

z/VM



CMS Planning and Administration

version 6 release 1

z/VM



CMS Planning and Administration

version 6 release 1

Note:

Before using this information and the product it supports, read the information in “Notices” on page 139.

This edition applies to version 6, release 1, modification 0 of IBM z/VM (product number 5741-A07) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC24-6078-01.

© **Copyright International Business Machines Corporation 1991, 2009.**

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

Contents

Figures	vii
Tables	ix
About This Document	xi
Intended Audience	xi
How to Read Syntax Diagrams	xi
Message and Response Notation	xiii
Where To Find More Information	xiii
How to Send Your Comments to IBM	xv
If You Have a Technical Problem	xv
Chapter 1. Planning for CMS	1
What CMS Does	1
CMS Command Language	1
Storage Requirements	2
Device Support	2
Libraries	4
Libraries on the CMS System Disk	4
Program Language Facilities	7
Limited Support of OS and VSE in CMS	7
Disk and File Management	8
Tape Support	11
Unit Record Support	11
Shared Segments	12
Configurations Supported by CMS	12
Planning for SFS, BFS and CRR	13
Estimating DASD Storage Requirements for CMS	13
Estimating DASD Storage Requirements for CMS Minidisks	14
Estimating DASD Storage Requirements for File Pools	15
Estimating DASD Storage Requirements for CRR File Pools	15
CMS Restrictions	16
Chapter 2. Planning for VSE Simulation and VSAM Support	21
Planning for CMS VSAM and Access Method Services	21
Hardware Devices Supported by VSE	22
Using Data Compression with the VSE System	23
Planning for CMS/DOS	24
VSE System Generation Considerations	24
When the VSE System Must Be Online	24
CMS/DOS Tape Handling	25
CMS/DOS Disk Label Information Area	25
Chapter 3. Tailoring CMS	27
Using the SYSPROF EXEC to Tailor the CMS Environment	27
System Profile Functions	27
Modifying the SYSPROF EXEC	28
Invoking the IPL CMS Command	28
Selecting a REXX Run-time Library	29
How to Change the SYSPROF EXEC	31
How to Bypass the System Profile	33
Setting Up a Protected Application Environment	33

What the System Profile Can Do for Installations	33
Sharing File Directory Information	40
Sharing Execs and XEDIT Macros.	40
Enlarging CMS Nucleus to Contain the Minidisk Directory (Y-STAT)	41
Example of Alternate CMS Nucleus Placement	41
Updating SAMPN55 to Change the CMS System Name	45
Rebuilding the CMS Nucleus.	45
Use the SYSTEM NETID File to Configure Communication Across an NJE Network	49
Chapter 4. CMS Pipelines Administration	51
CMS Multitasking Considerations	51
Configuring Capabilities.	51
Compiling Internals	51
CMS Pipelines Shared Segment	51
CMS Pipelines Repositories	51
System Message Repository	51
IBM-Supplied User Message Repository	52
User Created Message Repository	52
CMS Pipelines Internal Message Table	52
Search Order when CMS Pipelines Messages Are Issued	52
SQL/DS Access	53
General Set Up.	53
Identify CMS Pipelines to SQL/DS.	53
Chapter 5. The Programmable Operator Facility	57
Using the Programmable Operator Facility in Various System Environments	57
Use in a Single System.	57
Use in Distributed z/VM Systems	57
Use in a Mixed Environment	58
The Logical Operator	58
Routing Table Information	59
Action Routines.	59
Exit Execs	59
How It Works	59
Flow of Operation	60
Relationship to RSCS	61
Programmable Operator Virtual Machine	62
Installing the Programmable Operator Facility	62
Invoking the Programmable Operator Facility	62
Message Output Format	66
Stopping the Programmable Operator Facility	67
Running the Programmable Operator Facility from NCCF or NetView	68
The Programmable Operator/NCCF Message Exchange	68
Installing the PMX.	68
Communication between the Programmable Operator and NCCF or NetView	69
Stopping the PMX.	70
The Logical Operator	70
The Default Logical Operator.	71
The NCCF or NetView Logical Operator	71
Assigning or Changing the Logical Operator	72
The Routing Table.	74
How the Programmable Operator Facility Uses the Routing Table	74
Routing Table Configuration Entry Statements	74
Routing Table Configuration Example.	78
Routing Table Comparison Entry Statements	79

Tailoring the Routing Table	82
The Log File	90
Logging NCCF or NetView Messages in the Log File	91
Ensuring a Complete Log	91
The Feedback File	93
Communications Checking	94
Invoking Programmable Operator Facility Commands	95
Issuing Commands in the Single System Environment	97
Issuing Commands in the Distributed z/VM Environment	97
Issuing Commands in a Mixed Environment	99
Action Routines	100
Action Routine Interface	101
Description of Supplied Action Routines	106
Exit Execs	109
Exit Exec Interface	109
Problem Determination—Debug Mode	111
Chapter 6. Programmable Operator Facility Commands	113
CMD	114
FEEDBACK	116
GET	117
LGLOPR.	118
LOADTBL	120
LOG	122
QUERY	123
SET	126
STOP	129
Chapter 7. Managing the CMS Batch Facility	131
The CMS Batch Facility	131
Using the CMS Batch Facility	131
Starting the Batch Virtual Machine	132
CMS Batch Output	133
Purging, Reordering, and Restarting Batch Jobs	133
Stopping the Batch Virtual Machine	134
Installing the CMS Batch Machine	134
Resetting the CMS Batch Facility System Limits	135
Exec Procedures for the Batch Facility Virtual Machine.	135
Data Security under the Batch Facility	135
Improved IPL Performance Using a Saved System	136
Customizing the CMS Batch Facility	136
BATEXIT1: Processing a Non-CMS Control Language	136
BATEXIT2: Special Processing of /JOB Control Cards	137
Notices	139
Programming Interface Information	140
Trademarks.	141
Glossary	143
Bibliography	145
Where to Get z/VM Information	145
z/VM Base Library	145
Overview	145
Installation, Migration, and Service	145
Planning and Administration.	145

Customization and Tuning	145
Operation and Use	145
Application Programming	145
Diagnosis	146
z/VM Facilities and Features	146
Data Facility Storage Management Subsystem for VM	146
Directory Maintenance Facility for z/VM	146
Open Systems Adapter/Support Facility	146
Performance Toolkit for VM	147
RACF Security Server for z/VM	147
Remote Spooling Communications Subsystem Networking for z/VM	147
Prerequisite Products	147
Device Support Facilities	147
Environmental Record Editing and Printing Program	147
Index	149

Figures

1.	SYSPROF \$EXEC Comments for DMSTOEAG	30
2.	Sample Coding for the CMS Nucleus Generation Profile (DMSNGP)	39
3.	LGLOPR Command Authorization for an NCCF or NetView Operator	69
4.	QUERY Command Authorization for an NCCF or NetView Operator	72
5.	Sample LGLOPR Command Entries in a Routing Table.	73
6.	The Programmable Operator Facility in a Distributed System	79
7.	Partial Routing Table	82
8.	Routing Entries to Send Messages to an NCCF or NetView Operator	86
9.	Routing Entries to Filter Responses to Routine Commands	87
10.	Uncontrolled Authorization	88
11.	Restricting Authorization by Node ID	88
12.	Restricting Authorization by User ID and Node ID	89
13.	Restricting Command Use to Specific Users	89
14.	Example of Communication in the Single System Environment	97
15.	Example of Communication in the Distributed Environment	98
16.	Example of Communication in the Mixed Environment	99
17.	Register Conventions for Invoking an Action Routine	103
18.	CP Directory entry for CMS Batch Machine	134

Tables

1.	CMS Nucleus Size and Location	2
2.	Devices Supported by a CMS Virtual Machine	2
3.	CMS System Disk Libraries	5
4.	Volume Label Contents for CMS Formatted Disks.	10
5.	Unit Record Devices Supported by CMS	11
6.	CMS Blocks per Cylinder	14
7.	Usable KBs per Cylinder	14
8.	Effective Cache Size	37

About This Document

This document tells you how to plan for and administer the Conversational Monitor System (CMS) in your IBM® z/VM® system.

Intended Audience

This document is for anyone responsible for planning and tailoring CMS.

How to Read Syntax Diagrams

This document uses diagrams (often called *railroad tracks*) 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 following examples.

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.

►—KEYWORD—►◄

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.

Abbreviations:

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.

►—KEYWOrd—►◄

In this example, you can specify KEYWO, KEYWOR, or KEYWORD.

Symbols:

You must specify these symbols exactly as they appear in the syntax diagram.

*	Asterisk
:	Colon
,	Comma
=	Equal Sign
-	Hyphen
()	Parentheses
.	Period

Syntax Diagram Convention

Example

Variables:

A variable appears in highlighted lowercase, usually italics.



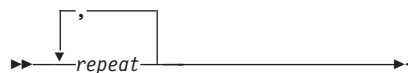
In this example, *var_name* represents a variable that you must specify following KEYWORD.

Repetitions:

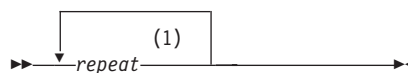
An arrow returning to the left means that the item can be repeated.



A character within the arrow means that you must separate each repetition of the item with that character.



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.



Syntax notes may also be used to explain other special aspects of the syntax.

Notes:

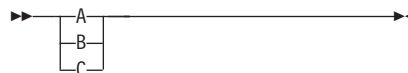
1 Specify *repeat* up to 5 times.

Required Item or Choice:

When an item is on the line, it is required. In this example, you must specify A.

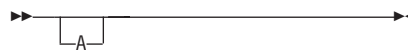


When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.



Optional Item or Choice:

When an item is below the line, it is optional. In this example, you can choose A or nothing at all.



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.



Defaults:

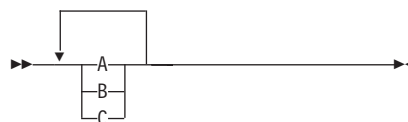
When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line.



In this example, A is the default. You can override A by choosing B or C.

Repeatable Choice:

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.



In this example, you can choose any combination of A, B, or C.

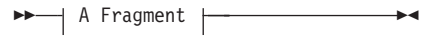
Syntax Diagram Convention

Example

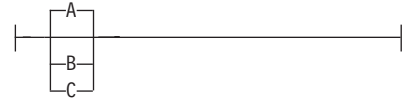
Syntax Fragment:

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.

In this example, the fragment is named “A Fragment.”



A Fragment:



Message and Response Notation

This document might include examples of messages or responses. Although most examples are shown exactly as they would appear, some content may 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 an optional item that might be displayed.
- { } Braces enclose alternative items, 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 more information about related publications, see the “Bibliography” on page 145.

Links to Other Online Documents

If you are viewing the Adobe® Portable Document Format (PDF) version of this document, it might contain links to other documents. A link to another document is based on the name of the requested PDF file. The name of the PDF file for an IBM document is unique and identifies the edition. The links provided in this document are for the editions (PDF names) that were current when the PDF file for this document was generated. However, newer editions of some documents (with different PDF names) might exist. A link from this document to another document works only when both documents reside in the same directory.

How to Send Your Comments to IBM

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

Use one of the following methods to send us your comments:

1. Send an e-mail to mhvrcfs@us.ibm.com
2. Visit the z/VM reader's comments Web page at www.ibm.com/systems/z/os/zvm/zvmforms/webqs.html
3. Mail the comments to the following address:
IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.
4. Fax the comments to us as follows:
From the United States and Canada: 1+845+432-9405
From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address
- Your e-mail address
- Your telephone or fax number
- The publication title and order number:
z/VM V6R1 CMS Planning and Administration
SC24-6171-00
- The topic and page number related to your comment
- The text of your comment

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

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

If You Have a Technical Problem

Do not use the feedback methods listed above. Instead, do one of the following:

- Contact your IBM service representative.
- Contact IBM technical support.
- Visit the z/VM support Web page at www.vm.ibm.com/service/
- Visit the IBM mainframes support Web page at www.ibm.com/systems/support/z/

Chapter 1. Planning for CMS

As a system administrator, you are responsible for managing system use for your installation. Your duties may vary depending on the size of your organization. For example, you may be involved in the day-to-day operation and control of the computer system's resources. You may also be responsible for designing or planning for additional resources as the need arises.

This chapter describes the general tasks for planning a system in the CMS environment. CMS runs in XC and XA virtual machines in z/VM. Previously, CMS (Level 11 and prior) also ran in 370 virtual machines. See the SET 370ACCOM command in the *z/VM: CP Commands and Utilities Reference* for assistance in executing 370-mode applications.

Complex tasks involving more design and control of the system may require interaction with the CP environment.

What CMS Does

The Conversational Monitor System (CMS) provides conversational facilities for virtual machine users. CMS operates only in a virtual machine, and together with CP, provides a time-sharing system suitable for program development, problem solving, and general time-sharing work.

CMS Command Language

The CMS command language lets you converse with CMS. This command language lets you use the following:

- Language compilers
- An assembler
- CMS file management system
- Context editing and line editing
- Execution control
- Debugging programs
- z/VM HELP Facility.

You can also use the CP commands available to all virtual machines under z/VM directly from CMS. By using these CP commands, you can send messages to the operator or to other users, change your virtual machine's configuration, and use spooling facilities.

To use CMS, you must first gain access to a virtual machine by the CP LOGON command and then IPL CMS. Then, you can enter commands or data from the terminal. Upon completion, each command returns control to you.

Note: If your system was built with your national language, you may be able to enter CMS commands in your own national language, rather than American English.

For information about how to use CMS, see the *z/VM: CMS User's Guide*. For a complete description of all the CMS commands, see the *z/VM: CMS Commands and Utilities Reference* book.

Storage Requirements

CMS requires virtual storage and auxiliary storage. A minimum of 2MB of virtual storage is required for a CMS virtual machine if CMS is IPLed from a named saved system. If you use a nonshared copy of CMS, your virtual machine size must be at least 20MB.

Auxiliary Storage

The CMS auxiliary storage requirements are distributed as follows:

- System residence for CMS (190 minidisk)—see Table 1.

Table 1. CMS Nucleus Size and Location

Device Type	190 Allocation	Nucleus Size	Location
3380	128 cylinders	8 cylinders	Cylinder 120
3390	107 cylinders	7 cylinders	Cylinder 100
9345	128 cylinders	8 cylinders	Cylinder 120
FB-512	153,600 blocks	9,600 blocks	Block 144,000
Notes:			
1. The CMS system and the CMS nucleus reside on the 190 minidisk. The CMS system must reside on a minidisk. It cannot reside in a file pool.			
2. The CMS system S-disk and Y-disk should be formatted as 4K block sizes and should be 4K page aligned. This means that the starting block should be divisible by 8. Formatting at 4K block size is especially important if the installation has expanded storage and uses minidisk caching.			

- Resident minidisk space or Shared File System (SFS) file pool space for application programs (CMS commands, user programs, IBM licensed programs)—the space needed is program-dependent and must be assigned by you.
- Work space (either minidisk space or file pool space) for application programs—the space needed is program-dependent and must be assigned by you.

Device Support

Table 2 shows the IBM virtual machine devices supported by CMS.

Table 2. Devices Supported by a CMS Virtual Machine

Virtual Device Type	Virtual Address ¹	Symbolic Name Default	Device Use
3210, 3215, 1052, 3066, 3270	vdev ²	CON1	System console

Table 2. Devices Supported by a CMS Virtual Machine (continued)

Virtual Device Type	Virtual Address ¹	Symbolic Name Default	Device Use
3380, 3390, 9336	190	DSK8	CMS System disk (read-only)
	191 ³	DSK1	Primary disk (user files)
	vdev	DSK2	Minidisk (user files)
	vdev	DSK3	Minidisk (user files)
	192	DSK4	Minidisk (user files)
	vdev	DSK5	Minidisk (user files)
	vdev	DSK6	Minidisk (user files)
	vdev	DSK7	Minidisk (user files)
	19E	DSK9	Minidisk (user files)
	vdev	DSK0	Minidisk (user files)
	vdev	DSKH	Minidisk (user files)
	vdev	DSKI	Minidisk (user files)
	vdev	DSKJ	Minidisk (user files)
	vdev	DSKK	Minidisk (user files)
	vdev	DSKL	Minidisk (user files)
	vdev	DSKM	Minidisk (user files)
	vdev	DSKN	Minidisk (user files)
	vdev	DSKO	Minidisk (user files)
	vdev	DSKP	Minidisk (user files)
	vdev	DSKQ	Minidisk (user files)
vdev	DSKR	Minidisk (user files)	
vdev	DSKT	Minidisk (user files)	
vdev	DSKU	Minidisk (user files)	
vdev	DSKV	Minidisk (user files)	
vdev	DSKW	Minidisk (user files)	
vdev	DSKX	Minidisk (user files)	

Notes:

¹ The device addresses shown are those that are preassembled into the CMS resident device table. These need only be modified and a new device table made resident to change the addresses.

² The virtual address of the system console may be any valid multiplexer address.

³ The 191 minidisk is automatically accessed as file mode A unless it is dynamically changed by an ACCESS at CMS initial program load (IPL). If the user ID is enrolled in an SFS file pool, the virtual machine can be set up so that the user's top directory is accessed as file mode A instead of a 191 minidisk. In this case, a 191 minidisk is optional. (The top directory is the SFS directory that is automatically created when the user ID is enrolled in an SFS file pool.)

Under CP, unit record devices and the system console may be simulated and mapped to addresses and devices other than the real ones. For instance, CMS expects a 3215, 3210, 1052, or 3270 type of operator's console, but some terminals are 2741s. Regardless of the real device type, the virtual system console is a 3215. CP handles all channel program modifications necessary for this simulation. CMS virtual disk addresses are mapped by CP to different real device numbers.

Disks

The read-only CMS system disk (S-disk), usually located at virtual address 190, contains the CMS nucleus functions and disk-resident CMS command modules. The CMS nucleus is loaded into virtual storage when you enter the CP IPL command. CMS remains resident until you enter another IPL command or until you log off. The disk-resident modules are loaded into virtual storage only when their services are needed.

CMS Planning

In addition to the system disk (file mode S) and the primary DASD storage, which can be either a minidisk or an SFS directory, each CMS user can access up to 24 additional disks or SFS directories at one time. Files that you wish to retain for later use can be stored on one of your accessed minidisks or SFS directories. Information stored on a minidisk or SFS directory remains there until you or some other authorized user erases it. Exceptions to this are the temporary minidisk (or temporary disk) and the virtual disk in storage (or vdisk). Files written on a temporary disk or vdisk are lost when you log off. See the *z/VM: CMS User's Guide* for more information about CMS disks and SFS directories.

Libraries

CMS supports simulated partitioned data sets that contain:

- CMS and OS macroinstruction/copy files to be used at compilation or assembly time (source/macroinstruction libraries). The CMS file type for these files is MACLIB.
- Object routines to be referred to at load and/or execution time (text libraries). The CMS file type for these files is TXTLIB.
- Callable services library (CSL) routines contained in VMLIB and VMMLIB.
- Executable routines that are loaded by OS SVCs that CMS simulates. The CMS file type for these files is LOADLIB.
- Executable routines that are fetched by DOS SVCs that CMS simulates. The CMS file type for these files is DOSLIB.

These simulated partitioned data sets can reside on a minidisk or in an SFS directory.

Libraries on the CMS System Disk

The CMS System Disk contains the various libraries described in Table 3 on page 5.

Table 3. CMS System Disk Libraries

Library Type	Description
<p>System Macroinstruction Libraries (File type is MACLIB)</p>	<p>Library Contents</p> <p>DMSGPI CMS macroinstructions that are part of the intended programming interface. In prior releases, these macroinstructions were in DMSSP MACLIB and CMSLIB MACLIB, which no longer exist.</p> <p>OSMACRO Macroinstructions that CMS provides for execution of programs using MVS interfaces in XA or XC virtual machines.</p> <p>MVSXA Simulated MVS/XA versions of the OS/MVS macroinstructions for the execution of programs using MVS interfaces in XA or XC virtual machines.</p> <p>OSMACRO1 Nonsimulated versions of OS/MVS macroinstructions that are used only for assembly on CMS.</p> <p>OSVSAM Subset of supported OS/VSAM macroinstructions.</p> <p>DMSOM CMS intended programming interface macros and internal assembler macroinstructions for source maintenance.</p> <p>FPLOM Pipelines intended programming interface macros and internal assembler macroinstructions for source maintenance.</p> <p>IXXOM Internal assembler macroinstructions used by the REXX component for source maintenance.</p> <p>FPLGPI Pipelines intended programming interface macroinstructions.</p> <p>HCPGPI CP intended programming interface macroinstructions.</p> <p>HCPPSI CP intended programming interface macroinstructions.</p> <p>If you plan to assemble VSE programs containing VSE macros in CMS/DOS, you must first create a CMS macroinstruction library that contains all the VSE macroinstructions you need. The <i>z/VM: CMS Commands and Utilities Reference</i> book contains procedures for copying an entire macroinstruction library. The procedure for copying individual macroinstructions is described in the section <i>Creating CMS MACLIB's</i> in the <i>z/VM: CMS Application Development Guide for Assembler</i>.</p> <p>If you plan to assemble VSE programs containing VSE/VSAM macroinstructions, you must first create a CMS MACLIB containing the VSE/VSAM macroinstructions.</p>

CMS Planning

Table 3. CMS System Disk Libraries (continued)

Library Type	Description
System Text Libraries (File type is TXTLIB)	<p>Library Contents</p> <p>CMSLIB CMS System Text Library</p> <p>TSOLIB Selected TSO routines necessary to support certain features of the language licensed programs</p> <p>CMSSAA Common Programming Interface (CPI) Communications Text Library</p> <p>DMSBASE CMS Kernel Text Library</p> <p>DMSAENV Assembler Language Environment® manager routines for multitasking</p> <p>DMSCENV C Language Environment manager routines for multitasking</p> <p>DMSAMT Multitasking initialization routines for assembler applications</p> <p>DMSCMT Multitasking initialization routines for C applications</p> <p>VMLIB Direct call stubs for routines in VMLIB callable services library</p> <p>VMMTLIB Direct call stubs for routines in VMMTLIB callable services library</p>
Callable Services Libraries	<p>Library Contents</p> <p>VMLIB CSLSEG Main CMS Callable Services Library, which can be placed into a saved segment with the SEGGEN command</p> <p>VMLIB CSLLIB Callable services library which can be loaded into user free storage</p> <p>VMMTLIB TEXT Callable services library included in the CMS nucleus, which contains the CMS Multitasking kernel and the OpenExtensions™ callable services.</p>
Other Libraries	<p>Library Contents</p> <p>CMSBAM DOSLIB Used to build the CMSBAM saved segment used with CMS/DOS.</p> <p>PROPLIB LOADLIB Supports the Programmable Operator Facility.</p>

Execution-time libraries are available with the licensed program language processors.

You can generate your own libraries and add, delete, or list entries in them by the MACLIB and TXTLIB commands. You can also specify which libraries (system and user) to use for program compilation and execution by way of the GLOBAL command. Up to 63 libraries may be specified (subject to system limits, such as command line length). Although CMS library files are similar in function to OS partitioned data sets, OS macroinstructions should not be used to update them.

Program Language Facilities

Languages supported by CMS include:

- COBOL (OS/VS COBOL and VS COBOL II)
- OS PL/I
- VS Pascal
- VS FORTRAN
- C
- Ada
- Assembler
- APL2[®]
- REXX.

Assembler F is distributed with z/VM. Language compilers that are licensed programs must be ordered separately.

CMS runs the compilers by way of the interface modules. Users should always recompile their programs and compiler interfaces under the system they are using to ensure any interface changes are incorporated (that is, control block changes). CMS commands are provided to use the compilers within the conversational environment of CMS.

Limited Support of OS and VSE in CMS

Object programs (TEXT files) produced under CMS or OS can be executed under CMS if they do not use certain OS functions not simulated by CMS. Object programs using nonsimulated OS macroinstruction functions must be transferred to an appropriate real or virtual OS machine for execution.

Sequential and partitioned data sets residing on OS disks can be read by OS programs running under CMS. Also, certain CMS commands can process data sets on OS disks.

CMS does not support multi-buffering for non-DASD devices. There is one DCB per device, not per file.

CMS simulates the control blocks, supervisor and I/O macroinstructions, linkage editor and fetch routines necessary to compile, test, and run VSE programs under CMS. The support for the VSE user is comparable to that for the OS user.

CMS supports VSAM and Access Method Services for VSE and OS users. See the *z/VM: CMS Commands and Utilities Reference* for the restrictions on using VSE/VSAM and Access Method Services in CMS. CMS also supports the VSE/VSAM macroinstructions and their options and a subset of the OS/VSAM macroinstructions.

CMS/DOS support of VSAM is based on the VSE/VSAM licensed program.

Application programmers who usually use CMS to interactively create, modify, and test programs may require facilities not supported in CMS (for example, an OS program using ISAM). They can alternately run CMS and another operating system in the same virtual machine.

A description of the actual processes for reading OS or VSE files is in the *z/VM: CMS Commands and Utilities Reference*. The *z/VM: Running Guest Operating Systems* book contains a description of alternating operating systems.

DL/I in the CMS/DOS Environment

Batch DL/I application programs can be written and tested in the CMS/DOS environment. This includes all batch application programs written in COBOL, PL/I, RPG II, and Assembler languages.

You can run any database description generation and program specification block generation. The database recovery and reorganization utilities must be run in a VSE virtual machine.

See the *z/VM: CMS Application Development Guide for Assembler* and the *DL/I DOS/VS General Information* book for more information.

Disk and File Management

For many CMS commands and application programs to use DASD storage, CMS must be made aware of that storage. To do this, the user must enter an ACCESS command. Up to 26 units of DASD storage can be accessed at the same time. The accessed items can be either SFS directories, minidisks, or full packs. Minidisks or full packs (often called virtual disks) can be in the following formats:

CMS CMS disks are formatted with the CMS FORMAT command. Files contained on these disks are in a format unique to CMS, and cannot be read or written using other operating systems.

OS or VSE

OS or DOS disks or minidisks may be used in CMS. OS or VSE programs running in CMS may read data sets or files on OS or DOS disks, but may not write or update them. OS and DOS minidisks can be formatted with the Device Support Facility or with an appropriate OS/VS or VSE disk initialization program, if the disk is a full pack.

VSAM

Minidisks used with VSAM must be formatted with the Device Support Facility. Full disks must be initialized using the appropriate OS/VS, Device Support Facility, or VSE disk initialization program.

Unlike minidisks or full packs, SFS file spaces and the directories created within them are not *formatted*. Instead, file spaces are allocated when someone administering a file pool enrolls a user and gives them space in the file pool (by using a CMS ENROLL USER command). Within this file space, one directory (known as the top directory) is automatically created for the user. The user can create a hierarchy of directories beneath this top directory by using the CMS CREATE DIRECTORY command.

Within an SFS directory, users can create and use CMS files, just as they can on CMS formatted minidisks. And, like CMS formatted minidisks, SFS directories can contain partitioned data sets of the forms supported by the CMS simulations of OS and DOS.

Although SFS directories can contain partitioned data sets, they cannot represent an OS, VSE, or VSAM disk. This means, you cannot *format* an SFS directory using the Device Support Facility. Also, you cannot format an SFS directory with an OS/VS or VSE disk initialization program.

While users typically access SFS directories, many CMS commands and program functions do not require the SFS directory to be accessed. Instead, the desired SFS directory is specified in the command or program function. CMS then directly accesses the chosen directory. There is no limit to the number of SFS directories that CMS can access at the same time in this manner.

Disk Access

Disks can be accessed so that files can only be read (read-only), or so that files can be read and written (read/write).

Both CP and CMS can control read/write access. If a disk is designated read/write by CP, then CMS determines if the access is read or write. If a disk is designated read-only by CP, then it can only be read by CMS.

To access a disk, you must:

- Identify the disk to CP as part of your virtual machine configuration. This disk is available if it is defined in your user directory entry, or it can be acquired with the CP LINK or DEFINE commands.
- Identify the disk to CMS by assigning it a file mode letter. You do this using the ACCESS command in CMS.

Or you can use the CMS VMLINK command to link and access the disk.

You may have many virtual disks known to CP in your virtual machine configuration at one time. CMS lets a maximum of 26 disks be accessed, with file mode letters A through Z. File mode S (usually at virtual address 190) is the CMS system disk. File mode A is your primary read/write file mode. It may be either a minidisk, in which case it is usually at virtual address 191, or an SFS directory.

When you access a minidisk or SFS directory, a list of files is stored in your virtual machine. Large amounts of virtual storage can be used when many minidisks and SFS directories are accessed or when they contain a very large number of files. See the CMS ACCESS command in the *z/VM: CMS Commands and Utilities Reference* for more information.

SFS Directory Access

Like disks, SFS directories can be accessed so that files can be read (read-only), or so that files can be read and written (read/write).

Unlike minidisks, however, CP has nothing to do with the mode in which SFS directories are accessed. Only CMS controls whether a particular SFS directory is accessed read or read/write. Directories that the user owns (or is authorized to access) can be accessed read or write depending on the ACCESS or VMLINK command options specified.

It is not necessary to use CP LINK and DEFINE commands to make an SFS directory a part of the virtual machine configuration. An authorized user only needs to enter an ACCESS command.

File Sharing

The CMS Shared File System lets users share files with others. Any number of readers and one writer can use a file at one time. SFS includes an automatic locking mechanism to prohibit multiple writers. The files can be shared within a processor. If connections via TSAF, ISFC links, or AVS and VTAM® are used, the files can be shared with users of other processors. For more information about using SFS to share files, see the *z/VM: CMS User's Guide*.

