24-6299
- [z/VM: Program Management Binder for CMS](#), SC24-6304
- [z/VM: Reusable Server Kernel Programmer's Guide and Reference](#), SC24-6313
- [z/VM: REXX/VM Reference](#), SC24-6314
- [z/VM: REXX/VM User's Guide](#), SC24-6315
- [z/VM: Systems Management Application Programming](#), SC24-6327
- [z/VM: z/Architecture Extended Configuration \(z/XC\) Principles of Operation](#), SC27-4940

Diagnosis

- [z/VM: CMS and REXX/VM Messages and Codes](#), GC24-6255
- [z/VM: CP Messages and Codes](#), GC24-6270
- [z/VM: Diagnosis Guide](#), GC24-6280
- [z/VM: Dump Viewing Facility](#), GC24-6284
- [z/VM: Other Components Messages and Codes](#), GC24-6300
- [z/VM: VM Dump Tool](#), GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- [z/VM: DFSMS/VM Customization](#), SC24-6274
- [z/VM: DFSMS/VM Diagnosis Guide](#), GC24-6275
- [z/VM: DFSMS/VM Messages and Codes](#), GC24-6276
- [z/VM: DFSMS/VM Planning Guide](#), SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (<https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support>), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/iaa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See *z/VM Performance Data Pump*.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See *Data Pump Messages*.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- [z/VM: TCP/IP Diagnosis Guide](#), GC24-6328
- [z/VM: TCP/IP LDAP Administration Guide](#), SC24-6329
- [z/VM: TCP/IP Messages and Codes](#), GC24-6330
- [z/VM: TCP/IP Planning and Customization](#), SC24-6331
- [z/VM: TCP/IP Programmer's Reference](#), SC24-6332
- [z/VM: TCP/IP User's Guide](#), SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Environmental Record Editing and Printing Program

- Environmental Record Editing and Printing Program (EREP): Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ifc2000_v2r5.pdf), GC35-0152
- Environmental Record Editing and Printing Program (EREP): User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ifc1000_v2r5.pdf), GC35-0151

Related Products

XL C++ for z/VM

- [XL C/C++ for z/VM: Runtime Library Reference](#), SC09-7624
- [XL C/C++ for z/VM: User's Guide](#), SC09-7625

z/OS

IBM Documentation - z/OS (<https://www.ibm.com/docs/en/zos>)

ANCHOR Identifier Registration Form

z/VM
ANCHOR Macro Support

Order No. SC24-6168-01

Date _____

IBM Branch Office Serving You

Please print your name, company name, and address:

Name of software product that will use your ANCHOR identifier.

Note: Do not preselect your ANCHOR identifier. This form will be returned to you with the ANCHOR identifier assigned by IBM. If an ANCHOR identifier is required for another product, please fill out a separate form and register with IBM.

Please Do Not Write Below This Line

IBM USE ONLY

Date _____

Your unique three-character identifier is _____ This ANCHOR identifier has been registered with IBM and should not be altered. This ANCHOR identifier is intended for use with the above product only.

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us

Index

Numerics

24-bit addressing [3](#), [4](#), [23](#)
31-bit addressing
 direct branches [23](#)
370 Accommodation Facility [3](#)
9346 tape cartridge [414](#)

A

ABEND exit routines, clearing or setting [18](#)
abend processing
 keeping SVC handlers across [244](#), [252](#), [253](#), [274](#)
 saving nucleus extensions across [329](#)
 saving saved segments across [380](#)
 saving subcommands across [399](#)
 saving subpools across [404](#)
abending a program [169](#)
ABNEXIT macro
 CLR option [19](#)
 ERROR= operand [20](#)
 EXIT= operand [19](#)
 RESET option [19](#)
 SET option [18](#)
 SYSTEM operand [19](#)
 UWORD= operand [19](#)
abnormal end
 causing [169](#)
ACCEPT function (CMSIUCV macro)
 format [62](#)
 parameter descriptions [62](#)
 return codes [65](#)
ACMSCVT, mapping using NUCON macro [333](#)
ADAPTRC macro [546](#)
ADDR= operand
 CMSSTOR OBTAIN [93](#)
 CMSSTOR RELEASE [101](#)
ADDRESS= operand
 AMODESW CALL [25](#)
addressing mode
 determining current setting [26](#)
 switching [24](#)
 switching inline [29](#)
ADEVTAB, mapping using NUCON macro [333](#)
AEXEC, mapping using NUCON macro [333](#)
alias
 erasing [200](#)
allocating free storage [91](#)
AMODE= operand
 AMODESW CALL [24](#)
 AMODESW RETURN [27](#)
 AMODESW SET [29](#)
 NUCEXT SET [328](#)
 SUBCOM SET [398](#)
AMODESW macro
 AMODESW CALL
 ADDRESS= operand [25](#)

AMODESW macro (*continued*)
 AMODESW CALL (*continued*)
 AMODE= operand [24](#)
 MODE=NO370 operand [25](#)
 REGS= operand [25](#)
 AMODESW QRY
 MODE=NO370 operand [26](#)
 AMODESW RETURN
 AMODE= operand [27](#)
 MODE=NO370 operand [28](#)
 REG= operand [27](#)
 AMODESW SET
 AMODE= operand [29](#)
 MODE=NO370 operand [29](#)
 SAVE= operand [29](#)
 general formats [23](#)
ANCHOR macro
 condition code setting [33](#)
 error handling [32](#)
 general format [31](#)
APPLMSG macro
 APPLID= operand [35](#)
 BUFFA= operand [36](#)
 building parameter list [42](#)
 COMP= operand [36](#)
 creating a header for messages [38](#)
 CSECT= operand [36](#)
 CSECTA= operand [36](#)
 DISP= operand [37](#)
 display format of message [37](#)
 execute format [36](#), [43](#)
 FMT= operand [39](#)
 FMTA= operand [39](#)
 generating code to fill parameter list [36](#)
 generating storage area for parameter list [35](#)
 HEADER= operand [38](#)
 invoking message facility [36](#)
 LET= operand [38](#)
 LETA= operand [38](#)
 LINE= operand [39](#)
 LINEA= operand [40](#)
 list format [35](#), [43](#)
 MAXSUBS= operand [35](#), [42](#)
 message format [39](#)
 message line number [39](#)
 message number [39](#)
 message repository [34](#)
 message severity [38](#)
 MF= operand [35](#)
 NUM= operand [39](#)
 NUMA= operand [39](#)
 processing [43](#)
 reserving program storage [42](#)
 retrieving messages from message repository [34](#)
 specifying buffer address [36](#)
 specifying call type [43](#)
 specifying macro format [35](#)

- APPLMSG macro (*continued*)
 - specifying message text [42](#)
 - standard format [35, 43](#)
 - SUB= operand [40](#)
 - substitutions [40, 42](#)
 - TEXT= operand [42](#)
 - TEXTA= operand [42](#)
 - TYPCALL= operand [43](#)
- ASA carriage control characters
 - listed [347](#)
 - specified using PRINTL macro [345](#)
- assembler language macros
 - supported by VSE [527](#)
- ATTN function
 - stacking an input line [494](#)
 - usage [494](#)

B

- batch machine [46](#)
- BATLIMIT macro [46](#)
- BATLSECT DSECT
 - mapping [46](#)
- block ID, tape [413](#)
- BNDRY= operand
 - CMSSTOR OBTAIN [95](#)
- branching to another program [23](#)
- BSIZE= operand
 - FSCB macro [192](#)
 - FSOPEN macro [204](#)
 - FSREAD macro [218](#)
 - FSWRITE macro [233](#)
- buffer synchronization (tape) [438](#)
- BUFFER= operand
 - CONSOLE OPEN macro [118](#)
 - CONSOLE QUERY macro [122](#)
 - CONSOLE READ macro [126](#)
 - CONSOLE WRITE macro [133](#)
 - FSCB macro [192](#)
 - FSOPEN macro [204](#)
 - FSREAD macro [218](#)
 - SCAN macro [373](#)
- buffered write mode (tape) [438](#)
- BYTES= operand
 - CMSSTOR OBTAIN [92](#)
 - CMSSTOR RELEASE [101](#)