If, for some reason, SFS is not available to users, they can use CP to share disks and minidisks. In this case, entire minidisks or disks are shared by using CP LINK commands. LINK command operands determine the type of access (multiple users read-only or read/write). CMS does not provide any control for multiple writes to minidisk files. Therefore it is *not recommended* that CMS disks be used with multiple write access. Password protection is provided for minidisks.

Disk File Format

All disks that contain CMS files must be formatted before first use. A disk can be formatted into one of four disk block sizes:

512, 1024, 2048, and 4096 bytes

To format the disk, use the CMS FORMAT command. The CMS FORMAT command initializes disks in CMS format and writes a label on the disk. See the *z/VM: CMS Commands and Utilities Reference* book for details.

The volume label is written on record 3 of cylinder 0, track 0 for CKD devices and on block 1 (origin zero) for FB-512 devices. The volume label contents are detailed in Table 4.

Table 4. Volume Label Contents for CMS Formatted Disks

Field Description	Byte Displacement, Length	Contents by Disk Block Size (512, 1KB, 2KB, 4KB)
Label identifier	0,4	C'CMS1'
Valid	4,6	user label
Version identifier	10,2	X'0000'
Disk block size	12,4	F'512', F'1024', F'2048', or F'4096'
Disk origin pointer	16,4	F'4' or F'5'
Number of usable cylinders/blocks	20,4	F'n'
Maximum number of formatted cylinders/blocks	24,4	F'n'
Disk size in CMS blocks	28,4	F'n'
Number of CMS blocks in use	32,4	F'n'
FST size in bytes	36,4	F'64'
Number of FSTs per CMS block	40,4	F'n'
Disk FORMAT date	44,6	X'yymmddhhmmss'
Reserved for IBM use	50,2	not used, zeros
Disk offset when reserved	52,4	F'n'
Allocation Map Block with next hole	56,4	F'n'
Displacement into HBLK data of next hole	60,4	F'n'
Displacement into user part of Allocation map	64,4	F'n'
Reserved for IBM use	68,4	not used, zeros
Name of shared segment	72,8	segment name

On CKD devices, each 512, 1 KB, 2 KB, or 4 KB block (called CMS block) represents one physical record of that size on disk. For FB-512 devices, each CMS block consists of the appropriate number of contiguous FB-512 (512-byte) blocks, logically concatenated to form the correct number of data bytes for that CMS block.

Files placed on CMS disks can have logical records that are fixed or variable length. In either case, the CMS file system places these file records contiguously

into fixed length CMS blocks, spanning blocks where necessary. As a file grows or contracts, its space is expanded or reduced as needed.

Files on a CMS disk are identified by a file directory. The file directory is updated when a command is entered that changes the status of the file on the disk.

A single CMS file can contain up to approximately $(2^{31}-1)$ -132,000 disk blocks of data, grouped into as many as $2^{31}-1$ logical records, all of which must be on the same minidisk.

To ensure that the saved copy of the S-STAT or Y-STAT¹ is current, a validity check is done when a saved system is IPLed. The validity checking consists of comparing the date when the saved directory was last updated with the date when the current disk was last updated. If the dates for the S-STAT are different, then the S-STAT is built in user storage. If the dates for the Y-STAT are different, then the Y-disk is accessed using the CMS ACCESS command:

```
ACCESS 19E Y/S * * Y22.
```

This means that even when the S-disk and Y-disk are accessed in read/write mode and then released, this message will result:

```
DMSINS100W S-STAT and/or Y-STAT NOT AVAILABLE
```

Identifying Disk Files

CMS commands can list the identifications of files on CMS and non-CMS formatted disks and minidisks. The LISTFILE and FILELIST commands list the entries in the master file directory for CMS disks. The LISTDS command lists the entries in the VTOC (Volume Table of Contents) for OS and DOS disks, or data spaces on VSAM volumes.

Tape Support

CMS supports any tape device that can be attached to a virtual machine. All kinds of tape devices are equally usable with CMS, within constraints imposed by the devices themselves. CMS provides the ability to read and write any tape recording format that the device is capable of reading or writing.

CMS uses virtual tape devices. To have a virtual tape device, you have to dedicate a real tape device for the purpose (for example, by having a z/VM operator issue a CP ATTACH command).

For information about tape handling in the CMS/DOS environment, see “Planning for CMS/DOS” on page 24. For more information about tape support, see the *z/VM: CMS Application Development Guide for Assembler* and *z/VM: CMS User’s Guide*.

Unit Record Support

Table 5 lists the IBM unit record devices supported by CMS.

Table 5. Unit Record Devices Supported by CMS

Virtual Device Type	Virtual Address	Symbolic Name Default	Device Use
3505	00C	RDR1	Virtual reader

1. The S-STAT and Y-STAT are blocks of storage that contain the file status tables associated with the S-disk and Y-disk.

2. DASD address of the Y-disk is whatever was specified when CMS was generated. For the standard system, this is 19E.

Table 5. Unit Record Devices Supported by CMS (continued)

Virtual Device Type	Virtual Address	Symbolic Name Default	Device Use
3525	00D	PCH1	Virtual punch
1403, 3203, 3262, 3800, 4245, 6262	00E	PRN1	Line printer
Note: The virtual addresses shown are those that are preassembled into the CMS resident device table.			

Under z/VM, these devices are spooled. CMS does not support real or dedicated unit record devices. Table 2 on page 2 lists the devices supported as virtual devices by CMS.

Shared Segments

The default system definition files place the CMS nucleus in high storage, in the 15 MB to 20 MB address range, where it is able to be IPLed in XC and XA virtual machines. Some parts of the CMS nucleus are installed and supported only below the 16 MB line. An installation may choose, however, to change the location of the CMS nucleus.

Many factors influence the location of the shared CMS nucleus. If a user's virtual machine size extends beyond the start location of the CMS nucleus, then the CMS nucleus will exist within the user's virtual storage. This may prevent a user with a small virtual machine size from acquiring a large contiguous CMSSTOR (GETMAIN) area. The CMS nucleus should be generated at an address high enough to satisfy user storage needs.

When the location of a shared segment such as the CMS nucleus is discontinuous from a user's virtual machine, CP includes entries in a user's segment tables for all of the segments between the end of the virtual machine and the start of the shared segment. These segment tables occupy real storage. An installation with real storage constraints may choose to locate the CMS nucleus at a lower address than the default.

The procedure for changing the location of the CMS nucleus is done during the installation process and documented in the *z/VM: Guide for Automated Installation and Service*.

Configurations Supported by CMS

CMS supports:

- A maximum virtual storage size of 2047 MB for virtual machines using Enterprise Systems Architecture/370™ (ESA/370™) architecture or Enterprise Systems Architecture/390™ (ESA/390™) architecture.
- Virtual console: any terminal supported by CP as a virtual machine operator's console.
- The same unit record devices (card readers, punches, and printers) supported by z/VM as spooling devices, except the 2520 Punch.
- Up to 26 logical 3380, 3390, 9345 or FB-512 DASDs. See "CMS Restrictions" on page 16 for the maximum size of a CMS minidisk.

- Up to sixteen virtual tape devices. A tape device must have one of the following virtual device numbers: 0180-0187, 0288-028F. Most CMS functions use the device at 0181 by default.

CMS users may be authorized to use DASD space within CMS Shared File System (SFS) file pools. A file pool is a collection of minidisks managed by a virtual machine. One file pool can contain the files for many CMS users. Virtual machines that manage file pools are called file pool server machines.

Any number of file pools can be defined on your z/VM system. A CMS user can be authorized to use space in more than one file pool.

Planning for SFS, BFS and CRR

During z/VM installation, you are given a choice of loading the following IBM-supplied file pools from the z/VM system DDR image:

VMSYS

VMSYS is a repository file pool that is managed by the VMSESVS server machine and is used to hold system data.

VMSYSU

VMSYSU is an repository file pool that is managed by the VMSESVU server machine and is used as a production file pool to hold users' data.

VMSYSR

VMSYSR is a CRR file pool that is managed by the VMSESVR server machine, which is the CRR recovery server.

Before you begin the z/VM installation process (see *z/VM: Guide for Automated Installation and Service*), you should consider the benefits of loading the IBM-supplied file pools. It is quicker and easier to load the IBM-supplied file pools then tailor them, than to create your own. See the *z/VM: CMS File Pool Planning, Administration, and Operation* book for detailed information about planning for file pools.

You may want to consider installing DFSMS/VM Function Level 221 to help manage user storage within SFS. (If you plan to install DFSMS/VM, the VMSYS file pool is required.) See the *z/VM: DFSMS/VM Planning Guide* for detailed information about planning for DFSMS/VM.

Estimating DASD Storage Requirements for CMS

You can let CMS users use two forms of DASD space:

1. You can allocate minidisks directly to the user's virtual machine.
2. You can authorize them to use DASD space within a file pool. (There are various file pools and file pool servers depending on how your installation has set up the definitions.) For a complete description of these types of file pools, see *z/VM: CMS File Pool Planning, Administration, and Operation*. This space is known as the user's file space. A user can organize the files in his or her file space into a hierarchy of directories. When the file space is created, one directory is automatically created for the file space. This directory is known as the top directory.

In many commands, directories are used as rough equivalents to minidisks. For instance, to make CMS aware of either a minidisk or file pool directory, users enter an ACCESS command for the minidisk or directory.

In a z/VM system, CMS users can have several kinds of DASD storage. You might, for example, allow one user to use only an SFS file space, a second user to use only a minidisk, a third user to use BFS file spaces, and a fourth user to use some combination of these. When planning for z/VM, you should decide what kinds of DASD storage your CMS users will have. In making your decisions, consider:

- SFS and BFS do not allocate DASD space until it is actually used. When a minidisk is allocated to a user, on the other hand, any unused space on the minidisk is wasted.
- SFS and BFS let users organize their files into a hierarchy of directories. Files that reside on minidisks cannot be organized into directories.
- SFS and BFS let users share files concurrently. Note that although multiple users can share minidisks by using the CP LINK command, the sharing is limited. Multiple users can read from a minidisk, but only one user can safely write to it at the same time. More than one user writing to the same virtual device can result in a permanent loss of data. (CMS does not protect a user from loss of data when multiple users have write access to the minidisk.) Unpredictable results can occur when one user has a read-only link to a device that is being updated by a user who has the device in write status.
- SFS and BFS ease DASD management. Rather than divide a DASD volume into multiple user minidisks, you can allocate an entire DASD volume to a SFS or BFS file pool and let the file pool manage that space.
- Remote users can access data in file pools.
- Minidisks generally provide better performance than SFS or BFS file spaces. If users have applications that demand optimal file I/O performance, consider giving those users minidisk space instead of or in addition to SFS or BFS file spaces.

For more about the CMS file pools and file pool servers, see the *z/VM: CMS User's Guide* and the *z/VM: CMS File Pool Planning, Administration, and Operation* book.

Estimating DASD Storage Requirements for CMS Minidisks

The following tables list the disk capacities for all DASD supported by CMS for the four supported block sizes. Table 6 and Table 7 can help you allocate enough DASD space for CMS minidisks.

Table 6. CMS Blocks per Cylinder

Disk/Blocksize	512	1024	2048	4096
3380	690	465	270	150
3390	735	495	315	180
9345	615	420	255	150
FB-512 CMS blocks/FBA block	1	0.5	0.25	0.125

Table 7. Usable KBs per Cylinder

Disk/Blocksize	512	1024	2048	4096
3380	345	465	540	600
3390	367	495	630	720
9345	307	420	510	600
FB-512 KB/FBA block	0.5	0.5	0.5	0.5

Note: For FB-512 devices, 512 bytes multiplied by the number of FBA blocks will yield the number of usable bytes.

Table 7 displays the maximum amount of space available for CMS data on the DASD. However, this space includes:

- Space required for the file directory
- Space required for the disk allocation map
- Space required for pointer structures of files
- Space wasted due to partially used blocks.

In addition, when calculating actual useful space, you may want to consider reducing the values above by as much as 30%. The actual amount will vary greatly based on the size of your files and the number of files on the disk.

Each physical disk contains file control information as well as your data. Data requires more file control information if put into many small files instead of a few large files.

For an average CMS user, the following minidisk space should be sufficient; if the user also has an SFS file space or BFS file space, you can reduce these amounts proportionally.

Device Type	Space Required
3380	8 cylinders
3390	5 cylinders
9345	5 cylinders
FB-512	6000 512-byte blocks

Estimating DASD Storage Requirements for File Pools

Because a file pool allocates DASD space dynamically, the initial amount of DASD space you would need for an average user is somewhat less than the amounts shown in the previous section. A file pool does require, however, additional DASD space beyond the space used strictly for user data. This additional DASD space is needed for file pool control data and for file pool log minidisks.

File pool server processing uses the control data to keep track of the objects in a file pool. The control data also maps the DASD space used within a file pool. The log minidisks help protect the integrity of a file pool from system failures or device errors.

The amount of space needed for file pool control data and for log minidisks varies with factors such as the number of files and directories, the rate of change activity, and the size of the file pool.

Even with the extra DASD overhead, however, the initial total DASD space requirement per average user is less than the amounts shown in the previous section. For a rough estimate, reduce the amounts shown by about 25 %. Remember that this is a rough estimate, because your average user could consume more or less space than the numbers shown in the chart.

After z/VM is installed and users begin consuming file pool space, you can more accurately project your DASD needs and plan accordingly. File pools are expandable. You can add DASD storage to them as it is needed. For more information on file pools and determining the allocations needed for them, see the *z/VM: CMS File Pool Planning, Administration, and Operation* book.

Estimating DASD Storage Requirements for CRR File Pools

The CRR facility requires a recovery server for CRR logging and CRR resynchronization functions. The CRR recovery server runs as a part of a file pool

CMS Planning

server. If you plan to use SFS or BFS (either locally or remotely), IBM recommends that you install **one and only one** CRR recovery server on each system. The CRR recovery server can optionally be generated during z/VM installation (see *z/VM: Guide for Automated Installation and Service*), or you can generate your own CRR recovery server. See the *z/VM: CMS File Pool Planning, Administration, and Operation* book for more information on CRR.

The CRR recovery server's file pool requires minimal DASD space (compared to a SFS or BFS (repository) file pool). Additionally, CRR file pools require two CRR log minidisks. Generally, you can plan for each minidisk in the CRR file pool to be one to two cylinders on any count-key-data DASD model or 1000 to 2000 blocks on FB-512 devices. This should be more than sufficient in nearly all cases.

For more specific information on determining the minidisk allocations needed for CRR file pools, see the *z/VM: CMS File Pool Planning, Administration, and Operation* book.

CMS Restrictions

The following restrictions apply to CMS, the conversational subsystem of z/VM:

1. CMS runs only on a virtual processor provided by z/VM.

2. The maximum sizes (in cylinders or blocks) of CMS minidisks are as follows:

Device Type	Models	CMS/VSAM, CMS 512-Byte, 1K, 2K, or 4 KB Format
3380	A04, AA4, B04, AD4, BD4, AJ4, BJ4, CJ2	885 cylinders
3380	AE4, BE4	1770 cylinders
3380	AK4, BK4	2655 cylinders
3390 ⁶	A14, A18, B14, B18, B1C	1113 cylinders
3390 ⁶	A24, A28, B24, B28, B2C	2226 cylinders
3390 ⁶	A34, A38, B34, B38, B3C	3339 cylinders
3390 ⁷	A98, B9C	10017 cylinders ^{8, 9, 11}
9336	010	920115 blocks
9336	020	2147483640 blocks ^{10,} ₁₁
FB-512 (virtual disk in storage)	(not applicable)	4194296 blocks

Notes:

⁶ These capacities apply to 3390 mode and 3380 track compatibility mode.

⁷ 3380 track compatibility mode is not supported for these 3390 models.

⁸ Minidisks used with VSAM are limited to 64K (65,536) tracks (4369 3390 cylinders). This is not a limitation of CMS, but rather is a limitation of the VSE/VSAM licensed program on which the CMS support of VSAM is based.

⁹ Value can be up to the maximum number of cylinders on the DASD (CMS is limited to 32767 cylinders).

¹⁰ VM supports a SCSI disk, which is emulated as a 9336-020, up to a capacity of 2147483640 512-byte blocks (1 TB). Note, however, that directory, paging, and spooling allocations must reside within the first 16777216 blocks (blocks 0 to 16777215) of a CP-formatted disk.

¹¹ Because the CMS file system requires file status and control information to reside below 16 MB in virtual storage, there is a practical limitation on the size of CMS minidisks. As a minidisk increases in size, or more files reside on the disk, the amount of virtual storage associated with the disk for CMS file system status and control increases in storage below 16 MB. The current ECKD™ DASD limitation is 32767 cylinders for a 3390 disk on an IBM TotalStorage® DASD subsystem, or about 22 GB of data. IBM suggests that customers defining FBA SCSI disks for use by CMS should set a practical limit of about 22 GB. If larger disks are defined, they should be limited to contain very few files, or the CMS file system may not be able to obtain enough virtual storage below 16 MB to format or access those disks. For more information, see the ACCESS command in the *z/VM: CMS Commands and Utilities Reference*. The maximum size for FBA SCSI disks supported for use by CMS or GCS is 381 GB.

3. Programs that operate under CMS should use documented interfaces. Those programs that modify internal CMS control blocks to communicate with the CMS system may hamper the performance and reliability of the system.
4. CMS uses z/VM spooling to perform unit record I/O. However, a program running under CMS can issue its own SIOs to attached dedicated unit record devices.
5. Only those OS and VSE tasks that are simulated by CMS can be used to run OS and VSE programs produced by language processors under CMS.

6. Many types of object programs produced by CMS (and OS) languages can be run under CMS using CMS simulation of OS supervisory functions. Although supported in OS and VSE virtual machines under z/VM, the writing and updating of non-VSAM OS data sets and VSE files are not supported under CMS.
7. CMS can read sequential and partitioned OS data sets and sequential VSE files, by simulating certain OS and VSE system services.

The following restrictions apply when CMS reads OS data sets that reside on OS disks:

- Read-password-protected data sets are not read unless they are VSAM data sets.
- Multivolume data sets are read as single volume data sets. End-of-volume is treated as end-of-file and there is no end-of-volume switching.
- BDAM and ISAM data sets are not read.
- Keys in data sets with keys are ignored and only the data is read, except for VSAM.
- User labels in user labeled data sets are ignored.

The following restrictions apply when CMS reads VSE files that reside on DOS disks:

- Only VSE sequential files can be read. CMS options and operands that do not apply to OS sequential data sets (such as the MEMBER and CONCAT options of FILEDEF and the PDS option of MOVEFILE) also do not apply to VSE sequential files.
 - The following types of VSE files cannot be read:
 - VSE DAM and ISAM files
 - Files with the input security indicator on
 - VSE files that contain more than 16 extents. **Note:** Because user labels occupy the first extent, the file can hold only 15 additional data extents.
 - Multivolume files are read as single volume files. End-of-volume is treated as end-of-file and there is no end-of-volume switching.
 - User labels in user labeled files are ignored.
 - Because VSE files do not contain BLKSIZE, RECFM, or LRECL parameters, these parameters must be specified in the FILEDEF command or DCB macroinstruction; otherwise, BLKSIZE=32760 and RECFM=U are assigned. LRECL is not used for RECFM=U files.
 - CMS does not support the use of OS/VS DUMMY VSAM data sets at program execution time, because the CMS/DOS implementation of the DUMMY statement corresponds to VSE implementation. Specifying the DUMMY option with the DLBL command will cause an execution-time error.
8. Assembler program usage of the ISAM Interface Program (IIP) is not supported.
 9. CMS/DOS support is based on the VSE/Advanced Functions licensed program. With VSE, prior releases of VSAM are not supported under CMS/DOS.
 10. System logical units (SYSIN, SYSRDR, SYSIPT, SYSLST, and SYSPCH) are not supported for VSE formatted FB-512 devices because the SYSFIL function of VSE is not supported under CMS/DOS.
 11. Programs created using CMS/DOS are not recommended for transfer directly to a VSE machine because:

- The CMS/DOS VSE linkage editor is designed to link edit VSE programs under CMS/DOS only.
 - Programs created using the CMS/DOS assembler may have incorrect ESDs. In this case, the OS assembler is used. The OS assembler is not compatible with VSE.
 - Some VSE macroinstructions and SVCs are simulated. The code generated is not complete under CMS/DOS.
12. To ensure that the saved copy of the S-STAT or Y-STAT is current, a validity check is performed when a saved system is IPLed. (The S-STAT and Y-STAT are blocks of storage that contain the file status tables associated with the S-disk and Y-disk.) This check is performed only for S-disks and Y-disks formatted in 512-, 1024-, 2048-, or 4096-byte CMS blocks. The validity checking consists of comparing the date when the saved directory was last updated with the date when the current disk was last updated. If the dates for the S-STAT are different, then the S-STAT is built in user storage. If the dates for the Y-STAT are different, then the Y-disk is accessed using the CMS ACCESS command:

```
ACCESS 19E Y/S * * Y2
```

This means that even when the S-disk and Y-disk are accessed in read/write mode and then released, this message will result:

```
DMSINS100W S-STAT and/or Y-STAT NOT AVAILABLE
```

13. Programs that modify the file ID of an FST can destroy the integrity of the file system, and they are not supported by CMS. These programs may cause a “file not found” condition for the file until the disk is accessed again.
14. When loading text that contains dummy sections or defines pseudo registers, their cumulative length must not exceed 32767 (decimal) or X'7FFF'. If this limit is exceeded, the values stored by the CXD entry will not be accurate and the load map will not contain the correct cumulative length values.
15. Applications dependent on 370-mode instructions will no longer execute without the 370 Accommodation Facility being enabled first in CMS Level 12. Subsequent CMS levels will no longer support execution in a 370 virtual machine.

Chapter 2. Planning for VSE Simulation and VSAM Support

CMS supports interactive program development for Operating System (OS) and Virtual Storage Extended (VSE) programs using Virtual Storage Access Method (VSAM). It also supports VSAM macroinstructions used in CMS programs. All of the VSE/VSAM macroinstructions and their options and a subset of the OS/VSAM macroinstructions are supported by CMS.

Access method services to manipulate OS, VSE VSAM, and SAM data sets, and VSAM for use with DOS/VS SORT/MERGE are also supported by CMS.

Planning for CMS VSAM and Access Method Services

Under CMS, VSAM data sets can span up to 25 DASD volumes. CMS does not support VSAM data set sharing. However, CMS does support the sharing of minidisks or full pack minidisks. Only one user may have write access to the VSAM master catalog, but many other users may read and reference the catalog.

VSAM data sets created in CMS are not in the CMS file format. Therefore, CMS commands currently used to manipulate CMS files cannot be used for VSAM data sets that are read or written in CMS. VSAM data sets cannot be stored in a CMS Shared File System (SFS) file pool.

Because VSAM data sets in CMS are not a part of the CMS file system, CMS file size, record length, and minidisk size restrictions do not apply. The VSAM data sets are manipulated with access method services programs running under CMS, instead of with the CMS file system commands. Also, all VSAM minidisks and full packs used in CMS must be initialized with the Device Support Facility or an appropriate VSE or OS/VS disk initialization program (if the minidisk is a full pack); the CMS FORMAT command must not be used.

Both CMS and GCS support the VSE/VSAM for VM Version 6 Release 1 Data Compression Services, which allows automatic compression and expansion of data records on clusters DEFINED as COMPRESS. VSAM Data Compression Services require a 'VSAM.COMPRESS.CONTROL' KSDS cluster to be defined in each catalog where compressed data will reside as a central control for Data Compression Services. ESDS, KSDS, and VRDS clusters can then be defined as COMPRESS and related to the compression control data set in the catalog where the cluster is defined. RRDS and SAM-ESDS cluster types cannot be defined as COMPRESS. Note that clusters defined as COMPRESS type cannot be opened in control interval mode. Compressed data is under the control of the VSE/VSAM program.

Existing customer data clusters can be redefined as COMPRESS and reloaded with data records through REPRO copy services. No application program changes are necessary to use these compressed data files. The VSE/VSAM program automatically compresses records on VSAM write requests and re-expands them again for VSAM read requests by the application.

Data Compression Services will take advantage of the CMPSC hardware compression instruction, if available, to improve performance. Otherwise, a software simulation of the instruction will be used to execute the actual data compression.

Some return codes and feedback reason codes for Data Compression Services differ between MVS/VSAM and VSE/VSAM environments. For more detailed

VSE and VSAM Planning

information on these differences, refer to the 'OS/VSAM Error Codes' section of the *z/VM: CMS Application Development Guide for Assembler* for OPEN, CLOSE, and I/O Request error code tables.

GCS users can find error code information in the 'VSAM Data Management Service Macros' section of the *z/VM: Group Control System*.

For more information on VSE/VSAM Data Compression Services, see *VSE/VSAM Version 6 Release 1 Commands*, *VSE/VSAM Version 6 Release 1 User's Guide and Application Programming*, and *VSE/ESA Version 2 Release 1 Messages and Codes*.

Hardware Devices Supported by VSE

CMS support of VSAM data sets is based on VSE/VSAM. Disks supported by VSE/VSAM can be used for VSAM data sets in CMS. These disks are:

Device Number	Model Number
3380	Direct Access Storage
3390	Direct Access Storage
9336	Direct Access Storage Models 010 and 020

When the z/VM processor is attached to a Mass Storage System (MSS), the CMS disk may be defined as a 3330 Model 1 that is mapped by z/VM to all or part of a 3330V volume.

CMS disk files used as input to or output from Access Method Services can reside in an SFS file pool or on any disk supported by CMS.

Data Set Compatibility Considerations

CMS can read and update VSAM data sets created under VSE/VSAM or OS/VS. VSAM data sets with physical record sizes .5K, 1K, 2K, or 4K created under CMS can be read and updated by OS/VS VSAM. See the *VSE/VSAM General Information Manual* for more information about VSE/VSAM and OS/VS VSAM.

If you perform allocation on a minidisk in CMS, you cannot use that minidisk in an OS virtual machine in any manner that causes further allocation. VSE/VSAM (and thus CMS) ignores the format-5, free space DSCB on VSAM disks when it allocates extents. If allocation later occurs in an OS machine, OS attempts to create an accurate format-5 DSCB. However, the format-5 DSCB created by OS does not correctly reflect the free space on the minidisk because OS expects it to be a full pack. In CMS, allocation occurs whenever data spaces or data sets are defined, and space is released whenever data spaces, catalogs, and data sets are erased.

ISAM Interface Program (IIP): CMS does not support the VSAM ISAM Interface Program (IIP). Thus, any program that creates and accesses ISAM (indexed sequential access method) data sets cannot access VSAM key sequential data sets.

There is one exception to this restriction. If you have OS PL/I programs that have files declared as ENV(INDEXED) and if the library routines detect that the data set being accessed is a VSAM data set, your programs will execute VSAM I/O requests.

Planning Considerations for Installing VSAM under CMS

CMS support of VSAM and Access Method Services is based on the VSE/VSAM licensed program. You must order the supported level of the VSE/VSAM product and follow its installation instructions to install VSAM under CMS.

VSAM installation under CMS requires that the DOSINST saved segment be generated. Information on generating these segments is in the *z/VM: Guide for Automated Installation and Service*.

Using Data Compression with the VSE System

The VSE/VSAM Version 6.1.0 (program number 5686-066) supports Data Compression to save DASD space in large customer databases. CMS and GCS also support the VSE/VSAM for VM Version 6 Release 1 interface for Data Compression. Compressed data sets created on the VSE system can also be used by applications used by CMS and GCS.

Creating a Compressed CLUSTER

Two things must be done to create a new data set in compressed format:

1. A 'VSAM.COMPRESS.CONTROL' KSDS compression control data set must be defined in each catalog where compressed data will reside.
2. The new data set CLUSTER must be defined as COMPRESS format.

When you use AMSERV to create a VSAM cluster, the COMPRESS parameter of the DEFINE function will allow record data to be compressed when it is written and will expand data when it is read. This parameter automatically lets VSAM know if the data is to be converted by VSAM when it is read or written; no application program changes are necessary.

For information on how to define compression control data sets and use the COMPRESS parameter on VM, see *z/VM: CMS Application Development Guide for Assembler*. For information on VSE/VSAM Data Compression Services, see *VSE/VSAM Version 6 Release 1 User's Guide and Application Programming*.

Application Migration Considerations

An existing application can take advantage of these VSAM Data Compression Services without the need for program changes. The compression controls are in the VSAM product and are not tied to the application code. Two things must be done to migrate existing data sets to compressed format:

1. A 'VSAM.COMPRESS.CONTROL' KSDS compression control data set must be defined in each catalog where compressed data will reside.
2. The existing data set CLUSTER must be redefined as COMPRESS format.

Existing data sets can be unloaded temporarily so that the cluster can be redefined as compressed. The cluster can then be reloaded to create the compressed database which is immediately usable by application programs.

For more information on VSE/VSAM Data Compression Services, see *VSE/VSAM Version 6 Release 1 Commands* and *VSE/VSAM Version 6 Release 1 User's Guide and Application Programming*.

Planning for CMS/DOS

Users of CMS/DOS must, in certain cases, have available a VSE SYSRES. To use either the DOS/VS COBOL or PL/I compilers under CMS/DOS, you must first install a VSE system and install the compilers on this system.

Note: CMS/DOS support is based on the VSE/AF 1.3.5 licensed program and does not support the VSE/AF 2.1 librarian.

If you plan to use CMS/DOS, you must also generate the CMSDOS and CMSBAM saved segments. These segments contain simulated VSE services that are necessary for running VSAM and other VSE programs under CMS. Running VSAM under CMS is dependent on the generation of CMSDOS and CMSBAM.

See the *z/VM: Guide for Automated Installation and Service* for more details on installing CMSDOS and CMSBAM saved segments.

VSE System Generation Considerations

CMS/DOS support in CMS uses a real VSE system disk in read-only mode. CMS/DOS provides the necessary interface, and then gets VSE logical transients and system routines directly from the VSE system libraries. Also, CMS/DOS gets the DOS/VS COBOL and DOS PL/I compilers directly from the VSE system or private core image libraries.

It is your responsibility to install the appropriate VSE system and then generate it. If you plan to use VSE compilers, you must order the DOS/VS COBOL and DOS PL/I optimizing compilers and install them on this VSE system.

When you install the compilers on the VSE system, you must link-edit all the compiler relocatable modules using the linkage editor control statement:

```
ACTION REL
```

You can place link-edited phases in either the system or the private core image library.

When you later invoke compilers from CMS/DOS, the library (system or private) containing the compiler phases must be identified to CMS. You identify all system libraries to CMS using the file mode letter that corresponds to that VSE system disk. Do this by specifying the file mode letter on the SET DOS ON command when you invoke the CMS/DOS environment. You identify a private library by coding ASSGN and DLBL commands that describe it. These VSE system and private disks must be linked to your virtual machine and accessed before you enter the commands to identify them for CMS.

CMS/DOS has no effect on the update procedures for VSE, DOS/VS COBOL, DOS/VS RPG II, or DOS PL/I. You should follow the usual update procedure for applying IBM-distributed coding changes to them.

When the VSE System Must Be Online

Much of what you do in the CMS/DOS environment requires that the VSE system pack and/or the VSE private libraries be available to CMS/DOS. In general, you need these VSE volumes whenever you do the following:

- You use the DOS/VS COBOL or DOS PL/I compilers. These compilers are run from the system or private core image libraries.

- Your DOS/VS COBOL or DOS PL/I source programs contain COPY, LIBRARY, %INCLUDE, or CBL statements. These statements copy code from your system or private source library. This function requires that the CMSBAM shared segment be generated and available to CMS/DOS.
- You invoke one of the librarian programs: DSERV, RSERV, SSERV, PSERV, or ESERV.
- You link-edit VSE programs that use nondisk LIOCS modules. CMS/DOS link-edits LIOCS routines with the VSE program from VSE system or private relocatable libraries.
- You run VSE programs that get phases directly from VSE system or private core-image libraries.

A VSE system pack is usable when it is:

- Defined for your virtual machine
- Accessed
- Specified, by mode letter, on the SET DOS ON command.

A VSE private library is usable when it is:

- Defined for your virtual machine
- Accessed
- Identified by the ASSGN and DLBL commands.

The VSE system pack and private libraries may reside on full packs or minidisks.

CMS/DOS Tape Handling

You can use the CMS tape label processing features described in the *z/VM: CMS Application Development Guide for Assembler* to process tapes defined with a DTFMT. The features described there let you define input and output tapes that have standard or nonstandard labels or are nonlabeled tapes. They also let you specify your own exits for processing user standard or nonstandard labels. Before CMS prepares your tape files for processing, it returns control to the tape label processing routines.

The CMS LABELDEF command, which is described in the *z/VM: CMS Commands and Utilities Reference*, is equivalent to the VSE TLB control statement for standard label tapes.

When a tape is defined as a work file, it is treated as nonlabeled and any labels encountered on the tape are written over.

Tape labels are not supported on tape files defined with DTFCP or DTFDI. Existing IBM standard header labels are bypassed on such tapes when they are used for input and any existing labels are written over when the tapes are used for output.

CMS/DOS Disk Label Information Area

CMS/DOS does not support a disk label information area. If the real VSE system pack used by CMS/DOS has a label information area, it is not used.

In CMS/DOS, ASSGN and DLBL commands provide functions similar to those provided by the VSE ASSGN, DLBL, and EXTENT control statements. In VSE, those control statements are in effect for only one job. Thus, it is convenient to place often used DLBL and EXTENT control statements on the label information area.

VSE and VSAM Planning

However, in CMS/DOS, there is no such thing as a job. Consequently, ASSGN and DLBL commands remain in effect for an entire CMS/DOS session, unless they are reset by another ASSGN or DLBL command. Also, in CMS, you can place all the commands you need to compile and run a program in an exec file and invoke that exec file by its file name.

Chapter 3. Tailoring CMS

This chapter explains the following things that you can do to tailor CMS:

- Use the system profile exec (SYSPROF EXEC) to tailor the CMS environment.
- Use the DEFNUC macro to define responses to the prompts the system issues during the CMS nucleus generation procedure.
- Place file directory information for a shared minidisk into a saved segment.
- Place frequently used execs and XEDIT macros into a saved segment.
- Enlarge the CMS nucleus to contain the Y-STAT.
- Change the name of the CMS saved system.
- Rebuild the CMS nucleus.
- Use the SYSTEM NETID file to configure communication across an NJE network.

Using the SYSPROF EXEC to Tailor the CMS Environment

PI

The system profile (SYSPROF EXEC) is an installation-wide exit that performs some of the CMS initialization function previously done in a module. You can tailor the CMS environment at initialization time by modifying this exec. You can do such things as access additional system disks or bring up application programs automatically. You can also tailor the SYSPROF EXEC to automatically access an SFS directory. See the *z/VM: CMS File Pool Planning, Administration, and Operation* book for more information. Decisions to tailor the environment are made on the basis of user ID, responses to prompting, CMS parameters on the IPL command (see the *z/VM: CP Commands and Utilities Reference* book), or other conditions defined by your installation.

You can also bypass the system profile. The system profile is not meant to force users into an environment. Instead, it enables you to change the default CMS environment without modifying a CMS module and rebuilding CMS.

CMS initialization calls the SYSPROF EXEC and executes it from a saved segment, the system disk, or the system disk extension. This is done before any user disks are accessed. When you enter the IPL CMS command, SYSPROF EXEC is executed unless you:

- Specify the NOSPROF parameter on the command line
- IPL a non-DASD virtual device, such as a reader
- Do not have SYSPROF EXEC in the CMS search order.

The default SYSPROF EXEC creates a standard CMS execution environment.

System Profile Functions

SYSPROF EXEC performs the following functions:

- Processes the parameters passed on the IPL command
- Displays the CMS system identification (system ID)
- Issues the initial console read
- Handles the first command entered at this read
- Accesses either the top directory in the default file pool (if defined) or the 191 minidisk as file mode A

Tailoring CMS

- Issues the S-STAT/Y-STAT messages
- Issues other initialization related messages
- Sets defaults for CMS communications directory support and sets name resolution ON
- Executes the PROFILE EXEC if the user has one.
- Selects the REXX run-time library used for the system.

If a minidisk is defined as virtual device number 192, the following special rules apply:

- If 192 is an unformatted temporary minidisk or virtual disk in storage, it is formatted for CMS and accessed as file mode D.
- If 192 is a CP-formatted temporary minidisk or virtual disk in storage, it is reformatted for CMS use and accessed as file mode D.
- If 192 is a CMS-formatted temporary minidisk, virtual disk in storage, or permanent minidisk accessed as a file mode other than D, it is reaccessed as file mode D.
- If 192 is an unformatted or CP-formatted permanent minidisk, it is not automatically formatted, reformatted, accessed, or reaccessed.

When CMS accesses a 192 minidisk as file mode D, any minidisk or SFS directory already accessed as D is released.

If a protected user drops into CP, CP automatically re-IPLs. (See “Setting Up a Protected Application Environment” on page 33 for a discussion of protected users.) Information indicating the nature of the problem is passed to SYSPROF EXEC. SYSPROF EXEC displays a message when one of these conditions is found. By modifying the exec, however, you can choose a different action, such as sending a message to an administrator.

Modifying the SYSPROF EXEC

You can also modify the SYSPROF EXEC to tailor your system to the requirements of your installation.

Attention

The System Profile completes initialization of the CMS environment so any modifications that you make must be carefully fitted into the existing code to ensure that you don't refer to any and do not make use of the communications directory until after the SET COMDIR commands. Violating this rule may have unintended consequences. For additional information on tailoring your system, see the *z/VM: CMS User's Guide*.

Invoking the IPL CMS Command

You can put the IPL CMS command in your z/VM system directory entry or issue it after you log on. The IPL command has a PARM keyword marking the start of any CMS parameters. These parameters can be up to 64 bytes of data (excluding all leading blank characters after the keyword, PARM, but including all other embedded and trailing blanks).

Note: If you are IPLing a non-DASD device, such as a reader, all CMS parameters are ignored.

All parameters following the PARM keyword, except a valid SAVESYS parameter, are passed to the SYSPROF EXEC. You can modify the SYSPROF EXEC to recognize the passed parameters. When a valid SAVESYS parameter is encountered, an attempt is made to save the system, and the SAVESYS parameter is cleared. When an invalid SAVESYS parameter is encountered, all parameters are ignored, and the SAVESYS error parameter is passed to the SYSPROF EXEC. See the *z/VM: CP Commands and Utilities Reference* for more information on the IPL command.

To properly use the system profile and the CMS parameters, PARMREGS=0-15 must be coded in the DEFSYS command for the CMS named system.

Selecting a REXX Run-time Library

In the SYSPROF EXEC, you can also specify which REXX library will be used to run the IBM-compiled z/VM REXX parts on your system. These parts consist of compiled system REXX execs and REXX Xedit macros.

IBM supplies the DMSTOEAG module, which enables you to use the REXX/370 library program product (EAGRTxxx) instead of the CMS library (DMSRTxxx) on your system. The DMSTOEAG module is used when the REXX/370 Library program product, 5695-014, Release 3 or later, is available on your system. DMSTOEAG makes the name of the runtime library supplied with CMS (DMSRTPRC) an alias of the REXX/370 Library (EAGRTPRC).

Invoking DMSTOEAG

DMSTOEAG has the following syntax:

```

▶▶—DMSTOEAG—————▶▶

```

The module can be invoked at any time but it would usually be invoked from the system profile, SYSPROF EXEC. As Figure 1 on page 30 shows, the SYSPROF \$EXEC file supplied with your system contains instructions for modifying your SYSPROF EXEC.

```

/!* ***** */
/!* */
/!* If the Rexx/370 Library product, 5695-014, is to be used */
/!* instead of DMSRTPRC, do the following: */
/!* */
/!* 1) Make sure DMSRTPRC is not in storage. It won't be in */
/!* storage if no compiled CMS Rexx exec or macro has */
/!* been executed yet. If necessary, you may do the */
/!* following: */
/!* 'NUCXDROP DMSRTPRC' */
/!* */
/!* 2) Load the Rexx/370 Library into storage. You may do */
/!* the following as necessary: */
/!* a) Access the disk or directory containing the */
/!* Rexx/370 Library */
/!* b) Uncomment the following command: */
/!* 'EAGRTPRC' */
/!* */
/!* 3) Uncomment the following command. */
/!* Note that DMSRTPRC must not be in storage at this */
/!* time and EAGRTPRC must be in storage at this time */
/!* and the appropriate EAG message repository set. */
/!* 'DMSTOEAG' */
/!* */
/!* See "z/VM CMS Planning and Administration" for the */
/!* return codes from DMSTOEAG. */
/!* */
/!* ***** */

```

Figure 1. SYSPROF \$EXEC Comments for DMSTOEAG

Usage Notes

1. The CMS library DMSRTPRC must **not** be in storage when DMSTOEAG is invoked. You must purge the segment and drop the nucleus extension, as appropriate.
2. The Rexx/370 Library EAGRTPRC **must** be in storage and the corresponding EAG message repository set when DMSTOEAG is invoked.
 The saved segment that contains EAGRTPRC can be loaded. Or, you can run the EAGRTPRC module to load the library and set the message repository; however, the module will return a non-zero return code. (The EAGRTPRC module is usually invoked by CMS when the first compiled exec is executed and assumes there is an exec to run. Invoking EAGRTPRC directly fails to provide it with an exec to run and the library initialization routine then assumes it was called by an internal routine that requested the address of a certain data field be returned in register 15. The result is a non-zero return code the value of which will vary depending on where the library is loaded in storage.)
3. DMSTOEAG will check that the loaded EAGRTPRC library is at Release 3 or later and that it is not the REXX/370 Alternate Library.
 If the REXX/370 Library product, Release 3 or later, is not available or if only the REXX/370 Alternate Library is available, DMSTOEAG should not be used.
4. DMSTOEAG issues the following return codes:

0	Operation successful; DMSRTPRC is now an alias for EAGRTPRC and EAGRTPRC is a Release 3 or later REXX/370 Library.
1	DMSRTPRC was located in storage (NucExt Query).
2	EAGRTPRC could not be located in storage (NucExt Query).
3	The loaded EAGRTPRC is the REXX/370 Alternate Library.
4	The loaded EAGRTPRC is not a REXX/370 Library.
5	The loaded EAGRTPRC is not at Release 3 or later.

- 6 The set EAG message repository could not be found (LangFind).
 25 Insufficient storage to create an alias (NucExt Set).

How to Change the SYSPROF EXEC

z/VM provides a default SYSPROF EXEC that can reside in a saved segment or on the system disk or system disk extension. If it resides on a system disk, it must have a file mode number of 2. For better performance, the SYSPROF EXEC should reside in a saved segment. However, if it resides in a saved segment and a user IPLs with INSTSEG NO, the SYSPROF EXEC is not invoked unless a copy also resides on the system disk or system disk extension.

The system profile is shipped as file SYSPROF \$EXEC. To change SYSPROF \$EXEC you should use the VMSES/E Local Modification procedure. For more information on the VMSES/E Local Modification procedure, see either the *z/VM: Guide for Automated Installation and Service* or the *z/VM: Service Guide*. This procedure uses the XEDIT command with the CTL option, the VMFEXUPD command and the VMFBLD command to apply an update to the SYSPROF \$EXEC file to generate the updated SYSPROF EXEC.

Note: The SYSPROF \$EXEC is a packed file (1024) on MAINT's 3B2 disk.

If the exec resides on a system disk, it can be changed by anyone who has write access to that disk. After the exec has been modified, if the system is to be IPLed as a named system, or if the S-STAT or Y-STAT capability is to be available, the system must be resaved. If the exec resides in a saved segment, the administrator must follow documented procedures for rebuilding and saving the saved segment after making changes to the exec. In any case, all users currently logged on must re-IPL the system or unpredictable results occur. It is recommended, therefore, that any logic in SYSPROF EXEC that changes frequently should be:

- Put in another exec that is called by the system profile.
- Placed on a disk other than file mode S or file mode Y.

Modifications to execs that reside on other disks do not necessitate the resaving of the system and re-IPL by all users. In SYSPROF EXEC, however, you must remember to access the disks these execs reside on.

Input to the SYSPROF EXEC

CMS initialization passes SYSPROF EXEC a string as an argument. This string is parsed by SYSPROF EXEC into four parts, each part separated by a single byte of X'FF'. Each of these four parts might in turn be parsed for further analysis and action. The four parts and the possible values for each are:

1. Keyword flags which indicate the occurrence of certain events or actions to be taken during CMS initialization. Possible values are:

Parameter	Meaning
AUTOOCR	AUTOOCR parameter specified on IPL command
BATCH	BATCH parameter specified on IPL command
NOSSTAT	S-STATs are unavailable
NOYSTAT	Y-STATs are unavailable
ACCD -or- FORM	Disk at address 192 should be accessed as D. Disk at address 192 was formatted and was accessed as D
NOWM <i>iucv-rc</i>	IUCV failed, with return code <i>iucv-rc</i> ; windowing cannot be started

Parameter	Meaning
SAVERR -or- CONFLICT	SAVESYS specified without system name. SAVESYS specified with other parameters
INSEGER1 -or- INSEGER2 <i>segname</i>	INSTSEG value is missing and no shared execs are loaded The CMS installation saved segment <i>segname</i> could not be loaded
MTSEGER1	MTSEG value is missing and no multitasking segment is loaded
NOFPOOL	Filepool identifier not specified with the FILEPOOL keyword
Note:	
<ol style="list-style-type: none"> 1. If multiple keywords appear in this part, each is separated by at least one blank. 2. Where “-or-” appears, the keywords immediately above and below are mutually exclusive. 3. When NOWM appears, the IUCV return code will be placed after the keyword and at least one blank. 	

2. A keyword and possibly other relevant information indicating whether or not a CMS re-IPL has occurred and the reason. Possible values are:

Parameter	Meaning
CMSErrorR	CP entered; Information unavailable
DWAITPSW <i>psw</i>	CP entered; Disabled wait PSW ‘psw’
EXTINTLP <i>psw</i>	CP entered; External interrupt loop
PAGERROR	CP entered; Paging error
PRGINTLP <i>psw</i>	CP entered; Program interrupt loop
SPAGEALT <i>sname hexloc</i>	CP entered; <i>sname</i> shared page <i>hexloc</i> altered
TRANEXCP	CP entered; Translation exception: While in non-EC mode
NOTREIPL	This was not an automatic re-IPL
Note:	
<ol style="list-style-type: none"> 1. All of the above keywords are mutually exclusive. 2. Where the keyword is followed by additional information (such as <i>psw</i>) there is at least one blank between the keyword and the additional information. In the case of SPAGEALT, at least one blank separates <i>sname</i> and <i>hexloc</i>. 	

3. The text to be displayed to announce the IPL (CMS system version ID). This text is derived from that supplied in the DMSNGP ASSEMBLE file for the VERSION parameter on the DEFNUC MACRO (page 35), or that supplied in response to the DMSINI611R prompt during the CMS nucleus build if VERSION was not specified in DMSNGP ASSEMBLE.
4. The IPL parameters, if any. These parameters include all parameters specified after the PARM keyword on the CP IPL statement. For example, AUTOOCR when an IPL CMS PARM AUTOOCR is executed. The passing of parameters to CMS is documented in the *z/VM: CP Commands and Utilities Reference* under the descriptions of IPL and DEFSYS.

Output from the SYSPROF EXEC

Some of the parameters passed to the SYSPROF EXEC cause messages to be displayed. You can modify SYSPROF EXEC to take further action when one of these parameters is found or to ignore these parameters completely.

When CP re-IPLs, it passes restart information to SYSPROF EXEC. This information indicates the nature of the problem. The default system profile issues a warning message when it finds this information. You can modify the exec, however, to take different action.

How to Bypass the System Profile

To bypass the system profile, you can specify the NOSPROF parameter in the PARM field of the IPL command or IPL a non-DASD device.

A warning message is displayed if the SYSPROF EXEC cannot be located.

Setting Up a Protected Application Environment

If you are interested in using only application programs, you can build a protected application environment. A protected user is automatically placed in an application environment at logon time and prevented from inadvertently dropping into CP. To set up this type of environment, you should use the CONCEAL entry on the OPTION control statement in the user's z/VM system directory. This prevents entry into CP by errors.

You should tailor SYSPROF EXEC, or any of the execs that it calls, to set up the proper application environment for the user. This environment can be set up based on user IDs, parameters passed in the PARM field of the IPL command, or any other function. Tailoring can range from simply accessing additional disks to suppressing normal CMS initialization messages and completely tailoring an application environment.

You should decide what degree of protection is required to make this environment. For example, you may want to turn off the CMS immediate commands such as HX, HI, and TS, for the duration of SYSPROF EXEC. Keep in mind, however, that since this must be done within SYSPROF EXEC, there is only a short period of time when the user can type in one of these immediate commands during CMS initialization.

If the protected user drops into CP, CMS re-IPLs. Information indicating the nature of the problem is passed to SYSPROF EXEC. Based upon the user and the reason for entry into CP, you may want to take action. For example, you may want to inform the system administrator.

What the System Profile Can Do for Installations

The system profile can set up several environments on a single system. Here are some examples of functions that can be done with SYSPROF EXEC at initialization time:

- New parameters can be recognized in the PARM field of the IPL command. For example, you can add the following IPL statement in a user's directory:

```
IPL CMS PARM OFFICE
```

Since OFFICE is an unrecognized parameter, it would be passed to the SYSPROF EXEC where it would be recognized and then set up for the particular user. By recognizing several parameters, different environments can be tailored.

- Additional system disks can be accessed, or some defaults currently provided can be changed.
- Specific users or groups of users can be recognized and placed into an application or other environment.

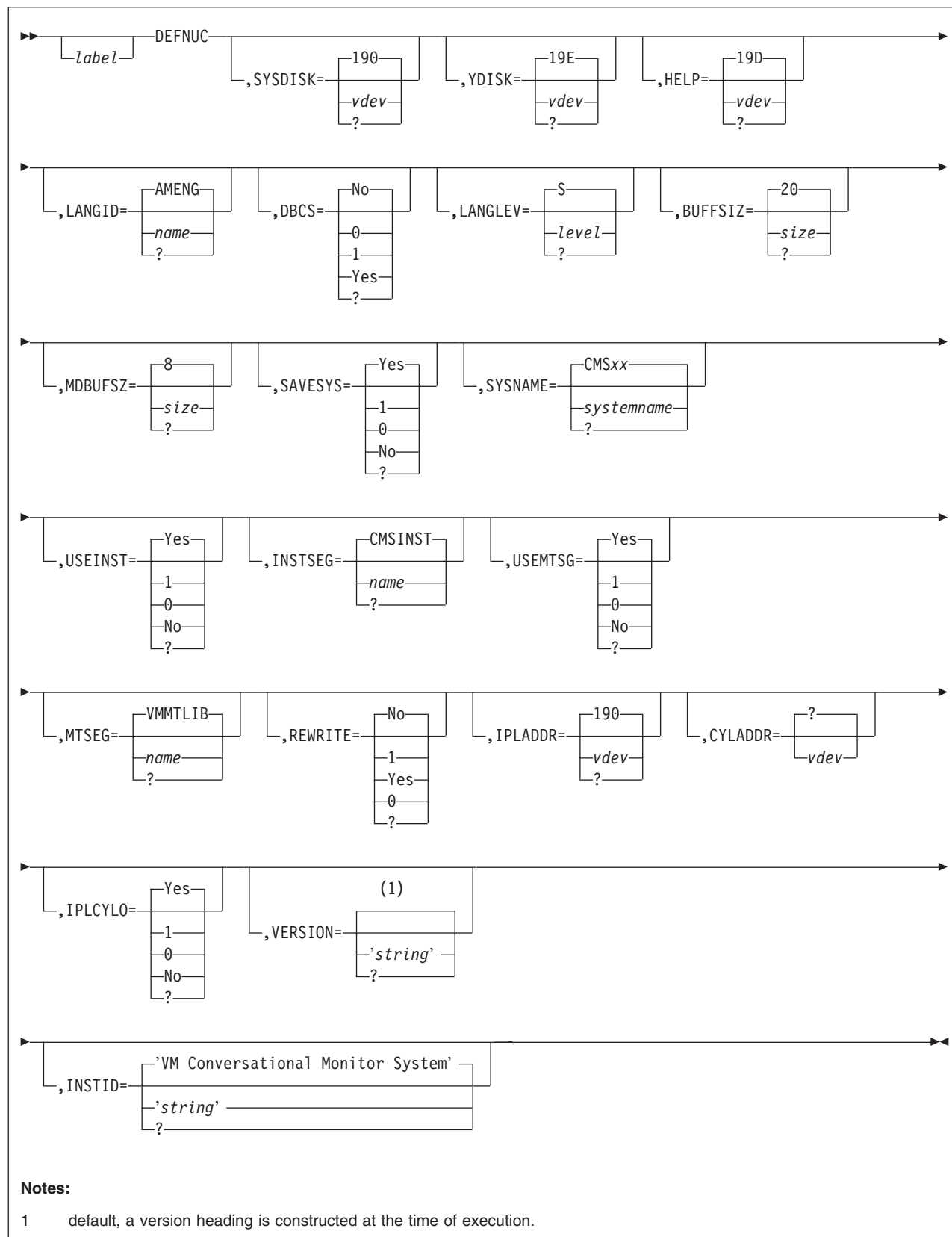
Tailoring CMS

- The initial console read can be suppressed, or the default can be changed to AUTOOCR.
- Novice users can be prompted for information.
- Certain system messages, such as the CMS system version ID, can be suppressed or overridden to hide the complexity of the system.
- Installations can decide for themselves how to handle conditions where protected users enter CP and are re-IPLed. For example, a message can be sent to the system administrator.
- System and user Communication Directory names can be set or changed, and name resolution can be set ON or OFF.

For additional information on the IPL statement and system profile, see page *z/VM: CP Planning and Administration*, and see them used in the *z/VM: Guide for Automated Installation and Service*.

PI end

DEFNUC (Macro)



Purpose: The DEFNUC macro defines responses to the system for prompts issued during the CMS nucleus generation procedure. The expansion of the macro produces the nonexecutable DMSNGP (CMS Nucleus Generation Profile) text file. Figure 2 on page 39 shows an example of how to code a DMSNGP text file. CMS Initialization reads the responses for prompts from the text file. The prompts will be issued if:

- The text deck is not present
- An operand is omitted in the macro
- A question mark is entered as the response
- DMSNGP contains a nonvalid response.

Parameters:

label

is any desired user label

SYSDISK=

defines the system disk address. The response may be four characters or less. A ? indicates that the prompt should be issued. If you specify SYSDISK= without a value, a null response is issued for the prompt and the default value of 190 is used.

YDISK=

defines the Y-disk address. The response may be four characters or less. A ? indicates that the prompt should be issued. If you specify YDISK= without a value, a null response is issued for the prompt and the default value of 19E is used.

HELP=

defines the HELP disk address. The response may be four characters or less. A ? indicates that the prompt should be issued. If you specify HELP= without a value, a null response is issued for the prompt and the default value of 19D is used.

LANGID=

specifies the language identifier of the default language. The response may be five characters or less. A ? indicates that the prompt should be issued. If you specify LANGID= without a value, a null response is issued for the prompt and the default value of AMENG (American English) is used.

DBCS=

specifies whether the system national language is a Double-Byte Character Set (DBCS) language. Enter YES, Y, or 1 to indicate that the default language is a DBCS language. Enter NO, N, or 0 if it is not. A ? indicates that the prompt should be issued. If you specify DBCS= without a value, a null response is issued for the prompt and the default value of NO is used.

LANGLEV=

specifies the level of the saved segment for the default language. The response may be one alphanumeric character, using A to Z or 0 to 9. A ? indicates that the prompt should be issued. If you specify LANGLEV= without a value, a null response is issued for the prompt and the default value of S is used.

BUFFSIZ=

specifies the maximum size in kilobytes to be used by the file system for Read/Write caching for files residing on Shared File System file pools. The response can be any number from 1 to 96. If you specify BUFFSIZ= without a value, a null response and a default size of 20 are issued. A ? indicates that the prompt should be issued.

MDBUFSZ=

specifies the maximum size in kilobytes to be used by the file system for Read/Write caching for files residing on minidisks. The response can be any number from 1 to 96. If you specify MDBUFSZ= without a value, a null response and a default of size of 8 are issued. A ? indicates that the prompt should be issued.

Note that the effective Read/Write cache will be in the range of 1 to 24 disk blocks. Thus, open files on minidisks that are formatted at small block sizes may not take advantage of the maximum cache size specified. Table 8 shows the maximum and minimum effective cache size possible for different disk block sizes.

Table 8. Effective Cache Size

Disk Block Size	Maximum Effective Minidisk Cache	Minimum Effective Minidisk Cache
4096	96KB	4KB
2048	48KB	2KB
1024	24KB	1KB
512	12KB	1KB

SAVESYS=

indicates whether the CMS nucleus is to be saved as a named saved system. Enter YES, Y, or 1 if you plan to save the system. Enter NO, N, or 0 if it will not be saved. response is issued for the prompt and the default value of YES is used. If you specify SAVESYS=, an attempt is made to save an NSS. A ? indicates that the prompt should be issued.

SYSNAME=

specifies the name of the saved system. To specify the system name, enter a valid name using one-to-eight alphanumeric characters, A to Z or 0 to 9. If you specify SYSNAME= without a value, a null response is issued and the default system name is used. This name is dependent on the virtual machine's mode at the time of system build: CMS for XC mode and CMSXA for XA mode. A ? indicates that the prompt should be issued.

USEINST=

indicates whether the CMS installation saved segment will be created. Enter YES, Y, or 1 if you plan to create the installation saved segment. Enter NO, N, or 0 if it will not be created. If you specify USEINST= without a value, a null response is issued for the prompt and the default value of YES is used. A ? indicates that the prompt should be issued.

INSTSEG=

defines the name of the CMS installation saved segment. To name the saved segment, enter a valid name using one-to-eight alphanumeric characters, A to Z or 0 to 9. If you specify INSTSEG= without a value, a null response is issued for the prompt and the default value of CMSINST is used. A ? indicates that the prompt should be issued.

USEMTSG=

indicates whether the VMMLIB logical saved segment residing in the CMSVLIB physical saved segment will be created. Enter YES, Y, or 1 if you plan to create the VMMLIB saved segment. Enter NO, N, or 0 if it will not be created. If you specify USEMTSG= without a value, a null response is issued for the prompt and the default value of YES is used. A ? indicates that the prompt should be issued.

Note: VMMLIB is no longer created as a saved segment; it now resides within the CMS nucleus. Therefore, this parameter has no effect and is retained only for compatibility.

MTSEG=

defines the name of the VMMLIB logical saved segment residing in the CMSVMLIB physical saved segment. To name the logical saved segment, enter a valid name using one-to-eight alphanumeric characters, A to Z or 0 to 9. If you specify MTSEG= without a value, a null response is issued for the prompt and the default value of VMMLIB is used. A ? indicates that the prompt should be issued.

Note: VMMLIB is no longer created as a saved segment; it now resides within the CMS nucleus. Therefore, this parameter has no effect and is retained only for compatibility.

REWRITE=

indicates whether the CMS nucleus should be written to disk. Enter YES, Y, or 1 if you want to write the nucleus to disk. Enter NO, N, or 0 if you do not. If you specify REWRITE= without a value, an MNOTE will appear in the text deck and the prompt will be issued. A ? indicates that the prompt should be issued. The default is YES.