C

- CACHE= operand
 - FSCB macro [192](#)
 - FSOPEN macro [206](#)
- calling programs [47](#)
- calling subroutines [24](#)
- CALLTYP= operand
 - CMSCALL macro [48](#)
- CCW(Channel Command Word)
 - building using CONSOLE EXCP macro [112](#)
 - building using CONSOLE macro [107](#)
- channel block ID, tape [414](#)
- Channel Command Word (CCW)
 - building using CONSOLE EXCP macro [112](#)
 - building using CONSOLE macro [107](#)

- checking for existence of file
 - using FSSTATE macro [224](#)
- checking tape labels using TAPESL [418](#)
- closing
 - files [196](#)
- closing files in EXEC procedures [196](#)
- CLR function (HNDIUCV macro)
 - parameter descriptions [258](#)
 - return codes [259](#)
- CLR operand
 - ABNEXIT macro [19](#)
 - HNDXT macro [243](#)
 - HNDINT macro [247](#)
 - HNDIO macro [251](#)
 - HND SVC macro [274](#)
 - IMMCMD macro [279](#)
 - REXEXIT macro [364](#)
- CLRLIST= operand
 - HNDIO macro [251](#)
- CMS functions described
 - invoking [493](#)
- CMS interface to APPC/VM
 - CMSIUCV ACCEPT function [62](#)
 - CMSIUCV CONNECT function [67](#)
 - CMSIUCV QCMSWID function [74](#)
 - CMSIUCV RESOLVE function [77](#)
 - CMSIUCV SEVER function [80](#)
 - HNDIUCV CLR (clear) function [258](#)
 - HNDIUCV HLD (hold) function [261](#)
 - HNDIUCV REP (replace) function [263](#)
 - HNDIUCV RES (resume) function [267](#)
 - HNDIUCV SET function [269](#)
- CMS level query [172](#)
- CMS macroinstructions described [15](#)
- CMS preferred functions described
 - invoking [443](#)
- CMS programming interface
 - compatibility group
 - contents of [10](#)
 - definition of [5](#)
 - DOS/VSE group
 - definition of [5](#)
 - nonsimulated OS/MVS group
 - definition of [5](#)
 - nonsimulated OS/MVS macros [14](#)
 - preferred interface group
 - advantages of [4](#)
 - components of [5](#)
 - simulated OS/MVS group
 - definition of [5](#)
 - macros contained in [11](#)
- CMS/DOS
 - DTFCD macro [537](#)
 - DTFCN macro [538](#)
 - DTFDI macro [539](#)
 - DTFMT macro [540](#)
 - DTFPR macro [541](#)
 - DTFSD macro [542](#)
 - physical IOCS macros [529](#)
 - SVC support routines [529–536](#)
 - VSE I/O macros [529](#)
 - VSE supervisor macros [529](#)
- CMSCALL macro
 - CALLTYP= operand [48](#)

CMSCALL macro (*continued*)
 comparison with SVC 202 [51](#)
 COPY= operand [49](#)
 EPLIST= operand [48](#)
 FENCE= operand [50](#)
 MODIFY= operand [50](#)
 PLIST= operand [48](#)
 PSW settings when called routine starts [52](#)
 register contents when called routine starts [52](#)
 UFLAGS= operand [49](#)

CMSCB macro
 mapping of FCBSECT DSECT [454](#)

CMSCVT macro [54](#)

CMSDEV macro
 DIAGNOSE code X'210' [57](#)
 obtaining virtual device characteristics [56](#)
 using with PRINTL macro [57](#)

CMSECVT macro [60](#)

CMSIUCV macro
 ACCEPT function [62](#)
 CONNECT function [67](#)
 QCMSWID function [74](#)
 RESOLVE function [77](#)
 SEVER function [80](#)

CMSLEVEL macro
 general format [84](#)

CMSRET macro
 FPR= operand [86](#)
 GR= operand [86](#)
 RC= operand [86](#)

CMSSTACK macro
 ORDER= operand [88](#)
 TEXT= operand [87](#)

CMSSTOR macro
 CMSSTOR OBTAIN
 ADDR= operand [93](#)
 BNDRY= operand [95](#)
 BYTES= operand [92](#)
 DWORDS= operand [92](#)
 ERROR= operand [95](#)
 generating reentrant code [96](#)
 LOC= operand [94](#)
 MSG= operand [94](#)
 SUBPOOL= operand [93](#)
 TYPCALL= operand [94](#)
 CMSSTOR RELEASE
 ADDR= operand [101](#)
 BYTES= operand [101](#)
 DWORDS= operand [101](#)
 ERROR= operand [103](#)
 MSG= operand [102](#)
 SUBPOOL= operand [101](#)
 TYPCALL= operand [102](#)

CODE= operand
 HNDEXT macro [243](#)

codes, return
 ACCEPT (CMSIUCV) function [65](#)
 CLR (clear) function [259](#)
 CONNECT (CMSIUCV) function [71](#)
 HLD (hold) function [262](#)
 QCMSWID (CMSIUCV) function [75](#)
 REP (replace) function [265](#)
 RES (resume) function [268](#)
 RESOLVE function [78](#)

codes, return (*continued*)
 SET function [272](#)
 SEVER (CMSIUCV) function [82](#)

coding conventions, macro [15](#)

commands
 search function [500](#)

communications vector table, mapping [54](#)

compatibility group
 contents of [10](#)
 definition of [5](#)

compiler switch flag [106](#)

COMPSWT macro
 OFF operand [106](#)
 ON operand [106](#)
 turning compiler switch flag on or off [106](#)

CON1ECB format [434](#)

conditional macro expansion [176](#)

CONNECT function (CMSIUCV macro)
 ERROR parameter [70](#)
 EXIT= operand [68](#)
 parameter descriptions [67](#)

CONSOLE macro
 accessing CMS full-screen console services [107](#)
 building the CCW (Channel Command Word) [107](#)
 checking device error status [107](#)
 EXCP function [110](#)
 issuing DIAGNOSE code X'58' [107](#)
 issuing SIO instructions [107](#)
 mapping information returned by [138](#)
 MODIFY function
 EXIT= operand [114](#)
 RESET= operand [114](#)
 UWORD= operand [114](#)
 OPEN function
 BUFFER= operand [118](#)
 ERROR= operand [118](#)
 RESET= operand [118](#)
 performing 3270 I/O operations [107](#)
 QUERY function
 BUFFER= operand [122](#)
 ERROR= operand [123](#)
 format [121](#)
 READ function
 BUFFER= operand [126](#)
 format [125](#)
 OPTIONS= operand [126](#)
 WAIT function [129](#)
 WRITE function
 BRKKEY= operand [134](#)
 BUFFER= operand [133](#)
 OPTIONS= operand [133](#)

console services
 accessing, using CONSOLE macro [107](#)
 CMS full-screen [107](#)
 OPEN function
 format [116](#)

control block mappings
 CMSCB macro [454](#)
 CQYSECT macro [138](#)
 DMSJNEPL macro [171](#)
 DMSSDWA macro [173](#)
 EPLIST macro [180](#)
 EXITBUFF macro [182](#)
 EXTUAREA macro [188](#)

control block mappings (*continued*)

- FSCBD macro [194](#)
- FSTD macro [230](#)
- IO macro [468](#)
- LABSECT macro [285](#)
- SCBLOCK macro [376](#)
- SGMTEXT macro [386](#)
- SHVBLOCK macro [387](#)
- VOLSECT macro [429](#)

conventions, macro coding [15](#)

Conversational Monitor System (CMS)

- CMS SUBCOM environment [501](#)
- command search function [500](#)
- storage

- STRINIT macro [485](#)

- SUBCOM function [500](#)

- VSE macros [529](#)

COPY= operand

- CMSCALL macro [49](#)

CQYSECT macro [138](#)

CRR Participation Macros [545](#)

CSFCB macro

- expansion [140](#)

CSLENTY macro

- general format [141](#)

CSLEXIT macro

- general format [144](#)

CSLFPI macro

- general format [146](#)

CSLGETP macro

- general format [150](#)

CSRCMPSC macro

- description [152](#)

- format of [152](#)

- parameter descriptions [152](#)

CSRYCMPD macro

- description [155](#)

- format [155](#)

CSRYCMPS macro

- description [160](#)

- format [160](#)

D

DCSS, managing [378](#)

DDNAME= operand

- TEOVEXIT macro [486](#)

declarative macros

- DTFCD [537](#)

- DTFCN [538](#)

- DTFDI [539](#)

- DTFMT [540](#)

- DTFPR [541](#)

- DTFSD [542](#)

default external interrupt handlers [242](#)

Definition Language for Command Syntax (DLCS) [337](#)

deleting CMS files from minidisk or SFS directory [199](#)

determining CMS level [172](#)

device information, mapping [283](#)

DEVICE= operand

- GETSID macro [239](#)

- HNDIO macro [251](#)

DEVNAME= operand

- GETSID macro [239](#)

DEVNAME= operand (*continued*)

- HNDIO macro [251](#)

DIAGNOSE code X'210'

- used by CMSDEV macro [57](#)

DIAGNOSE code X'70' issued by TODACCNT function

- using ENABLE subfunction [503](#)

- using QUERY subfunction [503](#)

DIRBUFF macro

- directory buffer mappings [163](#)

- general format [163](#)

direct branch linkage [23](#)

directory buffer mapping macro [163](#)

discarding files, aliases

- FSERASE macro [199](#)

disk file

- reading records from [217](#)

- writing records to [232](#)

Disk Operating System (DOS)

- declarative macros

- DTFCD [537](#)

- DTFCN [538](#)

- DTFDI [539](#)

- DTFMT [540](#)

- DTFPR [541](#)

- DTFSD [542](#)

- imperative macros [544](#)

- macros supported in CMS [527](#)

- support of physical IOCS macros [529](#)

- VSE macros under CMS [529](#)

DISKID function

- obtaining minidisk information [444](#)

- PLIST [444](#)

- usage [444](#), [446](#)

- using CP DASD Block I/O System Service [444](#)

DISPW macro [459](#)

DLCS [337](#)

DMSABEXP macro [168](#)

DMSABN macro

- abendng a program [169](#)

- TYPCALL= operand [169](#)

DMSBRAC COPY [510](#)

DMSEXS macro [460](#)

DMSFREE macro [461](#)

DMSFRES macro [463](#)

DMSFRET macro [464](#)

DMSFST macro

- FORM=E operand [170](#)

- setting up a file status table [170](#)

DMSJNEPL macro

- general format [171](#)

DMSKEY macro

- general format [466](#)

DMSQEFL macro

- general format [172](#)

DMSSDWA macro

- general format [173](#)

DMSSEQ

- obtaining the number of terminal input lines [446](#)

DMSSTATE macro

- ASCENV= operand [176](#)

- SET [176](#)

- TEST [176](#)

DMSTVI PLIST mapping [424](#)

- DMSWRAUD EXEC [510](#)
- DMSWRAUT EXEC [511](#)
- DMSWRESM EXEC [510](#)
- DMSWRRAC EXEC [512](#)
- DMSWSAUD module [515](#)
- DMSWSAUT module [517](#)
- DMSWSESM module [513](#), [514](#)
- DOS (Disk Operating System)
 - declarative macros
 - DTFCD [537](#)
 - DTFCN [538](#)
 - DTFDI [539](#)
 - DTFMT [540](#)
 - DTFPR [541](#)
 - DTFSD [542](#)
 - imperative macros [544](#)
 - macros supported in CMS [527](#)
 - support of physical IOCS macros [529](#)
 - VSE macros under CMS [529](#)
- DSECT for file system control block (FSCB) [194](#)
- DSECT generated for PARSECMD control block
 - CSRYCMPS macro [160](#)
 - using PARSERCB macro [340](#)
- DSECT generated for Parser Validation Code Table entry
 - using PVCENTRY macro [352](#)
- DTFCD macro [537](#)
- DTFCN macro [538](#)
- DTFDI macro [539](#)
- DTFMT macro [540](#)
- DTFPR macro [541](#)
- DTFSD macro [542](#)
- DUMMY operand
 - HNDXT macro [243](#)
- DWORDS= operand
 - CMSSTOR OBTAIN [92](#)
 - CMSSTOR RELEASE [101](#)

E

- ECB (event control block) format [433](#)
- ECB= operand
 - HNDXT macro [243](#)
- ENABLE macro [178](#)
- end-of-command
 - keeping SVC handlers across [243](#), [252](#), [274](#)
- ENDCMD= operand
 - NUCEXT SET [330](#)
- entry point, module [141](#)
- ENTRY= operand
 - NUCEXT SET [327](#)
 - REXEXIT macro [364](#)
 - SUBCOM SET [397](#)
- EPLIST macro [180](#)
- EPLIST= operand
 - CMSCALL macro [48](#)
 - PARSECMD macro [336](#)
- EQU (equate) statements
 - generating for registers using REGEQU macro [362](#)
- erasing files, aliases
 - FSERASE macro [199](#)
- error codes
 - ACCEPT (CMSIUCV) function [65](#)
 - CLR (clear) function [259](#)
 - CONNECT (CMSIUCV) function [71](#)

- error codes (*continued*)
 - HLD (hold) function [262](#)
 - QCMSWID (CMSIUCV) function [75](#)
 - REP (replace) function [265](#)
 - RES (resume) function [268](#)
 - RESOLVE function [78](#)
 - SET function [272](#)
 - SEVER (CMSIUCV) function [82](#)
- error information
 - FPERROR mapping macro [190](#)
 - WUERROR mapping macro [442](#)
- ERROR parameter
 - ACCEPT (CMSIUCV) function [64](#)
 - CLR (clear) function [258](#)
- ERROR= operand
 - ANCHOR macro [32](#)
 - CMSSTOR OBTAIN [95](#)
 - CMSSTOR RELEASE [103](#)
 - CONSOLE macro [118](#), [123](#)
 - FSPOINT macro [214](#)
 - FSREAD macro [219](#)
 - FSSTATE macro [226](#)
 - FSWRITE macro [234](#)
 - GETSID macro [240](#)
 - HNDXT macro [244](#)
 - HNDINT macro [247](#)
 - HNDVSC macro [275](#)
 - LINERD macro [293](#), [297](#)
 - NUCEXT ANCHOR [318](#)
 - NUCEXT CLR [321](#)
 - NUCEXT QUERY [323](#)
 - NUCEXT SET [331](#)
 - REXEXIT macro [366](#)
 - SCAN macro [374](#)
 - SEGMENT macro [380](#)
 - SUBCOM ANCHOR [390](#)
 - SUBCOM CLR [392](#)
 - SUBCOM QUERY [394](#)
 - SUBCOM SET [399](#)
 - SUBPOOL CREATE [404](#)
 - SUBPOOL DELETE [408](#)
 - SUBPOOL RELEASE [408](#)
 - TEOVEXIT macro [487](#)
- ESA virtual machine [3](#)
- ESA/390 architecture [3](#)
- ESA/XC architecture [3](#)
- event control block (ECB) format [433](#)
- exist buffer mapping macro [185](#)
- existence of file, determining
 - using FSSTATE macro [224](#)
- exit code, module [144](#)
- exit routines
 - clearing for external interrupts [243](#)
 - clearing for I/O interrupts [251](#)
 - clearing for SVC interrupts [274](#)
 - defining for external interrupts [242](#)
 - defining for I/O interrupts [250](#)
 - defining for SVC interrupts [273](#)
- EXIT= operand
 - ACCEPT (CMSIUCV) function [63](#)
 - CONNECT (CMSIUCV) [68](#)
 - HNDIO macro [251](#)
 - IMMCMD macro [280](#)
 - REP (HNDIUCV) function [264](#)