IPLADDR=

defines the disk address at which the nucleus will be written. The response may be three characters or less. If you specify IPLADDR= without a value, an MNOTE will appear in the text deck and the prompt will be issued. A ? indicates that the prompt should be issued. The default disk address is 190.

CYLADDR=

specifies the cylinder or block address at which the nucleus will be written on the IPLADDR disk. The response may be 10 characters or less. If you specify CYLADDR= without a value, an MNOTE will appear in the text deck and the prompt will be issued. The default of ? indicates that the prompt should be issued.

The nucleus resides on the last cylinder(s) or block(s) of the 190 minidisk. The following table shows the relative position from the beginning of the 190 disk, according to DASD type:

Device Type	190 Allocation	Nucleus Size	Cylinder/Block Address
3380	128 cylinders	8 cylinders	Cylinder 120
3390	107 cylinders	7 cylinders	Cylinder 100
9345	128 cylinders	8 cylinders	Cylinder 120
FB-512	153,600 blocks	9,600 blocks	Block 144,000

No response is defined in the sample file. Enter the value from the table that is correct for your system.

IPLCYL0=

indicates whether the cylinder or block zero should also be IPLed. Enter YES, Y, or 1 if you want to IPL the cylinder or block zero. Enter NO, N, or 0 if you do not. If you specify IPLCYL0= without a value, an MNOTE will appear in the text deck and the prompt will be issued. A ? indicates that the prompt should be issued. The default is YES.

VERSION=

defines the version identification that is displayed each time a user IPLs this CMS system. To specify the version heading, enter a valid character string up to 32 characters in single quotation marks. Special assembler characters & and ' require special handling, but may be used if entered twice in the string. For example:

```
VERSION='A&&B' becomes A&B;
VERSION='Don''t worry' becomes Don't worry.
```

If you specify VERSION= without a value, as is the default, a version heading is constructed at the time of execution. A ? indicates that the prompt should be issued.

INSTID=

specifies the heading that appears at the beginning of each output file. To specify the installation heading, enter a valid character string up to 64 characters in single quotation marks. Special assembler characters & and ' require special handling, but may be used if entered twice in the string. For example:

```
INSTID='A&&B' becomes A&B;
INSTID='Don''t worry' becomes Don't worry.
```

If you specify INSTID= without a value, as is the default, an installation heading is constructed at the time of execution, such as VM Conversational Monitor System. A ? indicates that the prompt should be issued.

```
NGP      TITLE 'DMSNGP      (CMS)      Nucleus Generation Profile'
*****
*  COPYRIGHT - CONTAINS RESTRICTED MATERIALS OF IBM      *
*                5654-030 (C) COPYRIGHT IBM CORP 1990,1998 *
*                LICENSED MATERIALS - PROPERTY OF IBM      *
*                REFER TO COPYRIGHT INSTRUCTIONS      *
*****
SPACE 2
DMSNGP  CSECT
DEFNUC  SYSDISK=190,      * S-disk address      *
        YDISK=19E,      * Y-disk address      *
        HELP=19D,      * Help disk address   *
        LANGID=AMENG,   * Default is American English *
        DBCS=NO,      * Default is not a DBCS lang *
        LANGLEV=S,     * Mult. lang. in saved seg.level ID*
        BUFFSIZ=20,    * SFS read/write cache buffer size *
        MDBUFSZ=8,     * Minidisk read/write cache size *
        SAVESYS=NO,    * Save CMS as a named saved system *
        SYSNAME=CMS,   * Name given to named saved system *
        USEINST=YES,   * Use EXEC/XEDIT in a saved segment*
        INSTSEG=CMSINST, * Name of above saved segment *
        USEMTSG=YES,   * Use VMMLIB in a saved segment *
        MTSEG=VMMLIB,  * Name of above saved segment *
        REWRITE=YES,   * Write nucleus to disk *
        IPLADDR=190,   * Address of where to write *
        CYLADDR=?,     * Cyl/Blk of where to write *
        IPLCYLO=YES,   * Write IPL text on cyl 0 *
        VERSION=,     * Version release time date *
        INSTID=      * VM Conversational Monitor System
END
```

Figure 2. Sample Coding for the CMS Nucleus Generation Profile (DMSNGP)

Sharing File Directory Information

You can place file directory information for a shared minidisk into a saved segment. The saved segment is then available to users who link to the minidisk and access it read/only. This helps to reduce your working set as well as the system paging rate and response time.

Note: As a guideline for determining the appropriate size of any saved file directory, multiply the length of a minidisk FST (currently 64 bytes) by the number of files on the disk and convert that to pages.

You can build this saved segment in one of two ways:

- As a logical saved segment. This method uses the SEGGEN command. For information about how to create the records that define a logical saved segment, see *z/VM: CP Planning and Administration*. For information about the SEGGEN command, see the *z/VM: CMS Commands and Utilities Reference*.
- As a member of a segment space or as a discontinuous saved segment. This method uses the SAVEFD command. For information about defining member saved segments and discontinuous saved segments, see *z/VM: CP Planning and Administration*. For information about the SAVEFD command, see the *z/VM: CMS Commands and Utilities Reference*.

Note: If you use SAVEFD to save file directory information in a member of a segment space, you will be unable to resave a file directory when other users on your system are using other members of the segment space. Users must purge the members they are using, or re-IPL, or log off before SAVEFD can be completed successfully. Saving file directory information in a logical saved segment does not have such a restriction.

It is recommended that you define and build your saved segment through VMSES/E. Defining your saved segment to VMSES/E means using the VMFSGMAP EXEC to create a saved segment definition. The advantage of using VMFSGMAP is that it lets you customize the saved segment definition and to see the results in a map of all the saved segments and saved systems defined on your system before you actually build the saved segment. When you are satisfied with the definition, you can use the VMFBLD EXEC to build the saved segment. VMFBLD issues the SEGGEN or SAVEFD command, according to how you have set up the definition.

Sharing Execs and XEDIT Macros

You can place frequently used execs and XEDIT macros into a saved segment. This allows many users to share the same executing copies of the execs and XEDIT macros.

You can build this saved segment in one of two ways:

- As a logical saved segment. This method uses the SEGGEN command. For information about how to create the records that define a logical saved segment, see *z/VM: CP Planning and Administration*. For information about the SEGGEN command, see the *z/VM: CMS Commands and Utilities Reference*.
- As a member of a segment space or as a discontinuous saved segment. This method uses the DCSSGEN command. For information about defining member saved segments and discontinuous saved segments, see *z/VM: CP Planning and Administration*. For information about the DCSSGEN command, see the *z/VM: CMS Commands and Utilities Reference*.

It is recommended that you define and build your saved segment through VMSES/E. Defining your saved segment to VMSES/E means using the VMFSGMAP EXEC to create a saved segment definition. The advantage of using VMFSGMAP is that it lets you customize the saved segment definition and to see the results in a map of all the saved segments and saved systems defined on your system before you actually build the saved segment. When you are satisfied with the definition, you can use the VMFBLD EXEC to build the saved segment. VMFBLD issues the SEGGEN or DCSSGEN command, according to how you have set up the definition.

Enlarging CMS Nucleus to Contain the Minidisk Directory (Y-STAT)

Because CMS is designed to run as a shared system, performance is generally improved when the S and Y minidisk directories (also called the file status tables, or S-STAT and Y-STAT) are included in the CMS nucleus.

When you IPL the CMS saved segment by name (`ipl cms`),

- If you receive these two messages:

```
DMSACC723I Y (19E) R/O
DMSWSP100W Shared Y-STAT not available
```

the 19E disk has changed. You should resave the CMS segment. The following steps show this procedure:

1. **Define the CMS named saved system.** SAMPNSS enters the DEFSYS command to define a skeleton system data file for CMS. Users will receive the new level of CMS whenever they enter the command `IPL cmsname`.

```
access 493 z
sampnss cmsname
```

2. **Save the CMS named saved system.**

```
ipl 190 clear parm savesys cmsname
```

- If you receive this message alone:

```
DMSACC723I Y (19E) R/O
```

the Y-STAT could not fit within the CMS nucleus at the time the system was saved. In that case, every user who IPLs CMS gets the Y minidisk accessed in nonshared user free storage, which is less efficient. If you wish to enlarge the CMS nucleus to contain the Y-STAT, you must expand the CMS nucleus segment by 1MB.

To enlarge the CMS nucleus to contain the Y-STAT, go to “Example of Alternate CMS Nucleus Placement.”

Example of Alternate CMS Nucleus Placement

This section describes how to save your CMS system at a storage location other than the one provided on your System DDR. You might want such an alternative for users with low storage requirements, or when a high nucleus location causes real storage constraints in the construction of segment tables.

The sample CMS saved system is contained in storage location X'F0000'-X'13FFFF'. Before you relocate the nucleus, you should evaluate the trade-off between real storage availability and CMS usage. In addition, when you decide on the optimal CMS location for your user mix, you should consider the size of the page tables that CP builds for the addressable memory for a given machine.

Make sure you have loaded the CMS source minidisk (393).

Example of Alternate CMS Nucleus Placement

To relocate your CMS saved system, follow these steps:

1. **Log on as MAINT and access the CMS minidisks.** The default password for MAINT is MAINT.

```
logon maint
```

2. **Access the CMS minidisks.** Note the file mode of the CMS LOCALMOD minidisk (3C4), the alternate tools disk (493), the production tools disk (193), and the base disk (3B2). You will use it in the next substeps.

```
vmfsetup zvm cms
```

3. **Modify the CMSLOAD EXEC** by creating and applying update files for local modifications. The procedures for applying updates with AUX files are fully discussed in the *z/VM: Service Guide*. The following is an example of this method.

- a. **Create an auxiliary control file for CMSLOAD EXEC.** You will add your local modifications at the top of the file.

```
xedit cmsload auxlcl
```

At the XEDIT command line, enter the following:

```
input ZL0001DS LCL LCL0001 *UPDATE CMSLOAD EXEC
```

In this example, ZL0001DS is the file type of the update file, and LCL0001 is the local tracking number. The rest of the line, beginning with the asterisk (*), is a comment explaining the purpose of this modification.

Now, enter:

```
file = = fm-3c4
```

fm-3c4 is the file mode of the CMS LOCALMOD minidisk (3C4).

- b. The shared portion of the CMS nucleus must reside in a contiguous, 5MB space. The sample CMS saved system is contained in storage location X'F00000'-X'13FFFFFF'. The code from DMSALP to DMSSIG must remain below the 16MB line. The other 4MB of the CMS nucleus may reside either above or below the 16MB line, but the 5MB of CMS nucleus must remain contiguous.

The storage locations where the CMS nucleus is loaded are determined by the SLC (Set Location Counter) statements in the CMS loadlist.

The SLC statement preceding the DMSALP entry defines the starting location of the CMS nucleus. The IBM supplied loadlist statement is SLC LF00000.

The minidisk directories for the S-disk and the Y-disk are saved between DMSSIG and DMSPHI and must remain below the 16MB line. In the IBM supplied loadlist, DMSPHI is placed at the 16MB mark (X'1000000'). The IBM supplied loadlist statement is SLC L1000000. If you are moving all or part of the CMS nucleus below the 16MB line, you must leave enough space between DMSSIG and DMSPHI to contain these minidisk directories. Leaving DMSPHI on a megabyte boundary should leave enough space to save these minidisk directories.

The SLC statement preceding the DMSOME entry defines the end of the CMS nucleus. The IBM supplied loadlist statement is SLC L1400000.

- 1) **Edit the CMS loadlist and change the SLC statements to tailor the location of the CMS nucleus.**

```
xedit cmsload $exec (ctl dmsvm
```

You will receive the following messages only if there are updates for the CMSLOAD file you are editing.

```
DMSXUP178I Applying CMSLOAD ELnnnnDS
```

Example of Alternate CMS Nucleus Placement

You will receive the following messages if your update file cannot be found because you have not created it yet.

```
DMSXUP180W Missing PTF file CMSLOAD ZL0001DS A
```

- 2) **Change the SLC statement that contains the address of the starting location for loading the main part of the CMS nucleus.** The SLC statement that precedes the DMSALP entry in the loadlist contains the address of the starting location for loading the main part of the CMS nucleus.

At the XEDIT command line, enter the following:

```
set case upper
top
locate /&3 DMSALP/
up 2
change / SLC LF00000 / SLC 1n/
```

n is the address of the starting location for the main part of the CMS nucleus.

- 3) **Change the existing SLC statement to the new starting statement.** The SLC statement that precedes the DMSPHI entry in the loadlist marks the start of the high part of the nucleus.

```
top
locate /&3 DMSPHI/
up 2
change / SLC L1000000 / SLC Ln /
```

n is the address of the starting location for the high part of the CMS nucleus.

- 4) **Change the existing SLC statement to the new end statement.** The SLC statement that precedes the DMSOME entry in the loadlist marks the end of the nucleus.

```
top
locate /&3 DMSOME/
up 2
change / SLC L1400000 / SLC Ln /
```

n is the address of the end location of the CMS nucleus.

- 5) **File the CMS loadlist file.**

```
file = = fm-3c4
```

- c. **Invoke the VMFEXUPD command to update the CMS loadlist (CMSLOAD).**

```
vmfexupd cmsload exec {zvm|uceng} cms (logmod filetype excl0001 outmode
localmod sid nohist
```

4. **Create the new SLC files to match the new loadlist.** Repeat this step for each new SLC value you added to the loadlist.

```
xedit SLC Ln fm-3b2
```

n is the address of the starting main location, starting high location, or end location of the CMS nucleus. *fm-3b2* is the file mode of the CMS OBJECT minidisk (3B2).

At the XEDIT command line, enter the following:

```
input $SLC n
set hex on
change/$/X'02'/
fm mode2
file
```

Example of Alternate CMS Nucleus Placement

There must be two blanks between SLC and the address. X'02' is an unprintable loader control character. *mode2* is the file mode of the 3B2 disk concatenated with the number 2. For example, N2, if N is the file mode of the 3B2 disk.

5. **Calculate the new page numbers for the DEFSYS entry for CMS in SAMPNSS.**

- a. To calculate the hexadecimal page numbers where the CMS named saved system begins, take the starting address location and drop the three low-order zeros. This is the new *strtcms* value to use when updating SAMPNSS. For example, X'F00000' becomes X'F00'.
- b. To calculate the hexadecimal page numbers where the CMS named saved system ends, take the ending address location, subtract one, and drop the low-order 12 bits. This is the new *endcms* value to use when updating SAMPNSS. For example, X'1400000' becomes X'13FF'.

6. **Modify the SAMPNSS EXEC to show the new page numbers** by creating and applying update files for local modifications. The procedures for applying updates with AUX files are fully discussed in the *z/VM: Service Guide*. The following is an example of this method.

a. **Create an auxiliary control file for SAMPNSS EXEC.**

```
xedit sampnss auxlcl
```

At the XEDIT command line, enter the following:

```
input EL0001DS LCL LCL0001 *UPDATE SAMPNSS
file = = fm-3c4
```

fm-3c4 is the file mode of the CMS LOCALMOD minidisk (3C4).

b. **Create the update file and apply the updates.**

```
xedit sampnss $exec (ctl dmsvm
```

You will see these messages only if there are updates for the SAMPNSS file you are editing.

```
DMSXUP178I Applying SAMPNSS VnnnnnDS
```

You will see these messages only if your update file cannot be found because you have not created it yet.

```
DMSXUP180W Missing PTF file SAMPNSS EL0001DS A
```

c. **Change the old starting and ending page numbers** to the numbers you calculated in step 5.

At the XEDIT command line, enter the following:

```
locate /DEFSYS/
change /F00-13FF SR/strtcms-endcms SR/
file = = fm-3c4
```

fm-3c4 is the file mode of the CMS LOCALMOD minidisk (3C4).

d. **Invoke the VMFEXUPD command to update SAMPNSS.**

```
vmfexupd sampnss exec zvm cms (logmod outmode localmod sid $select
```

e. **Copy the updated SAMPNSS file to the alternate and production TOOLS disks.**

```
vmfbld ppf zvm cms dmsbl493 sampnss.exec (serviced setup
vmfcopy sampnss exec fm-493 = = fm-193 (prodid 4vmvm430%cms replace olddate
fm-493 is the file mode of the alternate TOOLS disk. fm-193 is the file mode
of the production TOOLS disk.
```

Go to “Rebuilding the CMS Nucleus” on page 45 to rebuild the CMS nucleus.

Updating SAMPNSS to Change the CMS System Name

This section describes how to change the name of the CMS saved system. Make sure you have loaded the CMS source minidisk (393).

Tailor SAMPNSS by creating and applying update files for local modifications. The procedures for applying updates with AUX files are fully discussed in the *z/VM: Service Guide*. An example of this method appears next:

1. **Access the necessary CMS minidisks and create an auxiliary control file for SAMPNSS.** Note the file mode of the CMS LOCALMOD disk (3C4), the alternate TOOLS disk (493), and the production TOOLS disk (193).

```
vmfsetup zvm cms
```

2. **Add your local modifications at the top of SAMPNSS AUXLCL.**

```
xedit sampnss auxlcl
```

At the XEDIT command line, enter the following:

```
input ZLnnnnDS LCL LCLnnnn *UPDATE SAMPNSS
file = = fm-3c4
```

ZLnnnnDS is the file type of the update file, and **LCLnnnn** is the local tracking number. The rest of the line, beginning with the asterisk (*), is a comment explaining the purpose of this modification.

fm-3c4 is the file mode of the CMS LOCALMOD minidisk (3C4).

3. **Create the update file and apply the updates.**

```
xedit sampnss $exec (ctl dmsvm
```

You will see these messages only if there are updates for the SAMPNSS file you are editing.

```
DMSXUP178I Applying SAMPNSS VnnnnnDS
```

You will see this message if your update file cannot be found because you have not created it yet.

```
DMSXUP180W Missing PTF file SAMPNSS ELnnnnDS A
```

4. **Change the CMS named saved system name.**

```
change /CMS/cmsname/* *
```

At the XEDIT command line, enter the following:

```
file = = fm-3c4
```

cmsname is either the IBM supplied system name (CMS) or the name you defined in DMSNGP on the SYSNAME statement. *fm-3c4* is the file mode of the update file on the CMS LOCALMOD disk (3C4).

5. **Invoke the VMFEXUPD command to update SAMPNSS.**

```
vmfexupd sampnss exec zvm cms (logmod outmode localmod sid $select
```

6. **Copy the updated SAMPNSS file to the alternate and production TOOLS disks.**

```
vmfbld ppf zvm cms dmsbl493 sampnss.exec (serviced setup
vmfcopy sampnss exec fm-493 = = fm-193 (prodid 4vmvm430%cms replace olddate
```

fm-493 is the file mode of the alternate TOOLS disk (493). *fm-193* is the file mode of the production TOOLS disk.

Go to “Rebuilding the CMS Nucleus” to rebuild the CMS nucleus.

Rebuilding the CMS Nucleus

This section describes how to rebuild the CMS nucleus.

Rebuilding the CMS Nucleus

1. **Spool the output of your virtual punch and your virtual printer to your virtual reader.**

```
spool punch *
spool print *
```

2. **Create the CMS nucleus.** By issuing VMFBLD, you will incorporate updated CMS text files into a new CMS nucleus. VMFBLD builds the nucleus with the appropriate system default language by using the PPF name corresponding to the LANGID parameter in the DMSNGP profile.

```
vmfbld ppf {zvm|uceng|ger|kanji} cms cmsload (all setup
```

+-----If you receive message VMFBLD2185R-----+

```
VMFBLD2185R The following source product parameter files have been
serviced:
```

```
VMFBLD2185R 4VMVM430 $PPF
```

```
VMFBLD2185R When source product parameter files are serviced,
all product parameter files built from them must be
recompiled using VMFPPF before VMFBLD can be run
```

```
VMFBLD2185R Enter zero (0) to have the serviced source product
parameter files built to your A-disk and exit VMFBLD
so you can recompile your product parameter files
with VMFPPF
```

```
VMFBLD2185R Enter one (1) to continue only if you have already
recompiled your product parameter files with VMFPPF
```

- a. **Build the serviced product parameter files. Enter:**

0

- b. **Recompile your production parameter file.**

```
vmfppf {zvm|uceng|ger|kanji} cms
copyfile 4vmvm430 $ppf a = = fm-51d (olddate replace
erase 4vmvm430 $ppf a
vmfsetup {zvm|uceng|ger|kanji} cms
```

- c. **Reissue the VMFBLD command to build the CMS nucleus.**

```
vmfbld ppf {zvm|uceng|ger|kanji} cms cmsload (all setup
```

```
VMFBLD2185R Enter zero (0) to have the serviced source product
parameter files built to your A-disk and exit VMFBLD
so you can recompile your product parameter files
with VMFPPF
```

```
VMFBLD2185R Enter one (1) to continue only if you have already
recompiled your product parameter files with VMFPPF
```

- d. **Continue. Enter:**

1

+-----End of If you receive message VMFBLD2185R-----+

When your system default national language is uppercase English, 402 will appear here in place of 19D. If your system default national language is Kanji, 401 will appear. If your system default national language is German, 405 will appear.

The CMS nucleus will be sent to your reader.

```
VMFBLD2760I VMFBLD processing started
VMFSET2760I VMFSETUP processing started for ZVM CMS
VMFUTL2205I Minidisk|Directory Assignments:
String Mode Stat Vdev Label|Directory
VMFUTL2205I LOCALMOD E R/W 3C4 MNT3C4
VMFUTL2205I LOCALSAM F R/W 3C2 MNT3C2
VMFUTL2205I APPLY G R/W 3A6 MNT3A6
VMFUTL2205I H R/W 3A4 MNT3A4
```



```

VMFUTL2205I      I      R/W  3A2  MNT3A2
VMFUTL2205I DELTA  J      R/W  3D2  MNT3D2
VMFUTL2205I BUILD7 K      R/W  493  MNT493
VMFUTL2205I BUILD6 L      R/W  490  MNT490

VMFUTL2205I BUILD5 M      R/W  19D  MNT19D
VMFUTL2205I BUILD2 N      R/W  193  MNT193
VMFUTL2205I BASE2  O      R/W  3B2  MNT3B2
VMFUTL2205I BASE3  P      R/W  393  MNT393
VMFUTL2205I ----- A      R/W  191  MNT191
VMFUTL2205I ----- B      R/W  5E5  MNT5E5
VMFUTL2205I ----- D      R/W  51D  MNT51D
VMFUTL2205I ----- S      R/O  190  MNT190
VMFUTL2205I ----- Y/S    R/O  19E  MNT19E

VMFSET2760I VMFSETUP processing completed successfully
VMFBLD1851I Reading build lists
VMFBLD2182I Identifying new build requirements
VMFBLD2182I New build requirements identified

VMFBLD1851I (1 OF 1) VMFBDNUC processing CMSLOAD {EXEC|EXCnnnnn},
      target is BUILD6 490 (L)

LOAD LIST: $$$TLL$$ EXEC      A1 mm/dd/yy hh:mm (MNT191)

RDR FILE fileno SENT FROM MAINT      PUN WAS fileno RECS nnnn CPY 001 A
      NOHOLD NOKEEP

VMFBLD1851I (1 OF 1) VMFBDNUC completed with return code 0
VMFBLD2180I There are n build requirements remaining
VMFBLD2760I VMFBLD processing completed successfully

```

3. **Verify that the CMS nucleus (\$\$\$TLL\$\$ IPL) is in MAINT's virtual reader.** Note the file number (*fileno*) of the CMS nucleus file.

```
query rdr * all
```

4. **Order your reader so that the CMS nucleus (\$\$\$TLL\$\$ IPL) is processed first.**

```
order rdr fileno
```

5. **Change the nucleus reader file status to ensure it remains in the reader after you IPL.**

```
change rdr fileno keep
```

6. **Redefine the 190 and 490 minidisks** so the nucleus will be saved on the 490.

```
define 190 590
define 490 190
```

7. **IPL MAINT's virtual reader to load the new CMS nucleus.** Your virtual machine **must** have at least **21MB** of storage.

```
ipl 00c clear
```

If you changed the SAVESYS operand in DMSNGP to SAVESYS=YES, you will receive message HCPNSS446E. The named saved segment does not need to be saved here. It will be saved in substeps 17 on page 48 and 18 on page 49.

```
HCPNSS446E The Named Saved System (NSS) CMS was not previously
      defined and cannot be saved.
```

The following default version identification displays, unless you define your own version identification. If you defined your own, it displays here and each time that you IPL 490.

```
z/VM V4.3.0   yyy-m-dd hh:mm
```

Redefine the 190 and 490 minidisks.

```
define 190 490
define 590 190
```

Rebuilding the CMS Nucleus

8. **Purge the CMS nucleus from your reader to free spool space.**

```
purge rdr fileno
```

9. **IPL your System disk (190).** Note the file number of the CMS load map (*mfileno*).

You should not IPL your new CMS nucleus (490) until after the saved segments are rebuilt.

```
ipl 190 clear
```

The CMS load map has been sent to MAINT's virtual reader. If you do not get this message, it has already been sent there by a previous IPL 190.

```
RDR FILE mfileno SENT FROM MAINT PRT WAS mfileno  
RECS nnnn CPY 001 A NOHOLD NOKEEP
```

10. **Receive the file from the virtual reader to the alternate TOOLS disk (493).**

You may give the CMS load map any file name and file type you like. The CMS load map on the System DDR is CMSNUC MAP. You may want to erase the old version to save space. The CMS load map is loaded onto the alternate SYSTEM TOOLS disk.

```
access 493 e  
receive mfileno fn ft e (replace
```

11. **If you wish, print a copy of the CMS load map.**

12. **Examine the load map for unresolved symbols.** Unresolved symbols may indicate an error. Make sure that you understand the reason for any unresolved symbols you find before continue.

```
xedit fn ft e
```

On the XEDIT command line, enter:

```
locate /UNRESOLVED/  
qquit
```

13. **If you wish, pack the CMS load map to save minidisk space.**

```
copyfile fn ft e (olddate pack
```

14. **Copy the CMS load map to the production tools disk (193).**

```
access 193 f  
copyfile fn ft e = = f (olddate replace
```

15. **Now, copy the serviced 490 minidisk to the production minidisk (190).**

```
ddr  
    ZVM DASD DUMP/RESTORE PROGRAM  
    ENTER:  
sysprint cons  
    ENTER:  
input 490 dasd mnt490  
    ENTER:  
output 190 dasd mnt190  
    ENTER:  
copy all  
    ENTER:
```

Enter

16. **Relabel the 190 minidisk as MNT190** because the DDR program copied the minidisk label when you copied the 490 minidisk to the 190 minidisk,

```
access 190 r  
format 190 r (label  
Enter the disk label.  
mnt190
```

17. **Invoke SAMPNSS to define the CMS named saved system.**

```
access 493 z
sampsns cmsname
```

18. Save the CMS named saved system.

```
ipl 190 clear parm savesys cmsname
```

Use the SYSTEM NETID File to Configure Communication Across an NJE Network

The SYSTEM NETID file is referenced when you use CMS commands SENDFILE, RECEIVE, TELL, and NOTE to communicate across an NJE network. CMS uses the CPUIDs in the SYSTEM NETID file to determine the NJE node ID and the name of the RSCS virtual machine to be used to send files and messages through the NJE network. The node ID is also used by other subsystems, such as TCP/IP, to determine which configuration to use.

Record Format

The records in the SYSTEM NETID file have the following two formats:

```
cpuid nodeid netid
*comment
```

Operands

cpuid

is the processor (CPU) serial number found in CPUID positions 3-8. If this is an LPAR, the CPU serial number is preceded by the LPAR numbers.

nodeid

is the NJE node ID of this system, if you are using RSCS to communicate through an NJE network. If this system does not participate in an NJE network, then *nodeid* is a unique arbitrary value chosen by you to represent this system.

In order to avoid confusion, it is suggested that it be the same value as the System_ID in the SYSTEM CONFIG file. In fact, if you do not use RSCS, it is suggested that you do not modify the SYSTEM NETID file at all; CMS will automatically use the System_ID as the *nodeid*.

netid

is the user ID of the RSCS virtual machine, as defined in the CP directory.

**comment*

is a comment line. In a comment, each line must begin with an asterisk in column one. A comment line must have a logical record length no greater than 80.

Usage

When you enter commands to communicate across the network, the SYSTEM NETID file is referenced as follows:

1. To transmit notes, files, and messages, the NOTE, SENDFILE, TELL, and RDRLIST commands enter the IDENTIFY command.

SYSTEM NETID File

2. The IDENTIFY command:
 - a. Issues the QUERY CPUID command to retrieve the processor's serial number, and searches the SYSTEM NETID file for a matching serial number.
 - b. Issues the QUERY USERID command to retrieve the node identification, and compares it to the node in the SYSTEM NETID record.
- If there is a conflict in nodes between the SYSTEM NETID file and the response from QUERY USERID, the node in SYSTEM NETID takes precedence.

Separate CPUIDs are generated for each processor in a multiprocessor configuration and for each logical processor in an LPAR configuration. If you plan to run this system on multiple processors or in an LPAR environment, you must do one of these two steps:

- Create a record in the SYSTEM NETID file with the CPUID for each processor that you want to be able to IPL.
 - **OR** update each user's directory to include an OPTION control statement containing the CPUID parameter, and place that CPUID parameter value into a record in the SYSTEM NETID file.
- The value specified on the CPUID parameter overrides all of the actual processor CPUIDs, and allows CMS network communications to function independently of the real processor configuration.

Chapter 4. CMS Pipelines Administration

This chapter describes various aspects of CMS Pipelines administration.