- EXIT= operand (*continued*)
 - SET (HNDIUCV) function [270](#)
 - TEOVEXIT macro [487](#)
- EXITBUFF macro
 - general format [182](#)
- exits for REXX programs
 - clearing user exits [364](#)
 - defining user exits [364](#)
 - invoking list of user exits [363](#)
 - maintaining list of user exits [363](#)
- EXSBUFF macro
 - general format [185](#)
- extended communications vector table, mapping [60](#)
- extended format FSCB [192](#)
- extended parameter list, mapping [180](#)
- extended parameter lists
 - building [372](#)
- external interrupt, X'2603' [189](#)
- external interrupts
 - default external interrupt handlers [242](#)
 - handling [242](#)
 - resuming suspended task using EXTCTL macro [189](#)
 - X'2603' [189](#)
- EXTUAREA macro [188](#)
- EXTCTL macro
 - resuming execution [189](#)

F

- fast path for CSL routines [146](#)
- FCBSECT DSECT mapping [454](#)
- FENCE= operand
 - CMSCALL macro [50](#)
- file pool error information mapping [190](#)
- file status table (FST)
 - creating a copy of, using FSOPEN macro [208](#)
 - creating a copy of, using FSSTATE macro [226](#)
 - setting up [170](#)
- FORM=E operand
 - DMSFST macro [170](#)
 - FSCB macro [192](#)
 - FSOPEN macro [206](#)
 - FSPOINT macro [214](#)
 - FSREAD macro [218](#)
 - FSSTATE macro [225](#)
 - FSWRITE macro [233](#)
- format
 - macroinstructions [15](#)
- FPERROR macro
 - general format [190](#)
- FPR= operand
 - CMSRET macro [86](#)
- free storage
 - allocating [91](#)
 - free storage subpools
 - creating [402](#)
 - deleting [406](#)
 - releasing [406](#)
 - releasing [100](#)
- FSCB macro
 - BSIZE= operand [192](#)
 - BUFFER= operand [192](#)
 - CACHE= operand [192](#)
 - file system control block [191](#)

- FSCB macro (*continued*)
 - FORM=E operand [192](#)
 - general format [191](#)
 - multiple FSCBs [193](#)
 - NOREC operand [192](#)
 - OPENTYP= operand [192](#)
 - RECFM= operand [191](#)
 - RECNO= operand [192](#)
- FSCB= operand
 - FSCLOSE macro [196](#)
 - FSERASE macro [199](#)
 - FSOPEN macro [204](#)
 - FSPOINT macro [213](#)
 - FSREAD macro [217](#)
 - FSSTATE macro [224](#)
 - FSWRITE macro [232](#)
- FSCBD macro
 - DSECT for file system control block (FSCB) [194](#)
 - macro expansion [194](#)
- FSCLOSE macro
 - closing open files [196](#)
 - closing SFS files [197](#)
 - ERROR= operand [196](#)
 - FSCB= operand [196](#)
 - generating reentrant code [197](#)
 - threshold, SFS file space [198](#)
- FSERASE macro
 - deleting CMS files from minidisk or SFS directory [199](#)
 - ERROR= operand [199](#)
 - FSCB= operand [199](#)
 - SFS base files, aliases [200](#)
- FSOPEN macro
 - BISIZE= operand [204](#)
 - BUFFER= operand [204](#)
 - CACHE= operand [206](#)
 - FORM=E operand [206](#)
 - FSCB= operand [204](#)
 - generating reentrant code [208](#)
 - MSG= operand [205](#)
 - NOMSG= operand [205](#)
 - NOREC= operand [204](#)
 - OPENTYP= operand [207](#)
 - readying files for input or output [202](#)
 - RECFM= operand [204](#)
 - RECNO= operand [204](#)
 - SFS files, use with [208](#)
- FSPOINT macro
 - ERROR= operand [214](#)
 - FORM=E operand [214](#)
 - FSCB= operand [213](#)
 - generating reentrant code [214](#)
 - RDPT= operand [214](#)
 - resetting write and read pointers [213](#)
 - SFS files, accessing [214](#)
 - WRPNT= operand [214](#)
- FSREAD macro
 - BSIZE= operand [218](#)
 - BUFFER= operand [218](#)
 - ERROR= operand [219](#)
 - FORM=E operand [218](#)
 - FSCB= operand [217](#)
 - generating reentrant code [219](#)
 - NOREC= operand [219](#)

FSREAD macro *(continued)*

- reading records from CMS disk file to I/O buffer [217](#)
- RECFM= operand [218](#)
- RECNO= operand [218](#)
- SFS files, accessing [220](#)
- support for variable-length records [220](#)
- threshold, SFS file space [220](#)

FSSTATE macro

- creating a copy of FST (file status table) [226](#)
- creating a copy of, using FSSTATE macro [226](#)
- determining existence of files [224](#)
- ERROR= operand [226](#)
- FORM=E operand [225](#)
- FSCB= operand [224](#)
- mapping information returned by [230](#)
- MSG= operand [225](#)
- STATEW= operand [225](#)

FST (file status table)

- creating a copy of, using FSOPEN macro [208](#)
- creating a copy of, using FSSTATE macro [226](#)
- setting up [170](#)

FSTD macro [230](#)

FSWRITE macro

- BSIZE= operand [233](#)
- ERROR= operand [234](#)
- FORM=E operand [233](#)
- FSCB= operand [232](#)
- generating reentrant code [234](#)
- NOREC= operand [234](#)
- RECFM= operand [233](#)
- RECNO= operand [233](#)
- SFS files, work unit ID [235](#)
- update-in-place facility [235](#)
- updating files of variable-length records [235](#)
- writing records from I/O buffer to CMS disk file [232](#)
- writing records sequentially [234](#)

G

get information about passed parameters [150](#)

GETSID macro

- DEVICE= operand [239](#)
- DEVNAME= operand [239](#)
- ERROR= operand [240](#)

GR= operand

- CMSRET macro [86](#)

H

handlers

- clearing for external interrupts [243](#)
- clearing for I/O interrupts [251](#)
- clearing for SVC interrupts [274](#)
- defining for external interrupts [242](#)
- defining for I/O interrupts [250](#)
- defining for SVC interrupts [273](#)

HELP, online [16](#)

HLD function (HNDIUCV macro)

- ERROR parameter [262](#)
- parameter descriptions [261](#)
- return codes [262](#)

HNDTEXT macro

- CLR option [243](#)
- CODE= operand [243](#)
- DUMMY operand [243](#)
- ECB= operand [243](#)
- ERROR= operand [244](#)
- KEEP= operand [243](#)
- SET option [243](#)
- SYSTEM= operand [244](#)
- UWORD= operand [244](#)

HNDINT macro

- ASAP option [247](#)
- CLR option [247](#)
- ERROR= operand [247](#)
- handling I/O interrupts [247](#)
- SET option [247](#)
- WAIT option [247](#)

HNDIO macro

- CLR option [251](#)
- CLRLIST= operand [251](#)
- DEVICE= operand [251](#)
- DEVNAME= operand [251](#)
- EXIT= operand [251](#)
- INTBLOK= operand [252](#)
- KEEP= operand [252](#)
- NOTIFY= operand [251](#)
- PERSIST= operand [253](#)
- SET option [251](#)
- SYSTEM= operand [252](#)
- UWORD= operand [252](#)

HNDIUCV macro

- CLR (clear) function [258](#)
- HLD (hold) function [261](#)
- REP (replace) function [263](#)
- RES (resume) function [267](#)
- SET function [269](#)

HND SVC macro

- CLR option [274](#)
- ERROR= operand [275](#)
- KEEP= operand [274](#)
- SET option [273](#)
- SYSTEM= operand [274](#)
- UWORD= operand [274](#)

HSCVSAVE macro [277](#)

I

I/O (input/output)

- macros [529](#)

I/O buffer

- reading lines to [482](#)
- reading records to [217](#)
- writing records from [232](#)

I/O devices, handling interrupts for [247](#)

I/O interrupts

- handling [250](#)

IMMBLOK macro [278](#)

IMMCMD macro

- CLR option [279](#)
- EXIT= operand [280](#)
- NAME= operand [279](#)
- QRY option [279](#)
- SET option [279](#)

- IMMCMD macro (*continued*)
 - UWORD= operand [280](#)
- IMMCMD= operand
 - NUCEXT SET [330](#)
- immediate commands
 - clearing [279](#)
 - declaring [279](#)
 - querying [279](#)
- immediate write mode (tape) [438](#)
- imperative macros [544](#)
- implicit recursion of execs, inhibiting [140](#)
- inhibiting implicit recursion of execs [140](#)
- instruction
 - executing without nucleus protection [460](#)
- INTBLOK macro [283](#)
- INTBLOK= operand
 - HNDIO macro [252](#)
- interrupt mask, manipulating [178](#)
- interrupts
 - external
 - default external interrupt handlers [242](#)
 - handling [242](#)
 - I/O
 - handling [250](#)
 - SVC
 - handling [273](#)
- INTTYPE= operand
 - NUCEXT SET [328](#)
 - SUBCOM SET [398](#)
- IO macro
 - mapping of OPSECT DSECT [468](#)

K

- KEEP= operand
 - HNDXCT macro [243](#)
 - HNDIO macro [252](#)
 - HND SVC macro [274](#)
- KEY= operand
 - NUCEXT SET [329](#)
 - SUBCOM SET [398](#)
 - SUBPOOL CREATE [404](#)

L

- labeldefid on TAPESL macro [419](#)
- LABSECT macro
 - general format [285](#)
- LANGADD function
 - adding LANGBLKs to language block chain [447](#)
 - language control block [447](#)
- LANGBLK macro [287](#)
- LANGFIND function
 - language control block [449](#)
 - locating LANGBLKs on language block chain [449](#)
- language control block
 - LANGADD function [447](#)
 - LANGFIND function [449](#)
- languages, national [44](#), [447](#)
- LINEDIT macro
 - BUFFA= operand [478](#)
 - COMP= operand [473](#)
 - converting decimal values to EBCDIC [472](#)

- LINEDIT macro (*continued*)
 - converting decimal values to hexadecimal [472](#)
 - DISP= operand [477](#)
 - displaying conversion results at terminal [472](#)
 - displaying lines contained in buffer [473](#)
 - displaying multiple blanks [473](#)
 - displaying parameter lists [476](#)
 - DOT= operand [473](#)
 - execute format [479](#)
 - indicating message substitution [480](#)
 - list format [479](#)
 - MAXSUBS= operand [478](#)
 - MF= operand [478](#)
 - multiple substitution lists [480](#)
 - passing lines to CP [477](#)
 - period placement [473](#)
 - RENT= operand [479](#)
 - specifying message text [472](#)
 - specifying substitution length [480](#)
 - specifying substitutions [474](#)
 - standard format [478](#)
 - SUB= operand [474](#)
 - TEXT= operand [472](#)
 - TEXTA= operand [473](#)
- LINERD macro
 - ATTREST= operand [293](#), [297](#)
 - CASE= operand [292](#), [296](#)
 - COL= operand [291](#), [295](#)
 - DATA= operand [290](#), [294](#)
 - ERROR= operand [293](#), [297](#)
 - LINE= operand [291](#), [295](#)
 - LOGICAL= operand [292](#), [296](#)
 - PAD= operand [292](#), [296](#)
 - PROMPT= operand [290](#), [294](#)
 - reading lines of input from terminal [289](#)
 - specifying buffer address [290](#), [294](#)
 - TRANS= operand [292](#), [296](#)
 - TYPE= operand [292](#), [296](#)
 - VNAME= operand [291](#), [294](#)
 - WAIT= operand [293](#), [297](#)
 - writing prompt information [290](#), [294](#)
- LINEWRT macro
 - ALARM= operand [305](#)
 - COL= operand [303](#)
 - COLOR= operand [304](#)
 - DATA= operand [302](#)
 - displaying lines of output at terminal [301](#)
 - EXTHI= operand [304](#)
 - HILITE= operand [304](#)
 - LINE= operand [303](#)
 - NOCR= operand [305](#)
 - PRIOR= operand [305](#)
 - PROTECT= operand [304](#)
 - VNAME= operand [302](#)
- LOC= operand
 - CMSSTOR OBTAIN [94](#)
- Locate Block function, tape [413](#)
- LRDD macro
 - general format [308](#)
- LWRD macro
 - general format [310](#)

M

- macro coding conventions [15](#)
- macro formats [15](#)
- macro libraries
 - DMSGPI MACLIB [15](#)
 - MVSXA MACLIB [11](#)
 - OSMACRO MACLIB [11](#)
 - OSMACRO1 MACLIB [14](#)
- macro return code placement [15](#)
- mapping
 - file pool error information [190](#)
- macros
 - for CMSLEVEL [84](#)
 - for CQYSECT [138](#)
 - for CSFCB [140](#)
 - for directory buffer [163](#)
 - for DMSABEXP [168](#)
 - for DMSJNEPL [171](#)
 - for DMSSDWA [173](#)
 - for EPLIST [180](#)
 - for exist buffer [185](#)
 - for EXITBUFF [182](#)
 - for file pool error [190](#)
 - for file status table [170](#)
 - for file system control block DSECT [194](#)
 - for FSTD [230](#)
 - for HSVCSAVE [277](#)
 - for IMMBLOK [278](#)
 - for INTBLOK [283](#)
 - for LABSECT [285](#)
 - for LINERD descriptor [308](#)
 - for LINEWRT descriptor [310](#)
 - for NUCON [333](#)
 - for PARSERCB [340](#)
 - for PARSERUF [342](#)
 - for PVCENTRY [352](#)
 - for REXX language processor [370](#)
 - for SCBLOCK [376](#)
 - for SGMTEXTIT [386](#)
 - for SHVBLOCK [387](#)
 - for TRANTBL [423](#)
 - for TVISECT [424](#)
 - for USERSAVE [427](#)
 - for VOLSECT [429](#)
 - for work unit error [442](#)
- work unit error information [442](#)
- message examples, notation used in [xv](#)
- message facility, invoking
 - using APPLMSG macro [36](#)
- message repository
 - retrieving a message from [35](#)
- MF= operand
 - general description [15](#)
- MF=L parameter
 - ACCEPT (CMSIUCV) [65](#)
 - CLR (clear) [259](#)
 - CONNECT (CMSIUCV) [71](#)
 - HLD (hold) [262](#)
 - REP (replace) [266](#)
 - RES (resume) [268](#)
 - RESOLVE [79](#)
 - SET [271](#)
- minidisk information, obtaining [444](#)

- MNOTE [15](#)
- MODE=NO370 operand
 - AMODESW QRY [26](#)
 - AMODESW RETURN [28](#)
 - AMODESW SET [29](#)
- MODIFY= operand
 - CMSCALL macro [50](#)
- module entry point [141](#)
- module exit code [144](#)
- MSG= operand
 - CMSSTOR OBTAIN [94](#)
 - CMSSTOR RELEASE [102](#)
 - FSOPEN macro [205](#)
 - FSSTATE macro [225](#)
 - SUBPOOL CREATE [403](#)
 - SUBPOOL DELETE [407](#)
 - SUBPOOL RELEASE [407](#)
- MVSXA MACLIB
 - simulated macros [11](#)