CMS Multitasking Considerations

CMS Pipelines supports CMS multitasking.

Any number of threads may issue PIPE commands concurrently. CMS Pipelines itself does not create processes or threads, and it does not switch between threads. CMS Pipelines uses thread suspend/resume when it is waiting for external events and has determined that CMS Multitasking is active. (Examples of CMS Pipelines stages that use suspend/resume are STARMSG IMMCMD DELAY, TCPCLIENT, TCPDATA and TCPLISTEN; refer to the *z/VM: CMS Pipelines Reference* for more information.)

Note: The SQL/DS product can enter a wait state with a WAITECB, ignoring the suspend/resume condition. This forces the whole virtual machine into a wait.

CMS multitasking support is included in CMS Pipelines without any administrative action required.

Configuring Capabilities

A user may create their own specifications for CMS Pipelines using pipeline configuration variables. For more information, refer to the CONFIGURE stage in the *z/VM: CMS Pipelines Reference*.

Compiling Internals

The code to support CMS Pipelines is already optimized. Further optimization is not necessary. CMS Pipelines files shipped with a file type of EXEC or REXX are intended to be interpreted. Compiling these files may or may not produce error-free compilations.

CMS Pipelines Shared Segment

IBM recommends creating the CMS Pipelines shared segment as a means of improving both performance and usability.

CMS Pipelines Repositories

The CMS Pipelines messages have been moved from the CMS message repository (DMSMES REPOS) to a separate message repository (FPLMES REPOS). As a result, they are no longer contained in the CMS nucleus.

System Message Repository

By default, the CMS Pipelines module and FPLMES REPOS system message repository normally reside in the shared segment named CMSPIPES.

Note: The FPLMES REPOS system message repository resides in an NLS shared segment if an NLS language is used.

CMS Pipelines Administration

This is a physical segment containing a logical segment named PIPMOD. This logical segment contains entries for the DMSPIPE MODULE and the CMS Pipelines FPLMES REPOS message repository.

The FPLMES REPOS file is a “system-level” repository that should not be modified.

IBM-Supplied User Message Repository

A user version of the CMS Pipelines message repository (FPLUME TEXT) is also supplied on the CMS system disk (S-disk), which can be used if CMS Pipelines is not saved in a shared segment. This user message repository contains the same set of CMS Pipelines messages as the system message repository. However, an extra step must be performed for each CMS user on the system. Activation of this user message repository is accomplished the same way it is done for any other user repository, by entering the SET LANGUAGE command (refer to the *z/VM: CMS Commands and Utilities Reference* for more information on the SET LANGUAGE command):

```
SET LANGUAGE ( ADD FPL USER
```

If you have not saved the CMS Pipelines shared segment, you must either add this SET LANGUAGE command to the SYSPROF EXEC or have every CMS user enter this command or add it to their PROFILE EXEC in order to activate this IBM-supplied user message repository.

User Created Message Repository

You can override the system-level CMS Pipelines message repository by creating your own user message repository, FPLUME REPOS. You must process this to a TEXT file (refer to *z/VM: CMS Application Development Guide* for information on processing user message repositories). To activate the FPLUME TEXT user message repository that you created, the same SET LANGUAGE command (as shown under “IBM-Supplied User Message Repository”) must be invoked.

Notes:

1. Only one FPLUME TEXT user repository can be active at one time.
2. If a user message repository was created to add new messages or change existing messages, then those messages will be available only from that repository.

CMS Pipelines Internal Message Table

Users who choose not to create the CMS Pipelines shared segment will still see CMS Pipelines messages even if they have not activated a user message repository. There is an internal message table included with CMS Pipelines. However, these messages exist in English only; national language support is not provided.

Search Order when CMS Pipelines Messages Are Issued

Determining where the CMS Pipelines messages are being issued from depends on the following:

- Is a CMS Pipelines system level repository (FPLMES REPOS) available in a segment (CMSPIPES or NLS)?
- Has a user message repository (FPLUME TEXT) been activated?

The following scenarios provide the source or sources from which CMS Pipelines messages are obtained. If a message is not found at the first source, then the second source is searched and so on.

FPLMES REPOS exists—No FPLUME TEXT active:

1. FPLMES REPOS
2. CMS Pipelines internal message table

FPLMES REPOS exists—FPLUME TEXT is active:

1. FPLUME TEXT
2. FPLMES REPOS
3. CMS Pipelines internal message table

No FPLMES REPOS—FPLUME TEXT is active:

1. FPLUME TEXT
2. CMS Pipelines internal message table

No FPLMES REPOS—No FPLUME TEXT:

1. CMS Pipelines internal message table

SQL/DS Access

The remainder of this chapter describes how to set up CMS Pipelines for SQL/DS access.

Note: This procedure pertains to SQL/DS release 3.3.0 or later and DB2® Server for VSE & VM version 5, release 1.0 or later. References made in this procedure to SQL/DS apply to both products and the respective release qualifications for each. Earlier releases of SQL/DS are not supported by CMS Pipelines.

These tasks are done once for all CMS Pipelines users by a system or database administrator. If you are not a system or database administrator or you do not want to provide SQL/DS access for your CMS Pipelines users, skip this section.

To provide SQL/DS access, you must identify CMS Pipelines to SQL/DS. When CMS Pipelines is prepared for SQL/DS access, users can access SQL/DS through the CMS Pipelines SQL and SQLCODES stages. Users must also have the appropriate SQL/DS authorities to access an SQL/DS database (such as connect authority). The SQL and SQLCODES stages are described in the *z/VM: CMS Pipelines Reference*. Additional guidance on SQL/DS access is described in the *z/VM: CMS Pipelines User's Guide*.

Usually these administrative procedures are performed from the database administrator virtual machine. Your installation may, however, be configured differently. Use the virtual machine that is appropriate for your installation.

General Set Up

Before continuing with any or all of the sections below, you must obtain the correct access order for CMS by issuing:

```
vmfsetup esa cms
```

Identify CMS Pipelines to SQL/DS

This section describes how to prepare CMS Pipelines for SQL/DS access. You should be familiar with the SQL/DS product. Please read through the instructions before trying them.

CMS Pipelines Administration

If CMS Pipelines is to communicate with more than one SQL/DS machine, this section must be completed for each SQL/DS machine.

Attention:

1. The DMSPIPE user ID can be modified by setting the SQLpgrowner configuration variable with the CONFIGURE stage. The procedure in this section depicts an unmodified DMSPIPE user ID. For more information, refer to the *z/VM: CMS Pipelines Reference*.
2. The DMSPQI variable can be modified by setting the SQLpgrname configuration variable with the CONFIGURE stage. The procedure in this section depicts an unmodified DMSPQI variable. For more information, refer to the *z/VM: CMS Pipelines Reference*.

To prepare CMS Pipelines for use with SQL/DS do the following:

1. From any database administrator user ID, grant CONNECT authority to CMS Pipelines for the SQL/DS database to be accessed by issuing the following command:

```
grant connect to dmspipe identified by dbpassword
```

where *dbpassword* is the password associated with the DMSPIPE user ID.

The default CMS Pipelines user ID for the database is DMSPIPE.

2. Access the SQL/DS production disk by entering the following LINK and ACCESS commands:

```
link sqlid vdev 195 rr  
access 195 q/a
```

where *sqlid* is the user ID that owns the SQL/DS production disk. *vdev* is the virtual device number of the SQL/DS production disk, usually 195.

Notes:

- a. You may also need to know the RR password for the SQL/DS production disk.
 - b. While these commands use **195** as the virtual address and **q** as the file mode, you can substitute whatever address and file mode is appropriate for your installation.
3. Access the SQL/DS message repository by entering:

```
set language ameng (add ari user
```

4. Establish the name of the SQL/DS database you want to access by issuing the following command:

```
sqlinit dbname(dbname)
```

where *dbname* is the database name.

5. Run the SQL/DS preprocessor.

The code in CMS Pipelines that interacts with SQL/DS must be preprocessed by using the SQLPREP EXEC supplied with SQL/DS. It resides on the SQL/DS production disk that you linked and accessed in step 2.

SQLPREP creates an SQL/DS access module in the database. It also creates a source file that can be assembled. The code to be preprocessed is in the CMS Pipelines file DMSPQI ASMSQL, which can be found on the system tools disk (usually MAINT 193). The source file that will be created is DMSPQI ASSEMBLE.

Issue the SQLPREP command:


```
sqlprep asm pp(prepare=dmspqi userid=dmspipe/dbpassword  
block isolation(user)) in(dmspqi)
```

where *dbpassword* is the password associated with the DMSPIPE user ID that you used in step 1 on page 54.

Note: The file DMSPQI ASSEMBLE and the two SQL files (ARIxxx MODULE) generated by the SQLPREP command on the A-disk can be erased.

6. Grant authority to the access module.

CMS Pipelines users need to be able to run the access module created in the preceding step. From the database administrator's user ID issue the following SQL/DS statements to grant authority to all users:

```
connect dmspipe identified by dbpassword  
grant run on dmspqi to public
```

where *dbpassword* is the password associated with the DMSPIPE user ID that you used in step 1 on page 54.

Notes:

1. Each user must have proper authorization to connect to the SQL/DS database and to manipulate data. The database administrator grants authorization to the database.
2. If CMS is to communicate with more than one SQL/DS machine, the access module must be created on all the SQL/DS machines by repeating the preceding steps for each machine.

You have now completed preparing CMS Pipelines to work with SQL/DS and DB2 Server for VM.

Chapter 5. The Programmable Operator Facility

The Programmable Operator Facility is designed to increase the efficiency of system operation and to allow remote operation of systems in a distributed data processing environment. It does this by intercepting all messages and requests directed to its virtual machine and by handling them according to preprogrammed actions. It determines whether a message is to be simply recorded for future reference, whether the message is to be acted upon, or whether the message is to be sent on to the operator to handle.

Note: The programmable operator should always run with American English as the system national language. Routing tables and messages to the programmable operator should always be in American English to ensure that the uppercasing and routing table comparisons are handled correctly.

The tasks that can be performed by the programmable operator facility include:

- Logging messages
- Suppressing message display and routing messages to a logical (real) operator
- Executing commands
- Responding with preprogrammed message responses.

Using the Programmable Operator Facility in Various System Environments

The programmable operator facility can be used in the following environments:

- Single system
- Distributed z/VM system
- Mixed systems.

Use in a Single System

When the programmable operator facility is operational in a single-system environment, it can:

- Ease message traffic to the system operator, by:
 - Filtering (logging) nonessential, information-only messages
 - Routing messages (for example, I/O intervention requests) to someone else for specialized action.
- Increase productivity, by freeing the system operator from certain *routine* responses or tasks. Such responses (whether they consist of one or a series of commands, whether z/VM or guest operating system) may be preprogrammed to execute automatically upon receipt of a given message.

Thus, only essential, nonroutine messages (that is, those requiring the skill and experience of a system operator to handle) are sent on to the operator for response or action.

Use in Distributed z/VM Systems

The capabilities of the programmable operator, outlined above, also allow for the remote operation of systems in a distributed z/VM environment. When the programmable operator facility is operational in a distributed z/VM system, it can:

- Issue responses and perform tasks that do not require an on-site operator
- Filter (log) nonessential, information-only messages

Programmable Operator Facility

- Route messages requiring on-site (that is, manual) intervention to someone, not necessarily an operator, at the distributed site for action
- Route messages that require the skill and experience of a system operator to handle to the operator at the host system. The operator at the host site can also send commands to the programmable operator facility to control its operation, as well as commands to execute on the distributed system to control the system itself.

By running the programmable operator facility on z/VM systems distributed at several different locations (network nodes), one operator at a host site can control a network of systems.

Use in a Mixed Environment

The programmable operator facility also provides for distributed data processing in an SNA environment with mixed z/VM, OS/VS, and VSE distributed systems and host systems. This is called a mixed environment. The Programmable Operator/NCCF Message Exchange (PMX) provides an interface with the Network Communications Control Facility (NCCF) or the NetView[®] program so that an NCCF or NetView operator can operate a z/VM distributed system, whether the operator is on a z/VM, OS/VS, or VSE system.

Notes:

1. The Group Control System (GCS) is a requirement to use the programmable operator in a mixed environment. The Programmable Operator/NCCF Message Exchange (PMX) uses facilities unique to GCS, and therefore cannot run on any other supervisor. For more information on GCS, see the *z/VM: Group Control System*.
2. NCCF and NetView are network management VTAM applications. NCCF requires the Advanced Communications Function/Virtual Telecommunications Access Method (ACF/VTAM) Version 3 for z/VM. NetView requires the Advanced Communications Function/Virtual Telecommunications Access Method (ACF/VTAM) Version 3 Release 1 Modification Level 1 for z/VM. NCCF and NetView run on subsequent releases of ACF/VTAM unless otherwise stated.
3. **The programmable operator facility does not allow a user logged on to a z/VM virtual machine to enter NCCF or NetView commands.** A z/VM user cannot try to control NCCF or NetView as an authorized operator. The system only lets authorized NCCF or NetView operators enter z/VM commands within the programmable operator virtual machine.
4. The maximum message length that the programmable operator facility can process is 240 bytes. Messages received that exceed this limit are truncated and the remainder is rejected. Normal processing continues for the portion that is received.

The Logical Operator

Occasionally the programmable operator must send messages to another virtual machine. To ensure that the programmable operator will function properly, a user (other than the programmable operator virtual machine on the local system, in a distributed system, or in a mixed environment) is identified to the programmable operator to receive these messages. This user is called the logical operator, as opposed to the CP system operator. When the programmable operator starts (in the CP system operator virtual machine, for example), the logical operator receives an initiation message. The logical operator also receives error messages for severe errors, such as logging errors, and receives all messages routed to the logical operator explicitly or by default.

In a mixed environment, an NCCF or NetView operator can be assigned as the logical operator to control a z/VM distributed system. For more information, see “The NCCF or NetView Logical Operator” on page 71.

The default logical operator is specified in the LGLOPR statement of the routing table. However, other logical operators can be dynamically assigned, released, or replaced using the LGLOPR command of the programmable operator facility. Both methods of identifying the logical operator are described in detail later in this section.

Routing Table Information

The programmable operator routing table identifies the programmable operator facility environment, including the default logical operator’s user ID and node ID. It also specifies the action to take for each message, and authorizes certain users to invoke specific programmable operator commands. For a complete description of the information contained in a routing table, see “The Routing Table” on page 74.

The routing table is a separate CMS file that must be tailored for a specific use. The first routing table to be used is specified when the programmable operator facility is invoked. If no routing table name is specified, the default file name PROP is used.

The installation may define multiple routing tables to cover varying situations. For example, multiple routing tables can be defined to cover shift changes. Only one routing table can be active at a time. The active routing table may be replaced by issuing the LOADTBL command. Any person authorized in the active routing table may enter this command.

Action Routines

Action routines are programs or execs that receive control in response to the match of a message and a routing table entry. They handle a particular type of message or command intercepted by the programmable operator facility. A set of action routines is provided with the programmable operator facility. These need no tailoring to provide you with the control and function needed to operate the programmable operator facility. You can extend the programmable operator facility by writing a new action routine and adding it to the appropriate routing table. Action routines can be execs or written in Basic Assembler Language. (Basic Assembler Language action routines must also be added to the PROPLIB LOADLIB. The PROPLIB LOADLIB is built by the VMSES/E VMFBLD EXEC using the DMSBLPRP build list.)

Exit Execs

Exit execs receive control from the programmable operator facility in certain error and communication status situations. IBM supplies sample exit execs that can be used in their sample form, modified, or replaced with user-written exit execs.

How It Works

The programmable operator facility runs in a CMS virtual machine. Although it can run in any virtual machine, because of its programmed capability to log, handle, or redirect messages, it is most commonly run in the CP system operator’s virtual machine.

Programmable Operator Facility

The programmable operator facility compares all messages directed to it against entries listed in a routing table (a CMS file). When a match occurs, the prescribed action is performed. Any messages requiring a real operator's response or action are sent on to the defined operator (system, network, NCCF, and other defined operators) at another console, a *logical* operator console. If the logical operator is on a virtual machine in the same system, the programmable operator sends the messages with either the CP MESSAGE or CP MSGNOH command. If the logical operator is on a virtual machine in a different z/VM system (network node), a host system for example, it sends the messages using RSCS.

Consider this example: The OPER1 user ID is defined as the CP system operator when the CP system is configured. Set up the programmable operator to run in the OPER1 virtual machine and establish another virtual machine with a user ID of OPERX. In the routing table file(s), specify OPERX as the logical operator. Now any CP or user messages sent to the system operator virtual machine can be handled or filtered by the programmable operator or routed to the OPERX user ID.

Note: The logical operator cannot receive the messages when the logical operator virtual machine is not receiving, such as (but not limited to), being logged off, disconnected, has SET MSG OFF, or passed through to another system. If another user sends messages to the virtual machine running the programmable operator, the originator of the messages receives message HCP045 or HCP057. This happens because the TELL command, which sends the messages from the programmable operator virtual machine to the logical operator virtual machine, verifies the ability of the logical operator to receive messages.

If the logical operator is an NCCF or NetView operator, the programmable operator sends messages through the Programmable Operator/NCCF Message Exchange (PMX) portion of the programmable operator facility.

Flow of Operation

When the programmable operator facility is running in a virtual machine, CP intercepts all messages intended for that virtual machine console. CP then passes these messages to the programmable operator facility by IUCV (the maximum number of outstanding messages that IUCV can queue for presentation to the programmable operator facility is 65535). The messages are logged in a CMS file. The programmable operator facility then uses the active routing table to analyze the message and determine if further action is needed. Based on the contents of the routing table (such as message texts, message types, and user authorizations), the message can be passed to some specified action routine for further action.

Note: When the programmable operator facility is designated to run in the CP system operator virtual machine, message filtering is affected by the CP message handler. Message filtering will only occur when the programmable operator facility is running in the virtual machine recognized as the primary system operator. For more information, see the MESSAGE command in the *z/VM: CP Commands and Utilities Reference* book.

Before the message is logged or passed to any action routines, the programmable operator facility deletes any leading blanks from the message text. Thus, the message text is considered to start at its first nonblank character.

If the message is to be routed to the logical operator, and that person is on another virtual machine in the same physical machine, the programmable operator facility

routes the message directly to the logical operator. To do this, it uses the CP MSGNOH or CP MESSAGE command depending on the classification of the programmable operator virtual machine. If the logical operator is on a different physical machine, the programmable operator facility prefaces the message with the appropriate tag information and sends the message to RSCS using the CP SMSG command. If the logical operator is an NCCF or NetView operator, the programmable operator facility sends the message through the Programmable Operator/NCCF Message Exchange (PMX) which passes the message on to the NCCF or NetView logical operator.

The programmable operator facility usually operates in a disconnected virtual machine. If someone logs on to this disconnected virtual machine with the programmable operator facility running, no messages are displayed (unless the programmable operator facility is running in DEBUG mode). All messages are being intercepted or received by the programmable operator program from IUCV. If that person should enter a command, the programmable operator facility gets control and reads the command entered. Only two commands are accepted from this environment; the STOP command and the SET command. The programmable operator facility rejects any other commands. However, in some situations the redisplay of the entered command is CP console I/O and is presented to the programmable operator facility as a type-3 message. If the text of this command matches a record in the active routing table, the programmable operator facility may invoke an action routine.

If a CMS ABEND occurs while the programmable operator facility is executing, all files are closed and abend error messages are sent to the logical operator. A dump of the virtual machine storage is taken using the CP VMDUMP command and the last system or device that was IPLed is re-IPLed. If the abend occurs while an action routine is executing, abend error messages are sent to the logical operator and the requester (if any). Control is returned to the point in the programmable operator facility immediately following the action routine call.

Relationship to RSCS

When the programmable operator facility is running in a z/VM network environment, it is a normal user of RSCS Networking. This means that the programmable operator facility communicates to RSCS using the CP SMSG command. Any configuration of systems and networks supported by RSCS Networking can use the programmable operator facility. The time needed for a message to go from the system at a distributed site to the logical operator at the host system, or to go from the logical operator at the host system to the system at a distributed site, depends on the number and type of communications links between the message sources and destinations.

When the logical operator is an NCCF or NetView operator, the programmable operator does not use RSCS to route messages to the logical operator. It, instead, passes the messages to NCCF or NetView through the Message Queuing Service so that the messages are presented at the appropriate NCCF or NetView operator console.

A programmable operator can check on its ability to communicate with a host or distributed system. See "Communications Checking" on page 94.

Programmable Operator Virtual Machine

This section describes how to install, invoke, and stop the programmable operator facility. It also tells you about the messages that it sends and their format.

Installing the Programmable Operator Facility

The z/VM product contains the file PROPLIB LOADLIB which is the basis for the programmable operator facility. After receiving and installing z/VM, take the following steps before running the programmable operator facility.

1. The programmable operator facility manages the log file(s) and feedback file. It manages these files in the virtual machine in which the programmable operator will be running. You should ensure that the virtual machine has enough free space on a minidisk or in an SFS file space to contain the files. The amount of space needed depends on the amount of message traffic that will be going through the programmable operator facility, and on the number of comments you expect users to place in the log and feedback files. To help you determine the amount of space needed for the log and feedback files, see “The Log File” on page 90 and “The Feedback File” on page 93.
2. The sample routing table is located on the CMS 190 minidisk. To use the sample PROP RTABLE, take the following steps:

```
access 190 c/a
copyfile prop rtable c = = a
```

This places the sample routing table on the minidisk or SFS directory accessed as A. Edit the sample routing table (PROP RTABLE) to include the functions and authorizations to meet the various needs of the installation. For more information on tailoring the routing table, see “The Routing Table” on page 74. Place the edited file on a minidisk or SFS directory accessed by the programmable operator facility virtual machine.

3. PROPLIB LOADLIB is built by the VMSES/E VMFBLD EXEC using the DMSBLPRP build list. If you need to make any changes to the supplied action routines, you must create local modifications to those routines. If you want to add your own routines, you must provide a local modification to the DMSBLPRP build list. See the discussion on installing local service in the *z/VM: Service Guide* for information on creating local modifications.

Action routines written in Basic Assembler Language must be put in the PROPLIB LOADLIB. Exec action routines need not be put in the PROPLIB LOADLIB, but can reside on any minidisk or SFS directory accessible to the programmable operator.

If you are operating in a mixed environment and need the Programmable Operator/NCCF Message Exchange (PMX) to route messages to the NCCF or NetView logical operator, you must first install the PMX. For details on the installation procedure, see “Installing the PMX” on page 68.

Invoking the Programmable Operator Facility

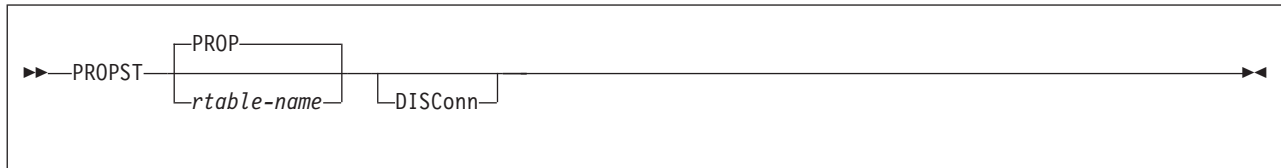
You can invoke the programmable operator facility manually or you can set it up to start automatically.

Manual Invocation

Before loading and invoking the programmable operator facility, IPL CMS in the virtual machine that will be running the programmable operator facility.

Use the PROPST EXEC to invoke the programmable operator facility manually. The PROPST EXEC drops *any* IBM-supplied programmable operator routines that are

currently loaded as a nucleus extension, and loads the programmable operator as a nucleus extension. It then invokes the programmable operator facility with the specified RTABLE. If you do not specify a routing table, the default RTABLE name is PROP. You may specify a disconnect parameter to disconnect the programmable operator before it is invoked. PROPST EXEC is located on the CMS System Disk (190) and can be tailored to your needs. The format of the invocation exec is as follows:



Alternatively, you can take the following steps before each time you invoke the programmable operator facility:

1. Enter the following FILEDEF command to assign a CMS file name to the PROPLIB LOADLIB file so CMS can read and load from it:

```
filedef proplib disk proplib loadlib *
```
2. Next, load the programmable operator program as a CMS nucleus extension by the NUCXLOAD command. Enter the command as follows:

```
nucxload prop dmspop proplib
```

(See the *z/VM: CMS Commands and Utilities Reference* for more details on the NUCXLOAD command.) These first two steps may be omitted for subsequent invocations as long as you do not:

- IPL CMS
 - Have a CMS ABEND from which the programmable operator does not automatically recover.
3. Following its loading as a CMS nucleus extension, invoke the programmable operator facility as if it were a CMS command. The format of the invocation is:

```
PROP rtable-name
```

rtable-name is the file name of the routing table that is to be used for the programmable operator facility. PROP is the default file name of the routing table if no other is specified at invocation.

Automatic Invocation

If you wish, you can set up the programmable operator facility to start running when the system is IPLed and to restart automatically in the event of CP system restart. This can be done as follows:

1. Place an IPL CMS PARM AUTOOCR entry for programmable operator's virtual machine in the z/VM system directory. You can do this even if the programmable operator virtual machine is the CP system operator.
2. Place the following entry in the PROFILE EXEC of the programmable operator's virtual machine:

```
EXEC PROPST rtable-name [DISConn]
```

You can precede or follow the invocation of the PROPST EXEC by issuing any virtual machine commands that you wish to have executed before or after the programmable operator facility is invoked. Virtual machine commands that are placed after the invocation of the PROPST EXEC are not executed until the programmable operator facility is stopped.

Programmable Operator Facility

3. If you want the programmable operator to run in other than the operator's virtual machine, place an AUTOLOG entry for the programmable operator's virtual machine in the PROFILE EXEC of the system operator or the AUTOLOG1 user.
4. After this is complete, if the logical operator is not already logged on, he should do so on the appropriate system.

The following example shows how to place entries in the z/VM system directory and the PROFILE EXEC of operator's virtual machine. These entries automatically invoke the programmable operator facility in the operator's virtual machine when the system is IPLed. The user ID of the programmable operator virtual machine is OPERATOR. The default, PROP RTABLE, is the name of the routing table being used.

- z/VM system directory entries:

```
⋮  
USER OPERATOR password 3M 3M ABCDEFG  
IPL CMS PARM AUTOCR  
⋮
```

- Entries in the PROFILE EXEC of the Operator's virtual machine:

```
⋮  
EXEC PROPST DISCONN  
⋮
```

After these changes (or similar ones) have been made, IPLing the system causes the programmable operator to be invoked automatically in the disconnected system operator virtual machine. After the DISCONNECTED message has been written to the console, indicating that the system operator virtual machine is disconnected, the operator can log on to whatever virtual machine is normally used, for example, the default logical operator virtual machine specified in the routing table.

Initialization

The programmable operator facility is initiated by IPLing CMS in a virtual machine and invoking the programmable operator facility. When the programmable operator facility gets control, it locates the specified or default routing table and loads it into virtual storage. The action routines named in this routing table are in turn loaded as CMS nucleus extensions. For each action routine specified in the routing table, an exec file or a corresponding member in a CMS simulated OS load library named PROPLIB LOADLIB must exist. If an exec does not exist, the LOADLIB member is loaded as a nucleus extension by the NUCXLOAD command. If both exist, the exec takes precedence.

If upon invocation, the programmable operator facility cannot find an action routine named in the routing table, an error message is issued, and, after displaying all detectable routing table errors, the programmable operator facility terminates operation. Otherwise, the programmable operator facility is fully initialized, and writes a message to the programmable operator's console, to the logical operator, and to the LOG file, indicating that the programmable operator facility has started.

The programmable operator facility then waits for either an incoming message or a programmable operator console command (STOP and SET are the only valid commands). The operator can disconnect at this point by having specified the DISConn parameter for the PROPST EXEC, by entering CP (pressing the PA1 key or equivalent) and typing CP DISCONN or by issuing #CP DISCONN where # is the logical line end character. After the DISCONNECTED message has been written to the console, indicating that the system operator virtual machine is disconnected, the

operator can log on to whatever virtual machine is normally used, for example, the default logical operator virtual machine specified in the routing table.

PI

When the programmable operator starts, it executes an exec called PROPPROF, which is located on the CMS System Disk (190). This exec can be copied to file mode A and tailored. PROPPROF is called after the programmable operator initialization is complete, that is, after the programmable operator message environment has been set up. PROPPROF tailors the programmable operator message environment to your specific needs.

For example, to prevent the programmable operator from trapping warning messages, PROPPROF could look like this:

```
/* PROPPROF Example */  
"CP SET WNG ON"  
Exit
```

Parameters passed to the PROPPROF EXEC are listed 104. Some do not have meaning to the PROPPROF EXEC. These parameters are the requester user ID, requester node ID, and the message class.

PROPPROF can be used for any special programmable operator start processing desired without having to rewrite the PROPST EXEC or create a new exec to invoke the programmable operator facility. After executing PROPPROF, the programmable operator then begins routing messages as defined by the routing table entries and the LGLOPR statement in the specified routing table.

PI end

When the programmable operator facility is automatically restarted following an abend, it, if possible, uses the routing table in effect at the time of the abend rather than that specified at invocation. Also, if a logical operator (other than the default logical operator) had been assigned, that assignment is restored.

Note: If the user enters a node ID into the SYSTEM NETID file that is not valid as a CMS file type, the programmable operator cannot start because it is not able to open the log file. For more information, see "Use the SYSTEM NETID File to Configure Communication Across an NJE Network" on page 49.

How the Programmable Operator Establishes Communications with IUCV

The programmable operator facility automatically establishes communications with CP through the Inter-User Communications Vehicle (IUCV). When the programmable operator facility is initialized, a CMSIUCV CONNECT, specifying *MSG and an application ID of PROP, is issued to establish the communications path with the Message System Service. This allows the programmable operator program to read and evaluate messages directly from CP.