N

- NAME parameter
 - ACCEPT (CMSIUCV) function [62](#)
 - CLR (clear) function [258](#)
- NAME= operand
 - IMMCMD macro [279](#)
 - NUCEXT CLR [320](#)
 - NUCEXT QUERY [322](#)
 - NUCEXT SET [327](#)
 - REXEXIT macro [364](#)
 - SEGMENT macro [379](#)
 - SUBCOM CLR [392](#)
 - SUBCOM QUERY [394](#)
 - SUBCOM SET [397](#)
 - SUBPOOL CREATE [402](#)
 - SUBPOOL DELETE [406](#)
 - SUBPOOL RELEASE [406](#)
- national languages [44](#), [447](#)
- NETDATA Format [551](#)
- NOMSG= operand
 - FSOPEN macro [205](#)
- nonreentrant code
 - generating, using LINEDIT macro [479](#)
- NOREC= operand
 - FSCB macro [192](#)
 - FSOPEN macro [204](#)
 - FSREAD macro [219](#)
 - FSWRITE macro [234](#)
- notation used in message and response examples [xv](#)
- NOTIFY= operand
 - HNDIO macro [251](#)
- NUCEXT function
 - ENDCMD attribute [497](#)
 - linkage conventions [497](#)
 - nucleus extensions [495](#)
 - nucleus storage [497](#)
 - NUCXDROP command [495](#)
 - NUCXLOAD command [495](#)
 - NUCXMAP command [495](#)
 - PLISTs [498](#)
 - PURGE and RESET service calls [496](#)
 - register contents upon entry [331](#), [498](#)
 - SYSTEM and SERVICE attributes [496](#)

- NUCEXT macro
 - ANCHOR option
 - ERROR= operand [318](#)
 - CLR option
 - ERROR= operand [321](#)
 - NAME= operand [320](#)
 - general formats [317](#)
 - QUERY option
 - ERROR= operand [323](#)
 - NAME= operand [322](#)
 - SET option
 - AMODE= operand [328](#)
 - ENDCMD= operand [330](#)
 - ENTRY= operand [327](#)
 - ERROR= operand [331](#)
 - IMMCMD= operand [330](#)
 - INTTYPE= operand [328](#)
 - KEY= operand [329](#)
 - NAME= operand [327](#)
 - ORIGIN= operand [328](#)
 - PERM= operand [331](#)
 - SERVICE= operand [329](#)
 - SYSTEM= operand [329](#)
 - UFLAGS= operand [327](#)
 - UWORD= operand [327](#)
- nucleus constant area, NUCON macro mapping [333](#)
- nucleus extensions
 - clearing [320](#)
 - defining [327](#)
 - determining existence of [322](#)
 - ENDCMD option [495](#)
 - IMMCMD option [495](#), [499](#)
 - managing [317](#)
 - obtaining the SCBLOCK anchor [318](#)
 - system [496](#)
 - user [496](#)
- NUCON macro
 - generating a mapping of GPI fields [333](#)
- NUCXDROP command [495](#)
- NUCXFRES, mapping using NUCON macro [333](#)
- NUCXLOAD command [495](#)
- NUCXMAP command [495](#)
- number of terminal input lines, obtaining [446](#)

O

- obtaining free storage [91](#)
- online HELP Facility, using [16](#)
- open
 - file for subsequent read or write
 - FSOPEN macro [202](#)
 - when you can view uncommitted changes, [209](#)
- OPENTYP= operand
 - FSCB macro [192](#)
 - FSOPEN macro [207](#)
 - use when opening a file [203](#)
- OPSECT DSECT mapping [468](#)
- ORDER= operand
 - CMSSTACK macro [88](#)
- ORIGIN= operand
 - NUCEXT SET [328](#)
- OS/MVS simulation
 - OS/MVS macros CMS simulates [11](#)
 - OS/MVS macros for assembly only [14](#)

- OS/MVS simulation (*continued*)
 - programming notes [11](#)
- OSMACRO MACLIB
 - contents of [11](#)
- OSMACRO1 MACLIB
 - nonsimulated contents of [14](#)

P

- page fault completion external interrupt [189](#)
- page fault initiation external interrupt [189](#)
- parameter lists
 - extended, building [372](#)
 - mapping for REXX language processor [370](#)
 - tokenized, building [372](#)
- PARSECMD macro
 - APPLID= operand [335](#)
 - EPLIST= operand [336](#)
 - MSGBUFF= operand [336](#)
 - MSGDISP= operand [336](#)
 - parsing command arguments [334](#)
 - PLIST= operand [335](#)
 - TRANSL= operand [337](#)
 - translating command arguments [334](#)
 - TYPCALL= operand [337](#)
 - UNIQID= operand [335](#)
 - UPPER= operand [336](#)
- Parser Validation Code Table [340](#), [352](#)
- PARSERCB macro
 - expansion [340](#)
 - generating a DSECT for PARSECMD control block [340](#)
- PARSERUF macro
 - expansion [342](#)
 - generating a mapping to parser interface [342](#)
 - mapping for CMS subcommand interface [140](#)
- parsing command arguments [334](#)
- PERM= operand
 - NUCEXT SET [331](#)
- physical block ID, tape [414](#)
- PLIST=operand
 - CMSCALL macro [48](#)
 - PARSECMD macro [335](#)
- preferred interface group
 - advantages of [4](#)
 - components of [5](#)
 - routines [9](#)
- preparing files for input or output
 - FSOPEN macro [202](#)
- PRINTL macro
 - CC= operand [345](#)
 - CMSDEV= operand [346](#)
 - ERROR= operand [346](#)
 - FORM= operand [345](#)
 - printing multiple records [345](#)
 - specifying device characteristics [346](#)
 - TRC= operand [345](#)
 - writing lines to virtual printer [343](#)
- PRMLIST parameter
 - ACCEPT (CMSIUCV) function [63](#)
- processors, subcommand [389](#)
- program calls
 - AMODESW macro [23](#)
 - returning to caller [86](#)
 - supervisor assisted linkage [47](#)

program stack, placing data on [87](#)

protected conversation

 CMSIUCV CONNECT [70](#)

 CMSIUCV SEVER [81](#)

 HNDIUCV CLR [259](#)

PSW interrupt mask, manipulating [178](#)

PUNCHC macro

 ERROR= operand [350](#)

 writing a line to a virtual punch [350](#)

PVCENTRY macro

 expansion [352](#)

 generating DSECT for Parser Validation Code Table entry [352](#)

Q

QRY operand

 IMMCMD macro [279](#)

QSAM tape end-of-volume [488](#)

query CMS level [172](#)

R

RACROUTE services [509](#)

RC= operand

 CMSRET macro [86](#)

RDCARD macro

 ERROR= operand [355](#)

 RDAHEAD= operand [355](#)

 reading a line from virtual reader [354](#)

RDPNT= operand

 FSPOINT macro [214](#)

RDTAPE macro

 reading blocks from tape device [357](#)

RDTERM macro

 ATTREST= operand [483](#)

 EDIT= operand [482](#)

 LENGTH= operand [483](#)

 PRBUFF= operand [483](#)

 PRLGTH= operand [483](#)

 reading a line from terminal into I/O buffer [482](#)

 TYPE=DIRECT operand [483](#)

Read Block ID function, tape [413](#)

read pointers, resetting [213](#)

reading records sequentially [219](#)

reading files for input or output

 FSOPEN macro [202](#)

RECFM= operand

 FSCB macro [191](#)

 FSOPEN macro [204](#)

 FSREAD macro [218](#)

 FSWRITE macro [233](#)

RECNO= operand

 FSCB macro [192](#)

 FSOPEN macro [204](#)

 FSREAD macro [218](#)

 FSWRITE macro [233](#)

recursion of execs, inhibiting [140](#)

reentrant code

 generating using CMSSTOR OBTAIN [96](#)

 generating using FSCLOSE macro [197](#)

 generating using FSOPEN macro [208](#)

reentrant code (*continued*)

 generating using FSPOINT macro [214](#)

 generating using FSREAD macro [219](#)

 generating using FSWRITE macro [234](#)

 generating using LINEDIT macro [479](#), [480](#)

REG= operand

 AMODESW RETURN [27](#)

REGEQU macro

 generating a list of EQU statements [362](#)

REGS= operand

 AMODESW CALL [25](#)

releasing free storage [100](#)

releasing storage

 CMSSTOR RELEASE [101](#)

REP function (HNDIUCV macro)

 ERROR parameter [265](#)

 EXIT= operand [264](#)

 parameter descriptions [263](#)

 return codes [265](#)

repository files

 message repository [34](#)

RES function (HNDIUCV macro)

 ERROR parameter [268](#)

 parameter descriptions [267](#)

 return codes [268](#)

resetting write and read pointers [213](#)

RESOLVE function (CMSIUCV macro)

 ERROR parameter [78](#)

 parameter descriptions [77](#)

response examples, notation used in xv

resuming suspended task using EXTCTL macro [189](#)

RETINFO= operand

 TEOVEXIT macro [487](#)

return code placement [15](#)

return codes

 ACCEPT (CMSIUCV) function [65](#)

 CLR (clear) function [259](#)

 CONNECT (CMSIUCV) function [71](#)

 HLD (hold) function [262](#)

 QCMSWID (CMSIUCV) function [75](#)

 REP (replace) function [265](#)

 RES (resume) function [268](#)

 RESOLVE function [78](#)

 SET function [272](#)

 SEVER (CMSIUCV) function [82](#)

REXEXIT macro

 CLR option [364](#)

 ENTRY operand [364](#)

 ERROR operand [366](#)

 NAME operand [364](#)

 QUERY option [364](#)

 SET option [364](#)

 SYSTEM operand [366](#)

 UWORD operand [365](#)

REXX language processor

 mapping parameter list for an exit routine [370](#)

RXITDEF macro [369](#)

RXITPARM macro [370](#)

S

SAM (sequential access method)

 declarative macros [537](#), [544](#)

 I/O macros [537](#), [544](#)

- save area, HSVCSAVE [277](#)
- SAVE= operand
 - AMODESW SET [29](#)
- saved segments
 - managing [378](#)
 - sharing [380](#)
- SCAN macro
 - BUFFER= operand [373](#)
 - building parameter lists [372](#)
 - ERROR= operand [374](#)
 - mapping the SCBLOCK [376](#)
 - TEXT= operand [372](#)
 - TRANS= operand [373](#)
- SCBLOCK
 - created by SUBCOM [500](#)
 - mapping [376](#)
 - obtaining the anchor for nucleus extensions [318](#)
- SEGMENT macro
 - ERROR= operand [380](#)
 - FIND option [379](#)
 - LOAD option [379](#)
 - managing saved segments [378](#)
 - NAME= operand [379](#)
 - PURGE option [379](#)
 - SHARE= operand [380](#)
 - SKELETON= operand [380](#)
 - SYSTEM= operand [380](#)
- segments, saved
 - managing [378](#)
 - sharing [380](#)
- sequential access method (SAM)
 - declarative macros [537](#), [544](#)
 - I/O macros [537](#), [544](#)
- SERVICE= operand
 - NUCEXT SET [329](#)
- SET 370ACCOM command [3](#)
- SET CMS370AC [3](#)
- SET function (HNDIUCV macro)
 - ERROR parameter [271](#)
 - EXIT parameter [270](#)
 - parameter descriptions [269](#)
 - return codes [272](#)
- SET GEN370 [3](#)
- SET operand
 - ABNEXIT macro [18](#)
 - HNDXT macro [243](#)
 - HNDINT macro [247](#)
 - HNDIO macro [251](#)
 - HND SVC macro [273](#)
 - IMMCMD macro [279](#)
 - REXEXIT macro [364](#)
- SEVER function (CMSIUCV macro)
 - ERROR parameter [81](#)
 - parameter descriptions [80](#)
- SGMTEXIT macro
 - general format [386](#)
- SHARE= operand
 - SEGMENT macro [380](#)
- SHVBLOCK macro [387](#)
- SID, storing in register 1 [239](#)
- simulation, OS/MVS
 - OS/MVS macros CMS simulates [11](#)
 - OS/MVS macros for assembly only [14](#)
 - programming notes [11](#)
- SMAPI
 - calling RACROUTE services [509](#)
- sparse record [219](#), [234](#)
- stack, program [87](#)
- standard format for macros
 - ACCEPT (CMSIUCV) function [65](#)
 - CLR (clear) [259](#)
 - CONNECT (CMSIUCV) [71](#)
 - HLD (hold) [262](#)
 - REP (replace) [266](#)
 - RES (resume) [268](#)
 - RESOLVE [79](#)
 - SET [271](#)
- STATEW operand
 - FSSTATE macro [225](#)
- status table, file [170](#)
- storage subpools, managing
 - creating [402](#)
 - deleting [406](#)
 - releasing [406](#)
- STRINIT macro
 - format of [485](#)
- SUBCOM function
 - calling routines dynamically [500](#)
 - command search function [500](#)
 - environment, CMS SUBCOM [501](#)
 - return codes [501](#)
- SUBCOM macro
 - ANCHOR function
 - ERROR= operand [390](#)
 - CLR function
 - ERROR= operand [392](#)
 - NAME= operand [392](#)
 - general formats [389](#)
 - managing subcommand processors [389](#)
 - QUERY function
 - ERROR= operand [394](#)
 - NAME= operand [394](#)
 - SET function
 - AMODE= operand [398](#)
 - ENTRY= operand [397](#)
 - ERROR= operand [399](#)
 - INTTYPE= operand [398](#)
 - KEY= operand [398](#)
 - NAME= operand [397](#)
 - SYSTEM= operand [399](#)
 - UFLAGS= operand [397](#)
 - UWORD= operand [397](#)
- SUBCOM option of CALLTYP= operand in CMSCALL [48](#)
- subcommand, using CMSCALL [48](#)
- subcommands
 - clearing [392](#)
 - defining [397](#)
 - managing [389](#)
- SUBPOOL macro
 - CREATE
 - ERROR= operand [404](#)
 - KEY= operand [404](#)
 - MSG= operand [403](#)
 - NAME= operand [402](#)
 - SYSTEM= operand [404](#)
 - TYPCALL= operand [403](#)
 - TYPE= operand [403](#)
 - DELETE