Several CP command settings determine the types of messages that the programmable operator facility can receive. The programmable operator facility issues these SET commands when initializing, and resets them when terminating. The commands issued during initialization are:

Programmable Operator Facility

```
SET MSG IUCV
SET WNG IUCV
SET SMSG IUCV
SET EMSG IUCV
SET IMSG IUCV
SET CPCONIO IUCV
SET VMCONIO OFF
```

VMCONIO is set OFF so that any messages produced by CMS or the programmable operator during initialization of the programmable operator facility are typed on its virtual machine console. If VMCONIO was set to IUCV, such data would be trapped by IUCV and not displayed.

When the programmable operator STOP command is issued, the following SET commands are issued:

```
SET MSG ON
SET WNG ON
SET SMSG OFF
SET EMSG ON
SET IMSG ON
SET CPCONIO OFF
SET VMCONIO OFF
```

Then, after the existing messages are handled, the IUCV connection is severed using the IUCV SEVER function.

Some other virtual machine settings that the programmable operator facility modifies are SET RUN ON, SET TIMER REAL, and TERMINAL MODE VM at initialization, and SET RUN OFF and SET TIMER ON at termination. (The SET TIMER command is not issued when the machine is running in XC mode.) SET RUN ON is issued to ensure that the programmable operator is not held up in CP console function mode for excessive periods of time, either because of some operator command entry or because of logging on to a disconnected programmable operator virtual machine. TERMINAL MODE VM is to ensure that programmable operator console commands are handled correctly.

Notes:

1. Single Console Image Facility (SCIF) operation supersedes IUCV Message System Service operation. If the programmable operator virtual machine has a SCIF secondary user, messages would be sent by SCIF to the secondary user rather than handled by the programmable operator virtual machine through the IUCV Message System Service. However, the programmable operator facility may be a SCIF secondary user for another virtual machine. For example, this can be used to control the operation of a guest operating system running in another virtual machine. In this case, SCIF messages are presented to the programmable operator virtual machine as IUCV message-type 8.
2. If you plan to run the programmable operator facility in the operator's virtual machine, you must remove the secondary user from the operator virtual machine z/VM system directory entry. Otherwise, the Single Console Image Facility (SCIF) overrides the programmable operator facility.
3. The receive buffer declared for IUCV is 240 bytes. Therefore, only the first 240 characters of a message are processed. The programmable operator facility rejects (IUCV REJECT) the remainder of any message that exceeds 240 bytes.

Message Output Format

The messages and responses from the programmable operator facility are sent to local z/VM users by the CP MSGNOH command if the programmable operator

virtual machine has user class B authorization. Otherwise, the CP MESSAGE command is used. Regardless of which message command is used the messages from another user that are routed to the logical operator are prefixed with the user ID and node ID of the originating user. The format of these messages appears as follows:

```
col 1      col 10     col 20
|          |          |
userid    nodeid:    text
```

Messages that the programmable operator facility sends as responses to the issuer of a programmable operator command or an asynchronous message to the CP operator originating at the programmable operator virtual machine have no such prefix. ***These responses are displayed in the language that is set for the programmable operator virtual machine. This language should always be American English.***

Stopping the Programmable Operator Facility

The programmable operator facility is easily stopped with the programmable operator STOP command. This command can be entered by a user logged on to the programmable operator facility virtual machine by typing STOP and pressing the ENTER key (or its equivalent). The programmable operator facility completes processing of all pending messages before stopping and returning control to CMS.

An alternative way to stop the programmable operator facility is for an authorized user to pass the STOP command to the programmable operator facility virtual machine from another virtual machine. The facility will stop processing as described above and return control to CMS. If the programmable operator facility virtual machine was running disconnected, it is left disconnected and the programmable operator facility is not active. To restart the programmable operator facility, someone must log on to the programmable operator facility virtual machine and invoke the facility as described in “Invoking the Programmable Operator Facility” on page 62.

PI

PROPEPIF EXEC

Before the programmable operator facility terminates completely, it executes an exec called PROPEPIF, which is located on the CMS System Disk (190). This exec can be copied to file mode A and tailored. PROPEPIF lets you restore any virtual machine settings that the programmable operator may have modified during operation. For example, when the programmable operator stops, it issues the command SET EMSG ON. If you want the final setting of EMSG to be something other than ON, simply put a command in the PROPEPIF EXEC, such as:

```
/* PROPEPIF Example */
"CP SET EMSG TEXT"
Exit
```

Parameters passed to the PROPEPIF EXEC are listed 104. Some do not have meaning to the PROPEPIF EXEC. These parameters are the requester user ID, requester node ID, and the message class.

PROPEPIF, along with PROPPROF, simply gives you more control over the changes that the programmable operator makes to a virtual machine while operating.

PI end

Programmable Operator Facility

Effect on an NCCF or NetView Logical Operator

If you use the STOP command to stop the programmable operator facility, the system sends a message to you (the NCCF or NetView logical operator). Someone must then logon to the programmable operator virtual machine and restart the programmable operator facility to continue operating in the mixed environment. If the programmable operator stops because of an abend and is able to recover, the programmable operator facility tries to re-establish the IUCV connection between the programmable operator and the PMX. In either case (STOP or ABEND), the programmable operator informs any assigned (using the LGLOPR ASN command) NCCF or NetView logical operator of its status (stopped or abended) by the PMX.

Running the Programmable Operator Facility from NCCF or NetView

To enhance the use of the programmable operator, it can be controlled by an NCCF or NetView operator, thus giving this operator control over mixed z/VM, OS/VS, and VSE distributed systems and host systems.

The Programmable Operator/NCCF Message Exchange

The Programmable Operator/NCCF Message Exchange (PMX) serves as a pipeline to transfer programmable operator commands from NCCF or NetView to the programmable operator virtual machine and to transfer routed messages and programmable operator responses to NCCF or NetView. The PMX executes in a GCS virtual machine with NCCF or NetView thus making GCS a requirement for programmable operator and NCCF or NetView communication. PMX uses IUCV to communicate with the programmable operator facility. The PMX also uses an NCCF or NetView command processor and the DSIMQS macro to communicate with NCCF or NetView logical operators. See the NCCF or NetView documentation for more information.

Installing the PMX

To use the Programmable Operator/NCCF or NetView connection, the user must:

1. Authorize the programmable operator virtual machine and the GCS virtual machine in which the PMX will execute to obtain IUCV connections with each other. The maximum number of IUCV connections allowed for the programmable operator must be at least three. For the PMX it must be at least two.

Make this authorization by adding IUCV statements for each of these users to the z/VM system directory. For more information on the z/VM system directory, see `cit bibid="hcpa5id">`.

2. Set up the GCS virtual machine in which the PMX (along with NCCF or NetView) will run as an authorized GCS virtual machine. See the NetView Program Directory for information about setting up the virtual machine. The PROPMX LOADLIB is located on the 190/490 disk and must be accessed prior to installation of PMX.
3. Specify in the GCS PROFILE (file name, file type, PROFILE GCS) that the PMX should start when GCS is IPLed. The format of the GCS command provided by NCCF for invoking the PMX is:

```
NCCF START PMX PARM userid
```

where *userid* is the user ID of the programmable operator virtual machine. Do this by modifying the exec NCCFSTRT GCS that is provided with NCCF.

The format of the GCS command provided by NetView for invoking the PMX is:

```
NETVIEW START PMX PARM userid (LOCAL)
```

where *userid* is the user ID of the programmable operator virtual machine. Do this by modifying the exec NETSTRT GCS that is provided with NetView.

If you do not specify this in the GCS PROFILE or the execs mentioned above, then the PMX must start manually after the programmable operator is started by issuing the same GCS command. When started, the PMX ATTACHes NCCF or NetView; that is, it runs as a subtask of the PMX in the same GCS virtual machine. When you specify the LOCAL option, the PMX attaches the hardware component of NetView which captures LOCAL device errors. See the NCCF and Netview documentation for more information.

- Use the VMFBLD command to build the PROPMX LOADLIB from which the PMX module is loaded by the NCCF or NetView command. The PROPMX LOADLIB must then be placed on a minidisk which is accessed by the virtual machine running PMX and NCCF or NetView. (An SFS directory cannot be used from a virtual machine in which GCS is running.) For example, it could be placed on the same minidisk as the NCCF LOADLIB. The PROPMX LOADLIB must also be GLOBALed (that is, GLOBAL LOADLIB PROPMX), along with the NCCF LOADLIB prior to invoking the NCCF or NetView command. This GLOBAL command can also be issued in the NCCFSTRT or NETSTRT GCS EXEC. The commentary in the exec describes how to do this. The format of the command is:

```
VMFBLD PPF ppfname GCS buildlst object (ALL
```

- Ensure that entries are added to the appropriate routing table files to authorize one or more NCCF or NetView operators to be assigned as the logical operator; that is, authorize them to enter the programmable operator LGLOPR command. For more uses of the LGLOPR command, see "Assigning or Changing the Logical Operator" on page 72.

Figure 3 would authorize any NCCF or NetView operator to enter the LGLOPR command. Restricted or specific authorization would be provided by putting the desired NCCF or NetView operator ID in the user field of the RTABLE entry.

```
*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
*  AUTHORIZE NCCF OR NETVIEW OPERATORS TO CHANGE LGLOPR ASSIGNMENT
*-----
/LGLOPR /           1   7 30           *NCCF   DMSPOR   LGLOPR
*-----
```

Figure 3. LGLOPR Command Authorization for an NCCF or NetView Operator

- A PROP command name must be supplied to NCCF or NetView through the CMDMDL statement in the file DSICMD NCCFLST, such as,

```
PROP      CMDMDL  MOD=GCTPNP
```

For more information on NCCF, NetView, and GCS, read the appropriate documentation for each product.

Communication between the Programmable Operator and NCCF or NetView

An NCCF or NetView operator (not necessarily the logical operator) and the programmable operator interact through both the NCCF or NetView PROP command and the Programmable Operator/NCCF Message Exchange (PMX). The PROP command is the NCCF or NetView operator's only means of sending data

Programmable Operator Facility

(commands) to the programmable operator. The PMX mediates all messages intended for NCCF or NetView logical operators.

An NCCF or NetView operator enters a programmable operator command by using PROP, an NCCF or NetView command, like this:

```
PROP CMD DETACH 00A FROM USER1
```

In a different domain, the operator must first establish an NCCF or NetView cross-domain session with the NCCF or NetView START command. Then, any programmable operator commands to be issued in that domain must be encapsulated in an NCCF or NetView ROUTE command. For example,

```
START DOMAIN=DOMN2
:
ROUTE DOMN2,PROP CMD SET LOGMSG 1 SHUTDOWN IS AT 7PM TONIGHT
```

See NCCF or NetView documentation for further information on domains and cross-domain sessions.

Stopping the PMX

If PMX stops or the NCCF or NetView operator session that is controlling the programmable operator stops, the NCCF or NetView logical operator (if any) is implicitly released. The programmable operator continues with the default logical operator specified on the LGLOPR statement of the active routing table.

The PMX may stop because:

- VM GCS stops
- NCCF or NetView stops (or the hardware monitor component of NetView, if the LOCAL option of the GCS NETVIEW command was used when starting the PMX)
- The PMX abends.

There are no PMX commands, per se; stopping NCCF or NetView normally stops the PMX.

When the PMX stops, NCCF or NetView does not necessarily stop—the PMX tries to wait until NCCF or NetView has stopped. However, you must stop NCCF or NetView before restarting the PMX.

The Logical Operator

The programmable operator facility is most commonly run in the CP system operator's virtual machine. It intercepts messages to the system operator, logs them in a CMS file and then performs additional actions on the messages, as defined by the active routing table. Sometimes, it is necessary for the programmable operator to send an intercepted message to another user for handling, because the programmable operator does not know or is not capable of performing the required action. (One example would be a message requesting that a tape be mounted. Since the programmable operator cannot perform physical tasks, such as mounting tapes, it simply sends the message to another user for handling.) For this reason, a user, other than the programmable operator virtual machine, is identified to the programmable operator to receive these messages. This user is called the logical operator, as opposed to the CP system operator.

When the programmable operator starts, the logical operator is sent an initiation message. While the programmable operator is active, the logical operator receives error messages for severe errors, such as logging errors, as well as all messages routed to the logical operator, either explicitly or by default. The logical operator also receives a message when the programmable operator is stopped.

The programmable operator facility can have only one logical operator at any given time. However, the role of the logical operator can be passed dynamically between several z/VM users and/or NCCF or NetView operators by means of the programmable operator LGLOPR command. Refer to “Assigning or Changing the Logical Operator” on page 72 and to “LGLOPR” on page 118 for more information.

The Default Logical Operator

Although the role of the logical operator can be passed from one z/VM or NCCF (or NetView) user to another, a single z/VM user must still be identified to the programmable operator. This user is called the DEFAULT logical operator and is identified to the programmable operator facility by means of the LGLOPR statement of the active routing table. The programmable operator facility begins routing messages to this DEFAULT logical operator when no other logical operator has been assigned or when the current logical operator issues the LGLOPR RLS command, implicitly or explicitly.

During programmable operator initialization, a message is sent to the logical operator indicating that the programmable operator facility is running. Normally (that is, unless the programmable operator is restarting after an abend situation), this logical operator is the DEFAULT logical operator specified in the active routing table, since the programmable operator is just starting up and no other users have been assigned yet. The DEFAULT logical operator will continue to be the current logical operator until an authorized z/VM or NCCF (or NetView) user enters the programmable operator LGLOPR command. Thus, if the active routing table does not authorize ANY users to enter the LGLOPR command, the DEFAULT logical operator will ALWAYS be the logical operator, until a new routing table is loaded.

Again, the DEFAULT logical operator MUST be a local or remote z/VM user; that is, the DEFAULT logical operator CANNOT be an NCCF or NetView operator.

Note: To avoid ambiguity, it should be noted that the term *logical operator* throughout this section is actually referring to the currently assigned logical operator, which MAY also be the DEFAULT logical operator. References to the *DEFAULT logical operator* are made when the information being discussed pertains ONLY to the DEFAULT logical operator.

The NCCF or NetView Logical Operator

To run a z/VM system from NCCF or NetView an NCCF or NetView operator must be assigned as the logical operator of the programmable operator in that particular system. This operator is the *NCCF logical operator* or the *NetView logical operator*. The routing table should specify which NCCF or NetView operators may be assigned as logical operators. The network installation through NCCF or NetView and VTAM defines where such NCCF or NetView operators may be logged on. This is independent of the programmable operator facility.

An NCCF or NetView operator may be authorized to be the logical operator by allowing that person to enter the programmable operator LGLOPR command. An NCCF or NetView operator can also be authorized to enter other programmable operator commands (QUERY, SET, CMD, and other programmable operator

Programmable Operator Facility

commands) without being authorized to use the LGLOPR command. That is, an NCCF or NetView operator can enter programmable operator commands without being authorized to be a logical operator. For example, assume the following record was in the active routing table.

```
*-----  
*T           S   E   T   U           N   A   P  
*E           C   C   Y   S           O   C   A  
*X           O   O   P   E           D   T   R  
*T           L   L   E   R           E   N   M  
*-----  
/QUERY /           1   6 30 NTWKOP1 *NCCF   DMSPOR   QUERY  
*-----
```

Figure 4. QUERY Command Authorization for an NCCF or NetView Operator

The entry above allows the NCCF or NetView operator, NTWKOP1, to enter the programmable operator QUERY command and receive the responses, even if he is not authorized to use the programmable operator LGLOPR command to assign himself as logical operator.

Assigning or Changing the Logical Operator

The LGLOPR command provides three options for assigning and releasing logical operators. These three options allow the role of logical operator to be passed back and forth between z/VM users and/or NCCF or NetView operators. For the format of this command, see "LGLOPR" on page 118. To authorize operators to use the LGLOPR command, place an entry for "/LGLOPR /" in the routing table to be used. Specify DMSPOR as the action routine, and LGLOPR as the parameter to DMSPOR.

Using the ASN (assign) and RLS (release) options is sufficient for many installations where perhaps there is only one z/VM operator or key operator (specified as the default logical operator) and one NCCF or NetView operator.

When more than one NCCF or NetView operator is capable of controlling the system or when a combination of multiple z/VM and NCCF or NetView logical operators are capable, the installation might want to allow some or all of the logical operators to use the RPL (replace) option. RPL forces the issuer of the command to be assigned as the logical operator, whether or not a logical operator is already assigned. This ensures that no messages are lost while changing logical operators because no messages are handled during the change.

For example, in establishment X, two NCCF or NetView operators on an MVS host system share responsibility for control of a set of z/VM distributed systems. The role of logical operator is passed back and forth between them depending on their individual workload. These operators pass control using the RPL option of the LGLOPR command so that no messages are routed to the default logical operator between releasing the old logical operator and assigning the new one.

The RPL option is also useful for forcing the assignment of a logical operator when the current logical operator is for some reason cut off from communication with the programmable operator and cannot do a RLS (release).

For example, a communications line connecting the terminal for an NCCF or NetView logical operator to the NCCF or NetView subsystem has gone down. The NCCF or NetView logical operator informs a local z/VM operator by telephone that the z/VM operator will have to take over as logical operator until the

communications line can be brought back up. The z/VM operator, if authorized, can enter a LGLOPR RPL command to take over the role of logical operator.

When the logical operator is re-assigned, the programmable operator tries the HOSTCHK function, if specified, following the re-assignment. However, if the new logical operator is an NCCF or NetView operator or a z/VM user on the same system as the programmable operator, the HOSTCHK function is suspended until a remote z/VM user is again assigned as the logical operator. For information on the HOSTCHK function, see "Communications Checking" on page 94 and the HOSTCHK statement in "Routing Table Configuration Entry Statements" on page 74.

If a user is assigned as the logical operator by the LGLOPR command, the assignment is accepted. The user specified on the LGLOPR statement remains a default logical operator in the event that the assigned logical operator is released. The user specified on the LGLOPR statement is always maintained as the default, and can never be released.

See Figure 5 for some sample uses of the LGLOPR command. In the samples, ACTIONX is an action routine that is executed each time an unauthorized user issues some form of the LGLOPR command. ACTIONX could be DMSPOR sending the offending command to the logical operator or it could be a user-written routine taking some other action against the issuer.

```

*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
* GENERAL AUTHORIZATION FOR LGLOPR COMMAND -- ASN, RLS, AND RPL
*-----
/LGLOPR /           1   7   NTKWOP1 *NCCF   DMSPOR   LGLOPR
*-----
* EXPLICIT AUTHORIZATION FOR LGLOPR REPLACE
*-----
/LGLOPR /RPL/           1   16   OPER2   HOSTSYS   DMSPOR   LGLOPR
/LGLOPR /RPL/           ACTIONX
*-----
* EXPLICIT AUTHORIZATION FOR LGLOPR ASSIGN
*-----
/LGLOPR /ASN/           1   16   NTKWOP4 *NCCF   DMSPOR   LGLOPR
/LGLOPR /ASN/           ACTIONX

```

An entry or entries could also be added as follows:

```

*-----
* AUTHORIZATION FOR LGLOPR ASSIGN AND RELEASE
*-----
/LGLOPR /~RPL/           1           NTKWOP5 *NCCF   DMSPOR   LGLOPR

```

Figure 5. Sample LGLOPR Command Entries in a Routing Table

The Routing Table

The **routing table** is a CMS file that contains the information used to control the operation of the programmable operator facility. It can reside on either a minidisk or SFS directory. The routing table enables the programmable operator facility to recognize a message as a command, to determine the action to take when a message comes in, and to recognize the authorized users of programmable operator functions.

If the routing table resides in an SFS directory, it can be shared with other users. You should, however, be aware of the implications of sharing files as described in the *z/VM: CMS User's Guide*. If, for example, someone has the routing table file exclusively locked when you start the programmable operator facility, programmable operator initialization fails because it cannot read the file.

Note: The routing table should always be coded in American English. Messages to the programmable operator should also always be in American English to ensure that the uppercasing and routing table comparisons are handled correctly.

How the Programmable Operator Facility Uses the Routing Table

When the programmable operator facility receives an IUCV interrupt with an incoming message, the active routing table is searched to find a matching entry. When the routing table is searched, all fields are checked. For a match to occur, each field must either match or be blank. If a matching entry is found, that entry contains information pertaining to any action to be taken. The action routine name tells the programmable operator facility which action routine to invoke when a routing table entry matches the incoming message. If no matching entry is found in the active routing table, no action is taken besides logging the message.

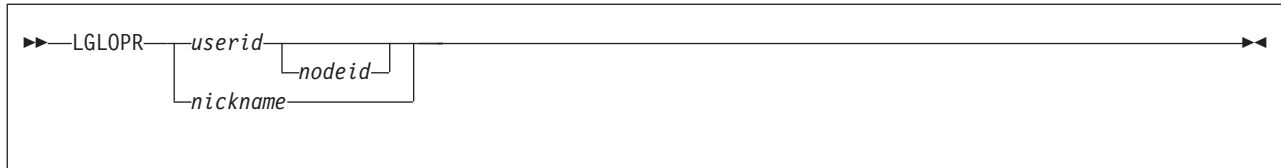
The order that the entries are placed in the routing table affects the way the programmable operator facility performs. The routing table is searched from top to bottom until a match is found. As the table is searched, lines that begin with an asterisk (*) in column 1 are ignored, and therefore may be used to place comments in the routing table. Also, lines that are completely blank are ignored in the routing table search and can be used to separate lines of text for easier reading. All entries must be made in upper case.

Routing Table Configuration Entry Statements

Every routing table must have specific configuration information in the first records of the routing table file (file type RTABLE) that are not comments or blank lines. These statements are in free format, meaning that they need not be positioned in any particular columns. See Figure 7 on page 82 for an example of a partial routing table. The statements and their parameters are as follows:

LGLOPR Statement

The **LGLOPR** statement identifies the default logical operator. The logical operator assignment can be changed using the LGLOPR command, but if a logical operator is released but not replaced the user specified in this statement resumes logical operator responsibilities. This user ID or nickname **must be** on a z/VM system.



userid is a valid user ID on the specified z/VM node.

nodeid is a valid ID of a z/VM system in the network. If no node ID is specified, the local system's node ID is used.

nickname

is a nickname defined in the programmable operator facility virtual machine CMS NAMES file.

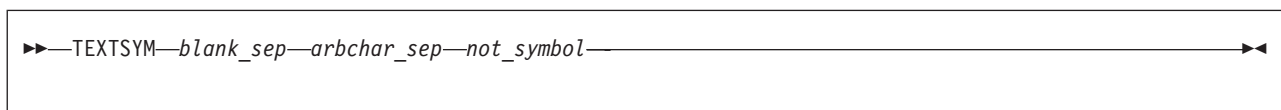
Note: If a nickname is used to identify the logical operator, the nickname entry must specify a single user ID. The nickname entry must also include node information when the logical operator is located on a different machine. Since PROP allows only 1 LGLOPR, the nickname entry cannot be another nickname or a list of nicknames or user IDs.

Either a nickname or a user ID must be specified. If a user ID is specified, a node ID may be specified. If both a user ID and a node ID are specified, they must be separated by one or more blanks. If the name specified is both a local user ID and a nickname, the programmable operator regards it as a nickname.

IMPORTANT NOTE: The programmable operator virtual machine should not be identified as the logical operator. This causes the programmable operator to go into a loop in the event it tries to do the routing. This includes specifying a user ID of OPERATOR (or any abbreviation thereof) in the LGLOPR statement. This also causes the message to be sent to the system operator virtual machine, even if the system operator virtual machine has a different user ID.

TEXTSYM Statement

The optional **TEXTSYM** statement specifies the characters that the programmable operator facility interprets as special symbols in the text field of the routing table entries. All three parameters must be specified if the statement is specified. If the statement is not specified, the default characters listed for each parameter are used.



blank_sep

is a separator character indicating that **blanks** are to be skipped over when scanning the message. A message is scanned for the next nonblank character string. This nonblank character string is then compared to the text in the routing table entry following this separator character. The default character is */*.

arbchar_sep

is a separator character indicating that **all nonmatching characters** are to be skipped over when scanning the message. A message is scanned for

Programmable Operator Facility

the text specified in the routing table entry until it is found or until the end of the message is reached. The default character is \$.

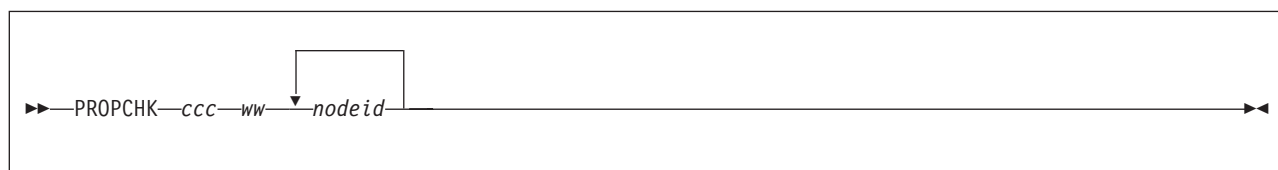
not_symbol

when it immediately follows a separator is the character indicating that the text should not be found in the message. If the text following the *not_symbol* is found in the message, then the message does not match that routing table entry. The default character is ~.

See “Filtering Messages” on page 86 and “Routing Text Comparison Examples” on page 83 for the use of TEXTSYM characters in routing table entries.

PROPCHK Statement

The optional **PROPCHK** statement identifies the distributed nodes that the host system is to check on. The RSCS node IDs of these distributed systems must be specified in this statement. A programmable operator must be running in the system operator virtual machine on the distributed systems being checked. Specify as many nodes on one statement as fit in an 80-column record. The programmable operator facility only reads the first 80 columns. Enter any number of PROPCHK statements to specify different checking or response wait intervals for different RSCS nodes. The PROPCHK statement must be after the LGLOPR statement.



ccc is the checking interval. This interval, in minutes, indicates how often acknowledgment requests are sent out to the specified nodes.

ww is the response wait interval. This interval is the number of minutes permitted to pass before a response must be received from the specified node(s).

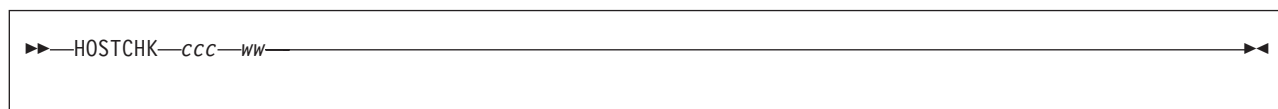
nodeid is a valid ID of a system in the network.

Notes:

1. The checking interval specified must be greater than or equal to the response wait interval.
2. The node ID of the logical operator must not be specified as a node ID on this statement.

HOSTCHK Statement

The optional **HOSTCHK** statement specifies the time interval for checking communication with the RSCS virtual machine at the logical operator node and the wait time for a response. The HOSTCHK statement must be after the LGLOPR statement.



ccc is the checking interval. This interval, in minutes, indicates how often acknowledgment requests are sent out to the logical operator's node.

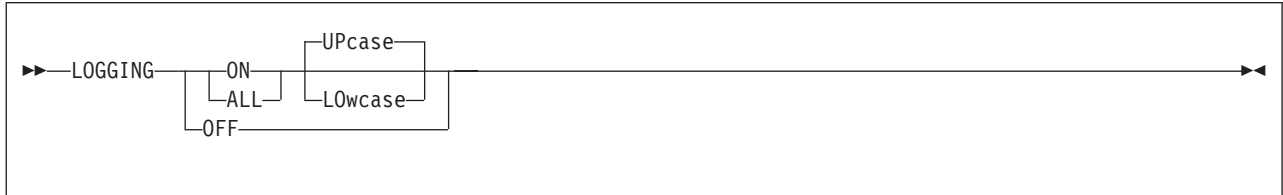
ww is the response wait interval. This interval is the number of minutes permitted to pass before a response must be received from the logical operator's node.

Notes:

1. The checking interval specified must be greater than or equal to the response wait interval.
2. The HOSTCHK function is suspended when an NCCF or NetView operator or a local z/VM user is assigned as the logical operator. It is resumed when a remote z/VM user is assigned as the logical operator.

LOGGING Statement

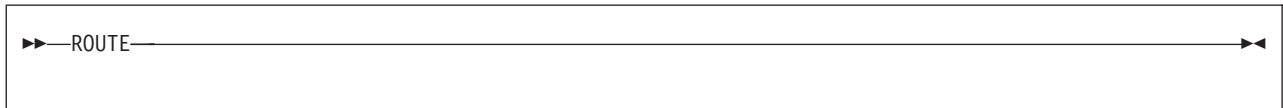
The optional **LOGGING** statement specifies whether messages or messages and command responses are to be logged or not logged. If the LOGGING statement is not in the routing table, messages are logged in uppercase format (LOGGING is ON and UPCASE). If the LOGGING statement is in the routing table, one of the three operands (ON, ALL, or OFF) must also be specified.



- ON** indicates that messages are to be logged while this RTABLE is active, unless it is explicitly turned off using the SET LOGGING command.
- ALL** indicates that messages and programmable operator command responses are to be logged while this RTABLE is active, unless it is explicitly turned off using the SET LOGGING command.
- UPcase** indicates that messages are to be logged in uppercase. This is the default.
- LOWcase** indicates that messages are to be logged with no change to the original message text.
- OFF** indicates that messages are not to be logged while this RTABLE is active, unless it is explicitly turned on using the SET LOGGING command.

ROUTE Statement

The **ROUTE** statement indicates the end of the configuration statements and the start of the routing entries.



The ROUTE statement **must** follow the other statements specified in this section.