- SUBPOOL macro (*continued*)
 - DELETE (*continued*)
 - ERROR= operand [408](#)
 - MSG= operand [407](#)
 - NAME= operand [406](#)
 - TYPCALL= operand [407](#)
 - TYPE= operand [407](#)
 - general formats [401](#)
 - managing storage subpools [401](#)
 - RELEASE
 - ERROR= operand [408](#)
 - MSG= operand [407](#)
 - NAME= operand [406](#)
 - TYPCALL= operand [407](#)
 - TYPE= operand [407](#)
- SUBPOOL= operand
 - CMSSTOR OBTAIN [93](#)
 - CMSSTOR RELEASE [101](#)
- subpools, managing free storage
 - creating [402](#)
 - deleting [406](#)
 - releasing [406](#)
- subroutine calls [24](#)
- subroutine, returning from [27](#)
- subsystem-identification word, storing in register 1 [239](#)
- SVC
 - CMS/DOS support routines [529–536](#)
- SVC 202
 - comparison with CMSCALL macro [51](#)
- SVC interrupts
 - handling [273](#)
- SYNCLVL= SYNCPT conversation
 - CMSIUCV CONNECT [70](#)
 - CMSIUCV SEVER [81](#)
 - HNDIUCV CLR [259](#)
- syntax diagrams, how to read [xiii](#)
- system character set translation tables
 - generating a DSECT for using TRANTBL [423](#)
- system MACLIBs
 - DMSGPI MACLIB [15](#)
 - MVSXA MACLIB [11](#)
 - OSMACRO MACLIB [11, 14](#)
 - OSMACRO1 MACLIB [14](#)
- system save area
 - DMSKEY macro [466](#)
- System/370 architecture [3](#)
- SYSTEM= operand
 - ABNEXIT macro [19](#)
 - HNDXT macro [244](#)
 - HNDIO macro [252, 253](#)
 - HND SVC macro [274](#)
 - NUCEXT SET [329](#)
 - REXEXIT macro [366](#)
 - SEGMENT macro [380](#)
 - SUBCOM SET [399](#)
 - SUBPOOL CREATE [404](#)

T

- table reference character (TRC)
 - specified using PRINTL macro [345](#)
- tape end-of-volume exit [488](#)
- Tape end-of-volume exits
 - restrictions [488](#)

- tape marks, generated by TAPESL [421](#)
- tape marks, processing by TAPESL [420](#)
- tape volume interface PLIST mapping [424](#)
- TAPECTL macro
 - BLKBUFF= operand [413](#)
 - ERROR= operand [413](#)
 - ERROR= parameter [358](#)
 - MODE= operand [412](#)
 - MODE= parameter [358](#)
 - positioning tape [410](#)
- tapes
 - checking and writing tape labels [418](#)
 - processing IBM standard HDR1 and EOF1 labels [418](#)
- tapes, end-of-volume exits [486](#)
- TAPESL macro
 - BLKCNT= operand [421](#)
 - ERROR= operand [421](#)
 - LABID= operand [419](#)
 - MODE= operand [420](#)
 - SPACE= operand [420](#)
 - TM= operand [421](#)
 - used with RDTAPE, WRTAPE, and TAPECTL [418](#)
- TEOVEXIT macro
 - CLR operand [486](#)
 - DDNAME= operand [486](#)
 - ERROR= operand [487](#)
 - EXIT= operand [487](#)
 - handling a CMS tape end-of-volume exit [486](#)
 - RETINFO= operand [487](#)
 - SET operand [486](#)
- terminal I/O, waiting to complete [435](#)
- terminal input lines, number [446](#)
- TEXT= operand
 - CMSSTACK macro [87](#)
 - SCAN macro [372](#)
- threshold, SFS filespace
 - FSCLOSE macro [198](#)
 - FSREAD macro [220](#)
 - FSWRITE macro [235](#)
- TODACCNT function
 - clock accounting interface [503](#)
 - ENABLE subfunction [503, 504](#)
 - issues DIAGNOSE code X'70' [503](#)
 - QUERY subfunction [503, 504](#)
 - usage [503](#)
- tokenized parameter lists
 - building [372](#)
- trailing blanks
 - removing using WRTERM macro [490](#)
- TRANS= operand
 - SCAN macro [373](#)
- translating command arguments [334](#)
- translation tables, system character set
 - generating a DSECT for using TRANTBL [423](#)
- TRANTBL macro
 - expansion [423](#)
 - generating a DSECT for system character set translation tables [423](#)
- TRC (table reference character)
 - specified using PRINTL macro [345](#)
- TVISECT macro [424](#)
- TYPCALL= operand
 - CMSSTOR OBTAIN [94](#)
 - CMSSTOR RELEASE [102](#)

TYPCALL= operand (*continued*)

- DMSABN macro [169](#)
- PARSECMD macro [337](#)
- SUBPOOL CREATE [403](#)
- SUBPOOL DELETE [407](#)
- SUBPOOL RELEASE [407](#)

TYPE= operand

- SUBPOOL CREATE [403](#)
- SUBPOOL DELETE [407](#)
- SUBPOOL RELEASE [407](#)

U

UFLAGS= operand

- CMSCALL macro [49](#)
- NUCEXT SET [327](#)
- SUBCOM SET [397](#)

user exits for REXX programs

- clearing [364](#)
- defining [364](#)
- invoking list of [363](#)
- maintaining list of [363](#)

USERLVL, mapping using NUCON macro [333](#)

USERSAVE macro [427](#)

USERSECT macro [428](#)

UWORD parameter

- ACCEPT (CMSIUCV) function [64](#)
- CONNECT (CMSIUCV) function [68](#)
- REP (Replace) function [265](#)
- SET function [271](#)

UWORD= operand

- HNDEXT macro [244](#)
- HNDIO macro [252](#)
- HNDSVC macro [274](#)
- IMMCMD macro [280](#)
- NUCEXT SET [327](#)
- REXEXIT macro [365](#)
- SUBCOM SET [397](#)

V

variable-length records

- support provided by FSREAD macro [220](#)
- support provided by FSWRITE macro [235](#)

vector table, mapping [54](#), [60](#)

virtual device characteristics

- obtaining using CMSDEV macro [56](#)

virtual machine environments

- CMS [3](#)

virtual printer

- writing lines to using PRINTL macro [343](#)

virtual printer files

- closing using CP CLOSE command [348](#)

virtual punch

- closing after PUNCHC macro [351](#)
- writing a line to [350](#)

virtual reader

- reading a line from [354](#)

VM/ESA HELP Facility, using [16](#)

VOLSECT macro

- general format [429](#)

VSE

- assembler language macros supported in CMS [527](#)

VSE (*continued*)

- I/O macros [529](#)

- macros supported under CMS [529](#)

- macros, supervisor [529](#)

- supervisor macros [529](#)

VSE macros

- declarative [537](#)

- imperative [544](#)

- SAM [537](#), [544](#)

- supervisor [529](#)

- VSE assembler language macros supported in CMS [527](#)

- VSE macros supported by CMS/DOS [529](#)

W

WAITD macro

- ERROR= operand [430](#)

- waiting for next interrupt [430](#)

WAITECB macro

- Console I/O wait [434](#)

- ECB format [433](#)

- FORMAT operand [433](#)

- OS format, of event control block [433](#)

- VSE format, of event control block [433](#)

- waiting on event control blocks (ECBs) [432](#)

WAITRD function

- logical line editing [506](#)

- reading a line of input through WAITRD [505](#)

- usage [506](#)

WAITT macro

- waiting for terminal I/O to complete [435](#)

write pointers, resetting [213](#)

writing data to a file

- using FSWRITE macro [232](#)

writing tape labels using TAPESL [418](#)

writing your own CSL routines

- CSLENTY macro [141](#)

- CSLEXIT macro [144](#)

- CSLFPI macro [146](#)

- CSLGETP macro [150](#)

WRPNT= operand

- FSPOINT macro [214](#)

WRTAPE macro

- ERROR= parameter [439](#)

- MODE= parameter [437](#)

- TRAN= parameter [438](#)

- writing blocks on tape [436](#)

WRTERM macro

- COLOR= operand [491](#)

- displaying lines at terminal [490](#)

- EDIT= operand [490](#)

WUERROR macro

- general format [442](#)

X

X'2603' external interrupt [189](#)

XA virtual machine

- running 370-only applications [3](#)

XC virtual machine

- ESA/XC [3](#)

- running 370-only applications [3](#)

XC virtual machine (*continued*)

z/XC [3](#)

Z

z/Architecture CMS [3](#)

z/CMS [3](#)

z/VM [3](#)

z/XC architecture [3](#)



Product Number: 5741-A09

Printed in USA

SC24-6262-73