MSGLIMIT Statement

The optional **MSGLIMIT** statement can be used to control the Message System Servive (*MSG) message limit.

Programmable Operator Facility

▶—MSGLIMIT—*nnnnn*—▶

nnnnn is the value to be used in establishing the message limit on the Message System Service IUCV path. Any value between 1 and 65535 may be used.

Notes:

1. See “How the Programmable Operator Establishes Communications with IUCV” on page 65 and the *z/VM: CP Programming Services* book for additional information on Message System Service (*MSG) and IUCV.
2. The path to *MSG is established on PROP initialization. As a result, control over the message limit is only available when RTABLE is used for initialization. LOADTBL cannot be used to establish a new MSGLIMIT value. A bad MSGLIMIT statement in a LOADTBL rtable will cause an error message to be issued and the new rtable will not load.

Routing Table Configuration Example

The LGLOPR and ROUTE statements are required in every routing table. The TEXTSYM, LOGGING, PROPCHK, and HOSTCHK statements are optional. An example of these statements in a routing table is as follows:

```
LGLOPR OPERATNS HOSTNODE
TEXTSYM / $ -
PROPCHK 5 1 NODE1 NODE2 NODE3
PROPCHK 3 1 NODE4 NODE5
HOSTCHK 2 1
LOGGING ALL
ROUTE
```

These special statements may be specified for each routing table in any order (as long as LGLOPR is first and ROUTE is last) and with at least one blank separating each parameter. The statements depend on the installation and, therefore, must be supplied by the installation for each routing table. These entries are processed only when the routing table is loaded, so they are not searched during programmable operator message handling.

The configuration shown in Figure 6 on page 79 can be described in a routing table with the first few lines like those in Figure 7 on page 82.

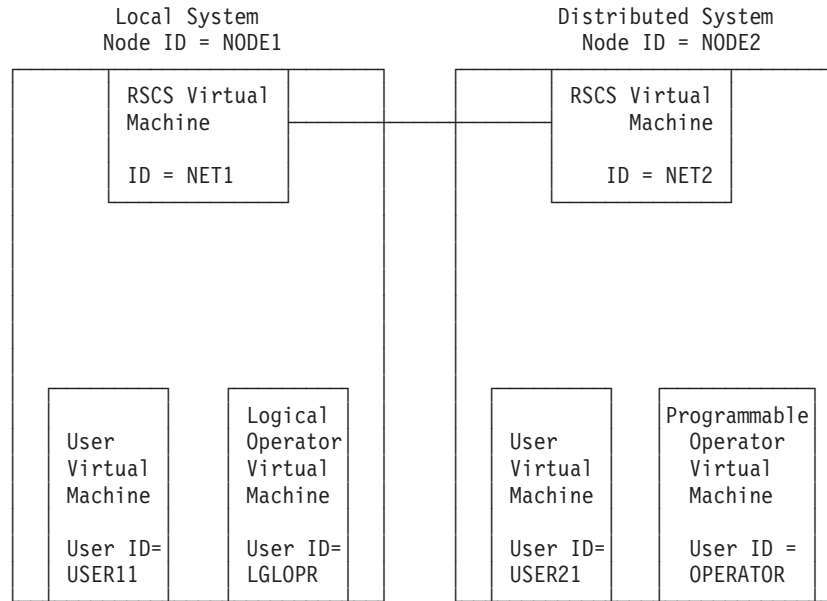


Figure 6. The Programmable Operator Facility in a Distributed System. The logical operator is situated at the host system and the programmable operator is running in a different system at a distributed site.

Routing Table Comparison Entry Statements

The routing table entries to be searched must be in the following fixed format. The contents of the fields described below may appear anywhere in the defined columns for that field (unaligned) unless the description for that field explicitly states that alignment is required.

Note: The words in parentheses correspond to the vertically aligned words in the comment records in Figure 7 through Figure 12, and also in the IBM sample routing table file.

FIELD	EXAMPLE	FIELD COLUMNS	LENGTH OF FIELD
COMPARISON TEXT (TEXT)	/FEEDBACK /	1-25	25
STARTING COLUMN (SCOL)	1	27-29	3
ENDING COLUMN (ECOL)	9	31-33	3
MESSAGE CLASS (TYPE)	1	35-36	2
USERID (USER)	USER21	38-45	8
NODEID (NODE)	NODE2	47-54	8
ACTION ROUTINE NAME (ACTN)	DMSPOR	56-63	8
PARAMETER TO ACTION ROUTINE (PARM)	TOFB	65-72	8

COMPARISON TEXT

is a particular character string that the programmable operator facility searches for in the incoming message. If this field is left blank, any text compared with this field is considered a match. Multiple texts may be specified, with the capability to skip over intervening blank or nonblank characters.

Programmable Operator Facility

STARTING COLUMN

is the column in the incoming message where the programmable operator facility starts looking for the character string mentioned in the COMPARISON TEXT field. If this field is left blank, the programmable operator facility starts scanning at the beginning of the message.

ENDING COLUMN

is the column in the incoming message where the programmable operator facility stops looking for the character string(s) mentioned in the COMPARISON TEXT field. If this field is left blank, the programmable operator facility continues scanning until the end of the message.

MESSAGE CLASS

identifies the origin of the incoming message according to the message type. Classes 1-9 are IUCV message types; class 30 is strictly a programmable operator facility message type for NCCF or NetView messages. For more details on IUCV message types, see the Message System Service section in the *z/VM: CP Programming Services*. If this field is left blank, any value compared with this field is considered a match.

The message types available to the programmable operator facility are:

Class Message Types

- | | |
|-----------|---|
| 1 | Message sent using CP MESSAGE and CP MSGNOH. |
| 2 | Message sent using CP WARNING. |
| 3 | Asynchronous CP messages, CP responses to a CP command executed by the programmable operator facility virtual machine, and any other console I/O initiated by CP. |
| 4 | Message sent using CP SMSG command. |
| 5 | Any data directed to the virtual console by the virtual machine (WRTERM, LINEDIT, and others). |
| 6 | Error messages from CP (EMSG). |
| 7 | Information messages from CP (IMSG). |
| 8 | Single Console Image Facility (SCIF) message from CP. |
| 30 | Message coming from the Network Communication Control Facility or NetView (*NCCF). |

Note: CP responses that are trapped in a buffer using the extended DIAGNOSE code X'08' do not become type-3 messages. For example, CP responses from the programmable operator CMD command are not type-3 messages, and therefore are not logged when LOGGING is set to ON.

USERID

is the character string compared to the user ID of the user that sent the incoming message to the programmable operator facility. It determines the authority of the user to cause an action to be performed. The identifiers for all NCCF or NetView operators, who may use the programmable operator, must be unique. If this field is left blank, all user IDs compared with this field are assumed to match. If this field is not left blank, the user ID must be left-justified in the field or you will receive an error message.

NODEID

is the character string compared to the node ID of the user that sent the incoming message to the programmable operator facility. Again, it determines the authority of the user to cause an action to be performed. To authorize an NCCF or NetView operator, *NCCF or blanks must be in the node ID field of the routing table. Also, the identifiers for all NCCF or NetView operators, who may use the programmable operator, must be

unique. If this field is left blank, all node IDs compared with this field are assumed to match. If this field is not left blank, the node ID must be left-justified in the field or you will receive an error message.

ACTION ROUTINE NAME

is the name of the LOADLIB member (a Basic Assembler Language routine) that the programmable operator facility is to NUCXLOAD when the LOADTBL function is performed, or the name of an exec, and subsequently, the routine that the programmable operator is to call when a match occurs on the entry in which the name is specified. If this field is left blank, no action is performed. If this field is not left blank, the action routine name must be left-justified in the field or you will receive an error message.

PARAMETER TO ACTION ROUTINE

is a character string of up to eight bytes passed as a parameter to the action routine by way of the programmable operator PARMLIST for a Basic Assembler Language routine or by way of a program stack for an exec. Often, this is used to specify a particular subroutine in the action routine. If this field is left blank, no parameter is passed. If this field is not left blank, the action routine parameter must be left-justified in the field or you will receive an error message.

Column 73 and beyond are reserved for future use.

Figure 7 on page 82 illustrates an example of a partial routing table.

Programmable Operator Facility

```

* THIS IS THE DEFINITION OF THE PROP CONFIGURATION.
* LOGICAL OPERATOR IS NICKNAME "LOP". SEE "OPERATOR NAMES" FILE.
* LOGICAL OPERATOR (NICKNAME "LOP") IS "OPERATOR" AT NODEID "NODE1".
LGLOPR LOP
* THE TEXT SEPARATOR CHARACTERS.
TEXTSYM / $ ~
* ESTABLISH 1,024 AS THE *MSG VALUE.
MSGLIMIT 1024
* WHICH NODES TO CHECK, AND AT WHAT INTERVAL.
HOSTCHK 5 1
* THE ROUTING ENTRIES START
ROUTE
*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
* SEND PROP FEEDBACK COMMAND TO FEEDBACK ACTION ROUTINE
*-----
/FEEDBACK /           1   9   1 USER21   NODE2   DMSPOR   TOFB
*-----
* AUTHORIZE NCCF OR NETVIEW OPERATORS TO CHANGE LGLOPR
* WITH THE LGLOPR COMMAND
*-----
/LGLOPR /           1   7 30           DMSPOR   LGLOPR
*-----
* FILTER OUT LOGON AND LOGOFF MESSAGES SO OPERATOR NEED NOT SEE THEM
* BUT LET "FORCED" LOGOFF MESSAGES THROUGH
*-----
/LOGON           21  26  3
/LOGOFF$~FORCED  21  80  3
*-----
* FILTER OUT COMMANDS THAT WE DO NOT WANT ISSUED.
*-----
/CMD /SYSTEM           WARNING
/CMD /SET /EC           WARNING
*-----
* ALLOW ONLY OPERATOR ON HOST TO ISSUE SHUTDOWN.
*-----
/CMD /SHUTDOWN           OPERATOR NODE1   DMSPOR   TOVM
*-----
* ROUTE ALL MESSAGES ABOUT DEVICE 00E TO THE SPOOL OPERATOR.
*-----
$ 00E           3           DMSPOS   SPOOLOP

```

WARNING

represents a user action routine which may send a warning message to the user issuing that command.

SPOOLOP

represents a nickname or user ID of the spool operator.

Figure 7. Partial Routing Table

Tailoring the Routing Table

Routing table entries determine what messages the programmable operator facility ignores (filtering), who is authorized to enter a particular command (authorization), and what action routines to invoke for a given circumstance. You can tailor the routing table to suit your system's individual needs by adding or changing entries in the routing table.

The programmable operator facility comes with a general purpose routing table named PROP RTABLE. (See “Installing the Programmable Operator Facility” on page 62 to locate the PROP RTABLE.) You can use this supplied routing table only after the LGLOPR, HOSTCHK, and PROPCHK statements are modified. You may also have to modify the routing table entries. Use XEDIT to make these changes. The programmable operator facility operates satisfactorily with no further changes. However, if you choose, you can modify the supplied routing table to change the operation of the programmable operator facility. You can also create different routing tables to cover varying circumstances. These tables can be dynamically loaded using the LOADTBL command. Only one routing table may be active at a time.

Routing Text Comparison Examples

Here are more examples of text comparisons for the programmable operator facility.

\$ is the arbitrary character separator.
/ is the blank separator.
¬ is the not_symbol.

These characters are specified by the TEXTSYM statement, which is described in “TEXTSYM Statement” on page 75. In all of these examples, it is assumed that the starting and ending columns do not interfere with the matching.

RTABLE Examples:

- The RTABLE entry

```
$LOGOFF
```

is matched by any message containing the word LOGOFF. If one text is preceded by the arbitrary character separator (\$), the text can appear anywhere in the message to be a match.

- The RTABLE entry

```
/LOGOFF
```

matches the message

```
LOGOFF USER1 IN 5 minutes
```

as there are no nonblank characters preceding the word LOGOFF, but the entry does not match the message

```
11:20:15 GRAF 0A0 LOGOFF AS USER1 USERS = 020
```

If only one text is preceded by the blank character separator (/), the text must be the first nonblank string in the message order to be a match.

- The RTABLE entry

```
$AUTO$LOGON$AUTOLOG
```

matches the message

```
11:09:02 AUTO LOGON *** USER2 USERS = 021 BY AUTOLOG1
```

and the message

```
11:09:02 AUTO LOGON *** AUTOLOG2 USERS = 021 BY SYSTEM
```

but not the message

```
11:09:02 GRAF 0A0 LOGON AS AUTOLOG1 USERS = 023
```

or the message

```
AUTOLOG WILL NOT LOGON TOMORROW
```

Programmable Operator Facility

A text with two or more texts preceded by arbitrary character separators (\$), is matched by a message with all those texts appearing in that order.

The texts in the message are scanned in the order that they appear in the routing table entry. One text is searched for at a time. If the arbitrary character separator (\$) precedes the text in the entry, a message is scanned until a match is found or the end of the message is reached. If the blank character separator (/) precedes the comparison text, blanks are skipped over and the first nonblank string of characters is compared to the comparison text, which may or may not match.

Routing table entries and messages are also affected if the text is preceded by a not_symbol (^). The not_symbol is always used with one of the other separator characters; it never stands alone. If matching text is found and the text in the routing table is preceded only by a \$ or a /, the position following the last matched text is remembered. If there are no more RTABLE texts to be searched for, the entry is a match. If there is another text in that RTABLE entry to be searched for, the scan continues from the position following the last matched text.

A match depends on the rest of the message text and the routing table entry. If matching text *is* found but the text in the routing table *is* preceded by the not_symbol (^), the entry *is not* a match and checking goes no further. Similarly, if a matching text *is not* found but the text in the routing table *is not* preceded by the not_symbol, the entry *is not* a match. If a match *is not* found and the text *is* preceded by the not_symbol (^), and if there is no more text, the entry matches the message. If there is more text to scan for, the scan continues as above starting with the character following the last match. A match depends on the rest of the message text and the routing table entry.

- The RTABLE entry

```
$-AUTO$LOGON
```

does match the message

```
12:04:28 GRAF 0A0 LOGON AS USER1      USERS = 027
```

Because the first text in the RTABLE entry (AUTO) is preceded by the arbitrary character separator (\$), the entire text is searched for AUTO. No match is found. Because the text is preceded by the not_symbol (^), the text is still a match at this point. The scan for the next text (LOGON) begins at the end of the last match. Because there was no previous match, the scan begins again at the start of the message. The LOGON text is preceded by the arbitrary character separator (\$), so the search proceeds through the message until LOGON is matched. Because LOGON appears in the message, this RTABLE entry and message *do* match.

- The RTABLE entry

```
$-AUTO/LOGON
```

does not match the same message

```
12:04:28 GRAF 0A0 LOGON AS USER1      USERS = 027
```

The message is scanned for AUTO as above. The search for LOGON again begins at the beginning of the message. In this case, however, the LOGON text is preceded by the blank separator (/), so only blanks are skipped prior to the comparison. No blanks are found, so the comparison is made at the beginning of the text and LOGON is compared with 12:04:.. This is not a match. Because this text was not preceded by the not_symbol, this RTABLE entry and message *do not* match.

- The RTABLE entry

```
$~AUTO/LOGON
```

does not match the message

```
12:04:28 AUTO LOGON *** USER1    USERS = 027 BY AUTOLOG1
```

Because AUTO is found in the message and is preceded in the RTABLE entry by the arbitrary character separator (\$) and the not_symbol (~), the RTABLE entry and message *do not* match.

- The RTABLE entry

```
$LOGOFF$~030/FORCED
```

does not match the message

```
12:04:28 USER DSC LOGOFF AS USER1    USERS = 026 FORCED
```

The first text, LOGOFF, is preceded by the arbitrary character separator (\$) and is scanned for through the text. LOGOFF is found. Because LOGOFF is not preceded by the not_symbol, the next text is scanned. The scan continues from the end of the previous match, which is the character following the LOGOFF text. Since the arbitrary character separator (\$), precedes 030, the entire remaining text is searched for 030. It is not found but because 030 is preceded by the not_symbol, the message and RTABLE entry still match. Finally, FORCED is scanned for. It is preceded by the blank separator (/). Blanks are skipped, and starting with the character following the last matched string (which was LOGOFF), FORCED is compared to AS USE. This is not a match. Because FORCED is not matched and is not preceded by the not_symbol, this RTABLE entry and message *do not* match.

Here are routing table entries that *do* match this message:

```
$LOGOFF$~030$FORCED
```

would match because arbitrary characters would be skipped before comparison for FORCED and

```
$LOGOFF$~030/~FORCED
```

would match because the first nonblank string after LOGOFF is not FORCED.

Routing Messages to an NCCF or NetView Operator

To route a message to an NCCF or NetView logical operator, that NCCF or NetView operator must have entered a LGLOPR ASN or LGLOPR RPL command. Be sure that the appropriate NCCF or NetView operators are authorized in the active routing table to enter the command.

For example, to route the message

```
12:04:15 GRAF 0A0 LOGOFF AS USER1    USERS = 020  FORCED
```

to the NCCF or NetView logical operator, the routing table should have the following entries, as shown in Figure 8.

Programmable Operator Facility

```

:
*-----*
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----*
* AUTHORIZE OPERATORS TO CHANGE LGLOPR ASSIGNMENT
*-----*
/LGLOPR /           1   7 30           DMSPOR   LGLOPR
*-----*
* FILTER OUT SPECIFIC MESSAGES
*-----*
$GRAF$LOGOFF AS$FORCED$           3           DMSPOS   LGLOPR
*-----*
:

```

Figure 8. Routing Entries to Send Messages to an NCCF or NetView Operator

DMSPOR LGLOPR specifies that the programmable operator LGLOPR command can be executed. Message type 30 says that the user executing the LGLOPR command must be an NCCF or NetView operator. DMSPOS LGLOPR specifies that the desired action is to send the matching message to the logical operator. First, the NCCF or NetView logical operator must have entered LGLOPR ASN or LGLOPR RPL command to the programmable operator facility. Then, when the above message arrives at the programmable operator virtual machine, it is routed to the assigned NCCF or NetView logical operator.

If you wish to put an ID in the node ID field of the routing table entries for the NCCF or NetView operator, it must be *NCCF.

Filtering Messages

This is the simplest application of the programmable operator facility. Entries can be placed in the routing table to filter informational messages. The messages are filtered because no action routine is specified in the routing table entries. For example, when the programmable operator facility is running in the system operator's virtual machine, informational messages resulting from commands such as, LOGON, LOGOFF, and DISCONN, can be prevented from being displayed at the logical operator's console. Although the messages are not displayed at the operator's console, they can be logged in the current day's log file. The routing table entries must identify the texts in the message that makes it unique and identify the columns between which the texts should be found in the message. With single or multiple texts, TEXTSYM characters should be selected accordingly. Figure 9 on page 87 shows an example of how entries may be placed in the routing table to filter unwanted responses directed to the logical operator. For example, the message

```
12:04:50 GRAF 055 LOGON AS USER1
```

would match the second routing table entry (/~AUTO\$LOGON). This RTABLE specification means that, starting in column 9 (SCOL) of the message, AUTO cannot be the first nonblank string and that LOGON must appear somewhere in the message. The message would be filtered out but logged in the current day's log file.

However, the message

```
12:04:28 AUTO LOGON *** USER BY AUTOLOG1
```

would not match the second routing table entry (/~AUTO\$LOGON) because AUTO is the first nonblank string in the message appearing in the columns between the

SCOL and ECOL fields. Thus, the message would not be filtered out and would be routed to the logical operator, as specified in the last entry in Figure 9.

```

* THIS IS THE DEFINITION OF THE PROP CONFIGURATION.
* LOGICAL OPERATOR IS NICKNAME "LOP".  SEE "OPERATOR NAMES" FILE.
LGLOPR  LOP
* THE TEXT SEPARATOR CHARACTERS.
TEXTSYM / $ -
* WHICH NODES TO CHECK, AND AT WHAT INTERVAL.
PROPCHK 5 1  NODE2A  NODE2B
PROPCHK 2 1  NODE1A  NODE1B
* THE ROUTING ENTRIES START
ROUTE
*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
* FILTER OUT LOGON AND LOGOFF MESSAGES SO OPERATOR NEED NOT SEE THEM
*-----
$OUTPUT/OF           19 36 3
/~AUTO$LOGON         9 33 3
$LOGOFF              19 34 3
$DISCONNECT          19 36 3
$RECONNECT           19 36 3
$DIAL                19 32 3
$DROP                19 32 3
*-----
* SEND REMAINING ASYNCHRONOUS CP MESSAGES TO LOGICAL OPERATOR
*-----
                                           3           DMSPOS  LGLOPR

```

Figure 9. Routing Entries to Filter Responses to Routine Commands

In Figure 9, the entries that appear in the TEXT field (OUTPUT OF, LOGON, and other entries) are the texts contained in the messages that are to be trapped by the programmable operator facility when they are issued by CP.

No user IDs and node IDs are specified for these entries because they are issued by CP. Because no action routine is specified, the only action taken is the logging of the messages in the current day's log file.

Looking at the last line in Figure 9, you can see that if a type-3 IUCV message is received that does not have a corresponding entry in the routing table, action routine DMSPOS together with the LGLOPR parameter routes the message to the logical operator. In this case, this entry has to be placed after the specific text entries that you want filtered from the message stream. If this entry appeared before the text entries in Figure 9, all type-3 IUCV messages would be routed to the logical operator.

Controlling Authorization

The routing table determines who is authorized to enter specific commands in the programmable operator facility. Programmable operator authorization is based entirely on the contents of the routing table. Therefore, controlling authorization is a relatively simple procedure. Authorization checking uses either the user ID, node ID, the command text, or any combination of these fields in a routing table entry. A change to any of these fields can result in a change in authorization. You can easily tailor the authorization structure to your particular needs by changing only these fields in the routing table entries, without changing the action routines.

Programmable Operator Facility

When a user ID and node ID *are not* specified for a routing table entry, all users are authorized to match that entry and to use the function that it describes. Figure 10 shows an example of unrestricted authorization for the FEEDBACK command. A message sent with the FEEDBACK command is passed to module DMSPOR, which supports most of the programmable operator commands. The TOFB parameter invokes the proper action routine contained in module DMSPOR that writes the message to the FEEDBACK file. See “The Feedback File” on page 93 or “FEEDBACK” on page 116 for more information.

Note: In the following examples, the TYPE (message class) field is left blank to allow the FEEDBACK (or FB) command to be entered with any class of IUCV message. The ECOL fields are 9 and 3 because the character string being looked for is FEEDBACK or FB followed by a blank, for example, FEEDBACK or FB would match.

```

*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
* PLACE A FEEDBACK MESSAGE IN THE PROP FEEDBACK FILE
*-----
/FEEDBACK /           1   9           DMSPOR TOFB
/FB /           1   3           DMSPOR TOFB
:               :   :               :   :
:               :   :               :   :

```

Figure 10. Uncontrolled Authorization

Authorization can be restricted to users at a particular network node by specifying only the node ID. In Figure 11, only users at NODE1 are authorized to enter the FEEDBACK command.

```

*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
* PLACE A FEEDBACK MESSAGE IN THE PROP FEEDBACK FILE
*-----
/FEEDBACK /           1   9           NODE1  DMSPOR TOFB
/FB /           1   3           NODE1  DMSPOR TOFB
:               :   :               :   :
:               :   :               :   :

```

Figure 11. Restricting Authorization by Node ID

When a user ID and node ID are specified, only that user at the specified node is authorized to match that entry. In Figure 12 on page 89, only JOHNDOE at NODE1 and JANEDOE at NODE2 are authorized to place messages in the feedback file.

```

*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
* PLACE A FEEDBACK MESSAGE IN THE PROP FEEDBACK FILE
*-----
/FEEDBACK /           1   9   JOHNDOE  NODE1   DMSPOR  TOFB
/FEEDBACK /           1   9   JANEDOE  NODE2   DMSPOR  TOFB
/FB /           1   3   JOHNDOE  NODE1   DMSPOR  TOFB
/FB /           1   3   JANEDOE  NODE2   DMSPOR  TOFB
:           :   :           :           :           :
:           :   :           :           :           :
*-----
* SEND REMAINING REQUESTS AND COMMANDS TO THE LOGICAL OPERATOR
*-----
                                           DMSPOS  LGLOPR

```

Figure 12. Restricting Authorization by User ID and Node ID

Since the user must explicitly enter the FEEDBACK or FB command to have a message placed in the feedback file, action routine DMSPOR TOFB must be specified in the routing table to carry out the required action. Any user trying to enter the FEEDBACK command that is not authorized by the routing table in Figure 12 will have their command sent to the logical operator as a message by action routine DMSPOS with parameter LGLOPR, as specified by the last record of the routing table.

Additional user IDs and node IDs may be added to the table to grant authorization to enter these commands. Conversely, user IDs and node IDs may be removed to revoke authorization.

Restricting Command Use

You can easily restrict command use to a specific group of users. You can specify first entries for those users who should be allowed to use the command, then specify an entry to trap the use of that command by any other user. Finally have an entry for any users who should have general command usage except for the restriction. The following example demonstrates command restriction.

```

*-----
*T           S   E   T   U           N   A   P
*E           C   C   Y   S           O   C   A
*X           O   O   P   E           D   T   R
*T           L   L   E   R           E   N   M
*-----
/CMD /SHUTDOWN /           1  12   LGLOPR  HOSTNODE  DMSPOR  TOVM
/CMD /SHUTDOWN /           ACTIONX
/CMD /NETWORK /SHUTDOWN /  1  25   NETOP1  HOSTNODE  DMSPOR  TOVM
/CMD /NETWORK /SHUTDOWN /           ACTIONX
/CMD /NETWORK /           1  20   NETOP1  HOSTNODE  DMSPOR  TOVM
/CMD /NETWORK /           1  20   NETOP   RMTNODE  DMSPOR  TOVM
/CMD /NETWORK /           ACTIONX
/CMD /           1   4   LGLOPR  HOSTNODE  DMSPOR  TOVM
/CMD /           1   4   MAINT   HOSTNODE  DMSPOR  TOVM
/CMD /           1   4   MAINT   PROPNODE  DMSPOR  TOVM
*-----

```

Figure 13. Restricting Command Use to Specific Users

An RTABLE containing these entries allows LGLOPR at HOSTNODE to use the SHUTDOWN command, but anyone else trying to use it invokes ACTIONX. ACTIONX could be DMSPOS sending the offending command to the logical

Programmable Operator Facility

operator or it could be a user-written action routine taking some other action against the issuer, depending of the severity of the offense. The same description applies to the NETWORK SHUTDOWN and NETWORK commands. So, routing table coding provides considerable flexibility in granting or restricting command authority.

The Log File

PI

If LOGGING is set to ON or ALL, every incoming message that the programmable operator facility receives is put into a CMS file referred to as the *log file*. The log file can reside on either a minidisk or an SFS directory. If LOGGING is set to ALL, all error messages and command responses generated by the programmable operator facility are also put in the log file. If LOGGING is set ON, responses from CP, CMS, and programmable operator facility commands are not logged; messages are.

When the log file resides in an SFS directory, it can be shared with other users. You should, however, ensure that no one will be writing to the file and that no one has the file locked when the programmable operator is running. If other users have write authority to the log file, consider locking the file before starting the programmable operator. See the CREATE LOCK command in the *z/VM: CMS Commands and Utilities Reference*.

Each message is identified by the date and time received. The user ID and node ID appear only if the text was sent by a CP MSG, SMSG, WNG, or sent using SCIF (Single Console Image Facility). The user ID and node ID are blank for a message sent by CP. A message sent by a remote RSCS network virtual machine has a node ID, but no user ID. A message sent from an NCCF or NetView operator console has *NCCF as the node ID.

Log entries generated and logged by the programmable operator have a user ID of PROP. The log file has the following format:

```
col 1      col 10     col 19     col 28     col 39
|          |          |          |          |
yy/mm/dd hh:mm:ss{userid  nodeid}:  text
```

The log file contains variable length records. The maximum record length that the programmable operator facility can place in the log file is 132 characters. Because the prefix uses 38 of the 132 characters, the text can be only 94 characters long. Therefore if the text of a message exceeds the maximum length of 94 characters the overflow is continued on the next record. This continued record has the same prefix as the preceding record, with no colon (:) preceding the text in column 37.

A separate log file is started for each day. The name of the file is:

```
LGyymmdd nodeid A5
```

yy is the current year

mm is the current month

dd is the current day

nodeid is the current RSCS node ID of the system on which the programmable operator facility is running. (If the node ID is not available, then the file type for the logfile will default to PROPROFILE.)

When the programmable operator facility is started, stopped, or debug mode is changed, a record is written to the log file. The messages written to the file have

the normal log prefix and a text corresponding to the changed function. Generally, responses to the programmable operator *console* commands are written to the log file when LOGGING is set to ON or ALL. It is also possible to have responses from the programmable operator commands written to the log file. See the LOGGING statement of the routing table or the programmable operator SET command for more information. The log file may be used as an alternative to spooling the virtual console when LOGGING is set to ALL and LOWCASE. When node-checking is in effect, by having PROPCHK or HOSTCHK statements in the RTABLE, if a node changes status from UP to DOWN or from DOWN to UP, a message is also written to the log file.

If a virtual machine resource limit is reached, such as *disk full* or *file space full*, it may not be possible to write another record to the programmable operator facility log file. If this happens, the programmable operator facility invokes the PROPLGER EXEC for recovery. IBM supplies a sample PROPLGER EXEC. See “Log Error Exit (PROPLGER)” on page 109. You can use the sample PROPLGER EXEC, modify it, or replace it with a user-written exec.

When using an SFS directory accessed as file mode A, PROPLGER EXEC is invoked whenever the file space threshold limit is exceeded. However, if the threshold is set too high or other users are writing to this storage group, the programmable operator facility may not detect certain resource limits until it tries to close the log file. At that point the programmable operator will not be able to invoke the PROPLGER EXEC, and some log data could be lost. If it is important for you to minimize the potential loss of log data, consider using a minidisk for the log file instead of an SFS directory.

Any z/VM user authorized in the active routing table³ can obtain the log file as a reader spool file by using the programmable operator GET LOG command (this does not include NCCF or NetView users). Messages can be placed in the log file by authorized users by using the programmable operator LOG command with no other action being taken.

An old log file can be purged by any user authorized in the active routing table to use the programmable operator CMD command by issuing the CMS ERASE command.

Logging NCCF or NetView Messages in the Log File

A message from an NCCF or NetView operator is logged just as any other message sent to the programmable operator. When logging a message or command from an NCCF or NetView operator, the keyword *NCCF is placed in the node ID field of the log record. You must ensure that the identifiers for all of the NCCF or NetView operators who may access the programmable operator facility are unique. This is also an NCCF or NetView requirement. As a result, the operator identifier with the *NCCF node ID is sufficient to identify the issuer of the programmable operator command. Also, messages received by the programmable operator facility from NCCF or NetView have the message type of 30.

Ensuring a Complete Log

When the programmable operator facility routes a message to the logical operator, the message contains the user ID of the sender. The operator, in responding to the message, may choose to send a message directly to the user without going through

3. An NCCF or NetView operator cannot use the GET command to obtain the log file. Use CMS commands (and the programmable operator CMD command) to type the file or portions of the file or to send the file to a user ID where you can process it.

Programmable Operator Facility

the programmable operator facility. However, if this is done, the message is not logged in the log file. To ensure that these messages are logged, the operator should send the message to the user through the programmable operator facility by using the programmable operator facility CMD command. For information about the programmable operator facility CMD command, see page 114.

Whether the message was sent through the programmable operator or not has little significance to the user. However, so that the messages received by the user always have the same ID (the programmable operator facility ID), the message should always be sent from the logical operator through the programmable operator facility.

In a Single System: To route a message through the programmable operator facility where the operator and user are on the same physical system, use the MSG command:

MSG operator CMD MSG user1 - RESPONDING TO YOUR REQUEST

operator

is the user ID of the programmable operator facility virtual machine.

CMD

is the programmable operator facility CMD command.

MSG

is the z/VM command that the programmable operator facility will execute.

user1

is the user ID of the user who will receive the message.

RESPONDING TO YOUR REQUEST

is the message text sent to the user.

In z/VM Distributed Systems: To route a message through the programmable operator facility where the operator and user are not on the same physical z/VM system, use the SMSG command:

SMSG net1 MSG node1 operator CMD MSG user1 - RESPONDING TO YOUR REQUEST

net1

is the user ID of the network machine at the user's node.

MSG

(first appearance) is the RSCS message command.

node1

is the node ID of the programmable operator facility virtual machine.

operator

is the user ID of the programmable operator facility virtual machine.

CMD

is the programmable operator facility CMD command.

MSG

(second appearance) is the z/VM command that the programmable operator facility executes.

user1

is the user ID of the user to receive the message.

RESPONDING TO YOUR REQUEST

is the message text sent to the user.

In a Mixed Environment: To route a message through the programmable operator facility from the logical operator who is an NCCF or NetView operator to the user on a z/VM system, use PROP, an NCCF and NetView command as follows:

PROP CMD MSG *user1* - RESPONDING TO YOUR REQUEST

PROP

is the NCCF or NetView command that sends the message from an NCCF or NetView operator to the programmable operator facility.

CMD

is the programmable operator CMD command.

MSG

is the z/VM command that the programmable operator facility will execute.

user1

is the user ID of the z/VM user who will receive the message.

RESPONDING TO YOUR REQUEST

is the message text sent to the user.

Note: See “Helpful Hints” on page 100 for ways to reduce typing of long text strings.

The Feedback File

The feedback file is another CMS file that the programmable operator facility manages. The file is named FEEDBACK *nodeid* A5 (*nodeid* will default to PROPFIL if a node ID is not available). The feedback file, like the log file, can reside on either a minidisk or SFS directory. If the feedback file resides in an SFS directory, it can be shared, but no other user should write to the file, or have the file locked while the programmable operator is running. The feedback file, unlike the log file, is not automatically written by the programmable operator facility. Authorized users can write time stamped notes and complaints about the operation of the system to this feedback file. To write a notice to the feedback file, you, as a user, must explicitly use the FEEDBACK (or FB) command. An example of such a message is:

```
m op feedback response time was slow during morning shift.
```

Because the feedback file is normally smaller than the log file, it is easier for the personnel in charge of the programmable operator facility’s maintenance to review the users’ comments and identify when and where particular problems occurred.

Each record in the feedback file is prefixed with the date and time the message was logged along with the sender’s user ID and node ID. The feedback file has the following format:

```
col 1      col 10     col 19     col 28     col 39
|         |         |         |         |
yy/mm/dd hh:mm:ss userid  nodeid:  text
```

The feedback file contains variable length records. The maximum record length that the programmable operator facility can place in the feedback file is 132 characters. Because the prefix uses 38 of the 132 characters, the text can be only 94 characters long. Therefore if the text of a message exceeds the maximum length of

Programmable Operator Facility

94 characters the overflow is continued on the next record. This continued record has the same prefix as the preceding record, with no colon preceding the text.

Any user authorized in the active routing table, except an NCCF or NetView operator⁴, can obtain the feedback file as a reader spool file by using the programmable operator GET FEEDBACK or GET FB command.

An old feedback file can be purged by any user authorized in the active routing table to use the programmable operator CMD command by issuing the CMS ERASE command.

PI end

Communications Checking

In a z/VM-only environment, the programmable operator facility can operate either from the host system or from a distributed system in a network or from both sides. Special functions can be performed depending on the ability of the programmable operators to communicate through RSCS Networking. The purpose of these functions is:

- To provide the host operator (logical operator) with timely information and/or action in the event of a break in communication with the programmable operator on one of the network's distributed systems.
- To provide a distributed system with timely information and/or capability for action in the event of a break in communication with the host system.

A programmable operator can periodically check on the link with another system to determine whether it is possible to communicate with that system. The systems to be checked must be identified in the routing table of the programmable operator doing the checking. This may be either the programmable operator at the host system checking on specified distributed systems or a distributed programmable operator checking on communications with its host system. When the programmable operator at the host system is checking on the distributed systems, the programmable operator needs another programmable operator running in the system operator virtual machine on the distributed system. This is not required when a distributed system is checking on the host system. In other words, for *host checking*, no programmable operator is required at the host system, but for *distributed system checking* programmable operators must be running at the distributed systems. These various types of checking may be collectively referred to as *node-checking*.

Note that the roles of the host and distributed systems need not be strictly defined. For example, a programmable operator may use the PROPCHK function to check communication with *any* other system (node) running a programmable operator in its system operator virtual machine in the network. With the HOSTCHK function, the system being checked is simply the system defined as the checking system's logical operator, and not necessarily **the** host system for the network.

The programmable operator facility that has been instructed to check on its distributed system(s) periodically tries to communicate with those systems by sending a message that causes a response. The programmable operator then waits

4. An NCCF or NetView operator cannot use the GET command to obtain the feedback file. Use CMS commands (and the programmable operator CMD command) to type the file or portions of the file or to send the file to a user ID where you can process it.

a specified time for a response. For checking the host system (HOSTCHK), the acknowledgement request goes to the RSCS on the logical operator node. For checking the distributed systems (PROPCHK), it goes to the programmable operator on the distributed system. No response indicates that something has prevented communication between the host and the distributed system(s). Getting a response after being delinquent for a time indicates that communication between the programmable operators has been restored. With the SET command, the user is able to set the checking function ON or OFF.

Any time that the programmable operator detects that a node has exceeded the time allowed for responding, that fact is recorded in the programmable operator log. Also logged is the fact that a node has resumed responding.

When one of these conditions (no response or a late response) is detected, the programmable operator doing the checking invokes a communication error exit routine.

For example, if a programmable operator on a distributed system has a HOSTCHK statement in its routing table, the programmable operator would invoke the PROPHCHK EXEC if communication with the host system were lost long enough to prevent the checking request or response from getting through. Similarly, if a programmable operator on a host system has a PROPCHK statement in its routing table, that programmable operator would invoke the PROPPCHK EXEC if communication with one of the specified distributed nodes were lost for such a period.

IBM supplies sample PROPHCHK and PROPPCHK EXECs. See “Communication Exit (PROPHCHK/PROPPCHK)” on page 110. You can use the sample PROPHCHK and PROPPCHK EXECs, modify them, or replace them with user-written execs.

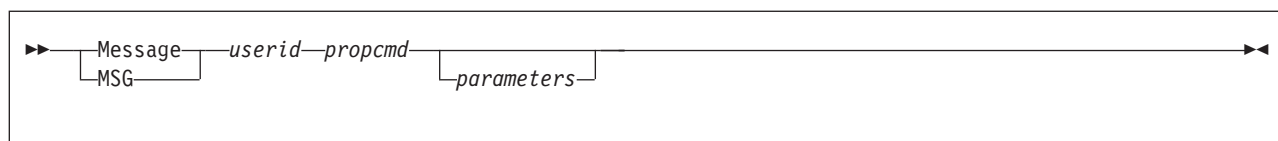
Invoking Programmable Operator Facility Commands

The commands used with the Programmable Operator Facility can be executed by anyone who is authorized by the active routing table. If an unauthorized user enters a programmable operator facility command, the command is not executed but is routed to the logical operator. To send a command to the programmable operator facility you must send a message to the programmable operator facility virtual machine. The text of the message is the command to be issued. Unless otherwise noted, the response from the command is returned to the user who sent the command to the programmable operator facility virtual machine. Responses are sent one line at a time by the CP MSGNOH or MESSAGE commands. For NCCF or NetView operators, the responses are sent by the Programmable Operator/NCCF Message Exchange (PMX). These responses are displayed in the language in which the programmable operator virtual machine is set.

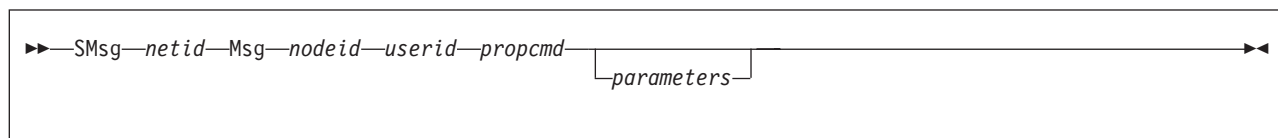
The format of the message sent to the programmable operator facility virtual machine is the same as used with the CP MESSAGE command for local use and the CP SMSG command for distributed (network) use. In a mixed environment, an NCCF or NetView operator uses PROP, an NCCF or NetView command, to send messages to the programmable operator facility virtual machine.

The local format is:

Programmable Operator Facility



The distributed (network) format is:



netid

is the user ID of the RSCS network machine at the user's node.

Msg

is the required RSCS message command. It can be entered as M or MSG.

nodeid

is the node ID of the programmable operator facility virtual machine.

userid

is the user ID of the programmable operator facility virtual machine.

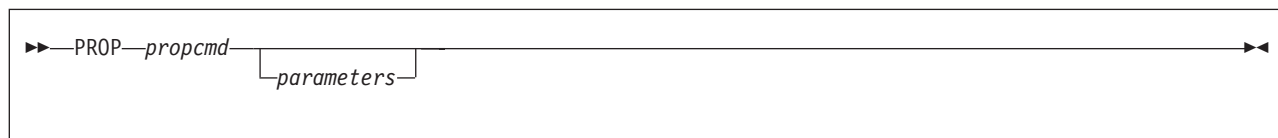
propcmd

is the command to be executed by the programmable operator facility. See Chapter 6, "Programmable Operator Facility Commands," on page 113 for information about the commands.

parameters

are the parameters associated with the command to be executed. The parameters will be either optional or required depending on the command that they are associated with.

The format for an NCCF or NetView operator station is:



propcmd

is the command that the programmable operator facility will execute

parameters

are the parameters associated with the command to be executed. The parameters will be either optional or required depending on the command that they are associated with.

Notes:

1. To use this format, you must be authorized by NCCF or NetView.
2. If you (NCCF or NetView operator) are not attached to the local system, use the NCCF or NetView ROUTE command to specify the domain where you want to execute the command.

Issuing Commands in the Single System Environment

If the Programmable Operator Facility is running in the operator's virtual machine, a user on the same physical machine might send the messages:

```
MESSAGE OPERATOR QUERY RTABLE
```

```
MESSAGE OPERATOR WHAT TIME IS SHUT DOWN?
```

where QUERY is a programmable operator facility command. See page 123 for information about the programmable operator facility QUERY command.

Figure 14 shows how messages are transferred between a user, the programmable operator facility, and the logical operator when all are located on the same physical system.

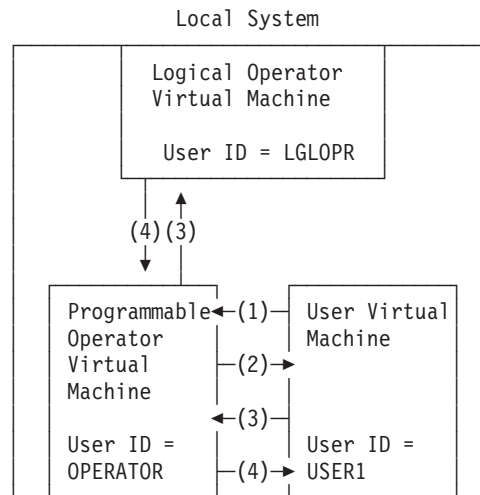


Figure 14. Example of Communication in the Single System Environment

1. USER1 enters the message:
message operator query rtable
2. The programmable operator facility responds by sending the following message to USER1:
PROP running with routing table 'PROP RTABLE A5'
3. USER1 enters the message:
message operator what time is shut down?

Because this message cannot be processed by the programmable operator facility, the message is routed to the logical operator.

4. The logical operator responds to USER1's message by sending a message to USER1 through the programmable operator facility. (See "Ensuring a Complete Log" on page 91.)

Issuing Commands in the Distributed z/VM Environment

A user (USER21) on another z/VM system (assuming a remote system with the node ID of NODE2, a network machine with the ID of NET2, and that the programmable operator facility is running in the operator's virtual machine with a user ID of OPERATOR) might send the messages:

Programmable Operator Facility

```
message operator query rtable
```

```
message operator what time is shut down?
```

where QUERY is a programmable operator facility command. See page 123 for information about the programmable operator facility QUERY command.

Figure 15 shows how a user on one physical system exchanges messages with the programmable operator facility on a different physical system and how the programmable operator facility responds directly to a user on the same physical system or routes messages to the logical operator who is located on a different physical system.

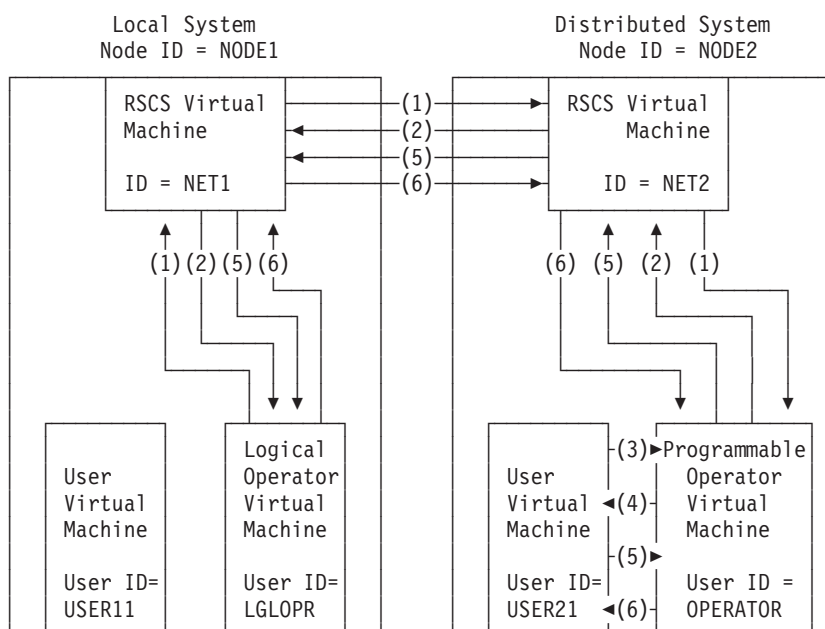


Figure 15. Example of Communication in the Distributed Environment

1. LGLOPR enters the message:


```
cp smsg net1 msg node2 operator query rtable
```
2. The programmable operator facility responds by sending the following message to LGLOPR:


```
PROP running with routing table 'PROP RTABLE A5'
```
3. USER21 enters the message:


```
message operator query rtable
```
4. The programmable operator facility responds by sending the following message to USER21:


```
PROP running with routing table 'PROP RTABLE A5'
```
5. USER21 enters the message:


```
message operator what time is shut down?
```

Because this message cannot be processed by the programmable operator facility, the message is routed to the logical operator.

- The logical operator responds to USER21's message by sending a message to USER21 through the programmable operator facility. (See "Ensuring a Complete Log" on page 91.)

Issuing Commands in a Mixed Environment

In a mixed environment, message paths are more complex. A user on a z/VM system (VMUSER3) might send the messages:

```
message operator query rtable
message operator what time is shut down?
```

where QUERY is a programmable operator facility command. See page 123 for information about the programmable operator QUERY command.

The message sent containing the QUERY command simply follows the paths of a message sent by a user to the programmable operator facility in the same system. The message is processed by the programmable operator facility and a response is returned to the user (VMUSER3).

Figure 16 shows how messages are transferred between a user, the programmable operator facility, and an NCCF or NetView logical operator. It also shows how messages and commands are transferred from an NCCF or NetView operator to the programmable operator facility and its z/VM system.

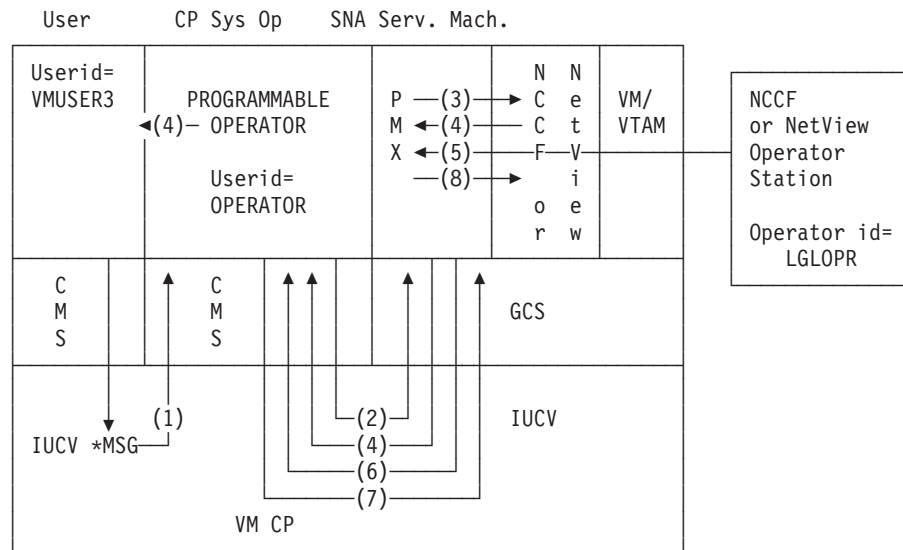


Figure 16. Example of Communication in the Mixed Environment

- VMUSER3 enters the message:
message operator what time is shut down?

that reaches the programmable operator facility through the IUCV *MSG service. The programmable operator facility determines that the message is to be routed to the logical operator.

- The programmable operator determines that the logical operator is an NCCF or NetView operator and sends the message though IUCV to the PMX.

Programmable Operator Facility

3. The PMX, upon receiving the message from IUCV, queues (through the NCCF or NetView Message Queueing Service) the message for display on the specified NCCF or NetView operator's console.
4. Upon receiving the message from VMUSER3, the NCCF or NetView logical operator issues the command:

```
prop cmd msg vmuser3 shutdown is at 10:00 pm.
```

which travels back through the PMX, IUCV, and the programmable operator facility, before the response reaches VMUSER3's screen.

5. An NCCF or NetView operator enters a programmable operator command:

```
prop query rtable
```

which is queued for the PMX.

6. The PMX sends the command as an IUCV message to the programmable operator.
7. The programmable operator receives the message from IUCV and processes it. Upon determining that the requester was an NCCF or NetView operator, the programmable operator sends any responses through IUCV to the PMX.
8. The PMX receives the response:

```
PROP running with routing table 'PROP RTABLE A5'
```

from IUCV and queues the response for display on the specified NCCF or NetView operator's terminal.

Helpful Hints

The typing of large text strings, such as the message command string preceding the message text, can be minimized by assigning the string, up to the user ID, to a PF key. An exec can also be used that prompts for, or accepts as parameters, that part of the message command that the user must type in.

For example:

```
set pf01 immed msg net1 msg node2 operator query rtable
```

will allow the logical operator to press PF01 to find out the name of the active routing table when the logical operator and programmable operator facility are located on different physical systems.

The logical operator can use the CMS NAMES command to set up nicknames for remote users and can use the CMS TELL command to send messages to those users. If the logical operator has a NAMES file entry assigning the nickname PROP1 to OP at node NODE1, the logical operator could send the following messages:

```
tell prop1 query rtable
```

```
tell prop1 cmd wng all ...will re-ipl in 5 min - pls logoff.
```

These commands are described in the *z/VM: CMS Commands and Utilities Reference* and in the *z/VM: CMS User's Guide*.

Action Routines

PI

Action routines are programs or execs that receive control in response to the match of a message and a routing table entry. They handle a particular type of message or command intercepted by the programmable operator facility. The action routines supplied with the programmable operator facility are DMSPOR, DMSPOS, and DMSPOL. These routines (or subroutines in the case DMSPOR) correspond to the programmable operator facility commands described in Chapter 6, “Programmable Operator Facility Commands,” on page 113. The supplied action routines need no tailoring to provide you with the control and function needed to operate the programmable operator facility.

You can extend the programmable operator facility by writing new action routines and adding them to the appropriate routing tables. Action routines can be written as execs or Basic Assembler Language programs. (Basic Assembler Language routines must also be added to the PROPLIB LOADLIB. PROPLIB LOADLIB is built by the VMSES/E VMFBLD EXEC using the DMSBLPRP build list. If you want to add your own routines, you must provide a local modification to the DMSBLPRP build list. See the discussion on installing local service in the *z/VM: Service Guide* for information on creating local modifications.)

When the programmable operator facility invokes an action routine, it assumes that the originator of the message (the requester) is authorized to receive any responses or error messages issued. Therefore, all VMCONIO responses (such as console I/O generated by LINEDIT, WRTERM, DMSERR, &TYPE, or SAY) or error messages resulting from CP commands are returned to the requester. However, if the action routine uses the extended DIAGNOSE code X'08' to issue a CP command and receive the responses in a buffer, the programmable operator facility never sees these responses, and therefore, cannot send them to the requester. (For more information on DIAGNOSE code X'08', refer to the *z/VM: CP Programming Services* book.)

An action routine is provided with the programmable operator facility that uses the extended DIAGNOSE code X'08' to issue a CP command and receive the responses in a buffer. This method ensures that the responses, error messages, or informational messages from the CP command are NOT presented to the programmable operator facility as IUCV message types 3, 4, or 7, respectively. Instead, the programmable operator will send the buffered messages or responses to the requester. (To review the IUCV message types, see 80.)

If an action routine abends, abend error messages are sent to the logical operator and the requester (if any). Control is returned to the point in the programmable operator facility immediately following the action routine call.

Note: Programs written in Basic Assembler Language can access the parameter list built by the programmable operator facility. The programmable operator parameters are available in a different fashion for EXEC action routines; execs may be written using the REXX, EXEC 2, or CMS EXEC languages.

Action Routine Interface

Action Routine Call Interface

Action routines are loaded by the programmable operator facility as CMS nucleus extensions. As a result, they must be invoked by the programmable operator facility as CMS commands by SVC 202. Also, addresses cannot be resolved between separate nucleus extensions; they must be passed dynamically if they are desired.

Action Routine Parameter Interface

An installation can write additional action routines in assembler language. Action routines may also be written as execs. Programs written in assembler language can access the parameter list built by the programmable operator facility. (The programmable operator parameters are available in a different fashion for EXEC action routines.) The parameter list contains a list of addresses pointing to data that may be significant to the action routine invoked. The programmable operator facility then passes the address of the list as a parameter when it invokes the required action routine.

The register conventions used for invoking an action routine are:

- Register 1 points to a list of eight-byte tokens (CMS PLIST) containing the following information:

TOKEN 1

Contains the command name (action routine name).

TOKEN 2

Contains two fullwords. These fullwords contain the following:

Fullword 1 -

Reserved for IBM use.

Fullword 2 -

Contains the address of a list of addresses that point to data that may be needed by the action routine. This list is described by the PARMLIST DSECT.

TOKEN 3

Contains eight X'FF's to mark the end of the parameter list.

- Register 13 points to a standard OS 18 word save area.
- Register 14 points to the address that receives control when the action routine completes processing, that is, the address to which the action routine must return control.
- Register 15 points to the action routine entry point and may be used as a base register.

Figure 17 on page 103 offers a graphic representation of the previous discussion. It illustrates the data areas that can be accessed through Register 1.

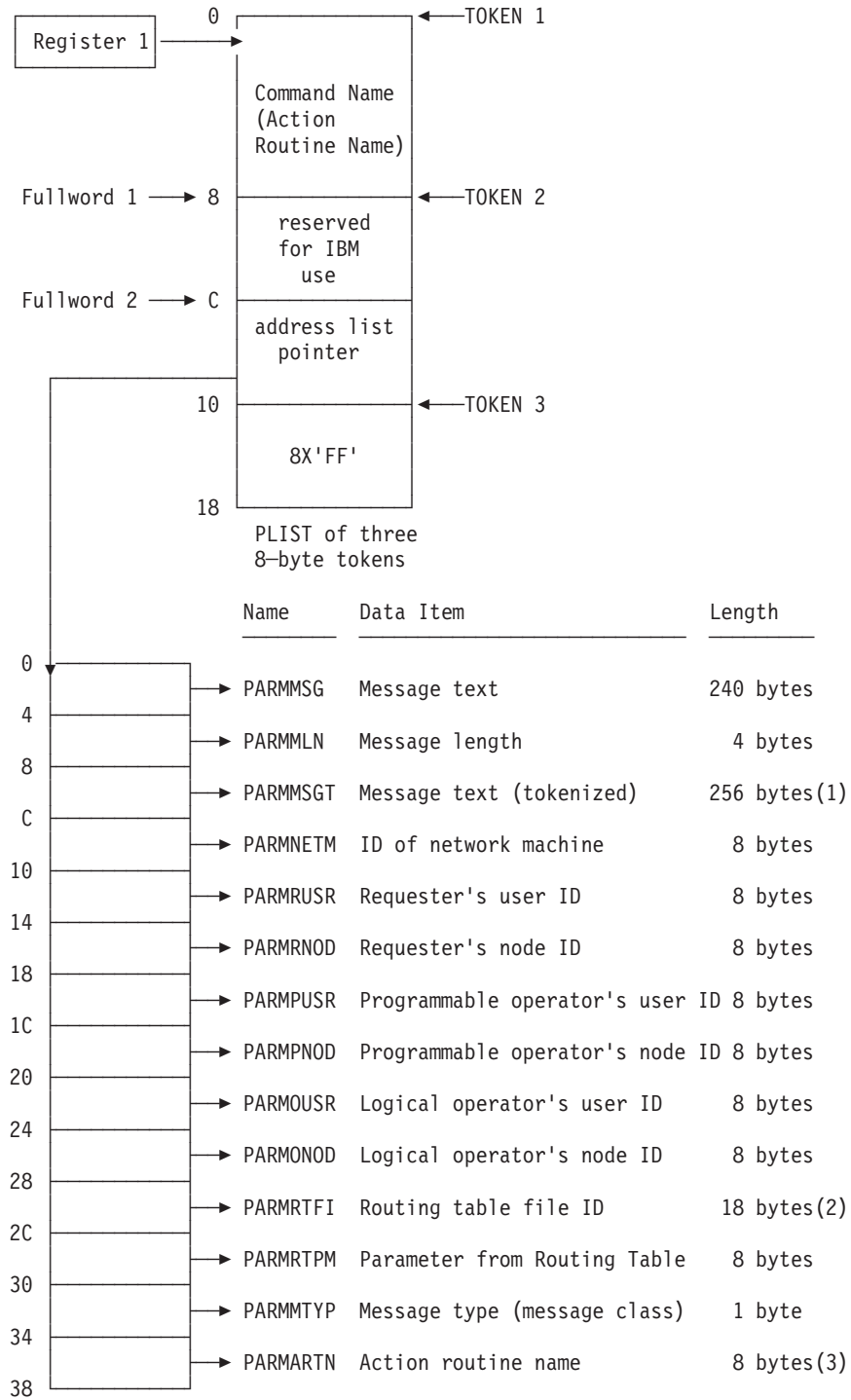


Figure 17. Register Conventions for Invoking an Action Routine

Address List described by PARMLIST DSECT

Notes:

1. In addition to the original message text, the message text is also provided in CMS tokenized form (8-byte tokens followed by 8 X'FF's).
2. The Routing table file ID is made up of the file name (8 bytes), file type (8 bytes), and file mode (2 bytes).

Programmable Operator Facility

3. The high-order bit (X'80') of the last fullword of this list of addresses is set to one to indicate that it is the last entry in the list according to standard OS linkage conventions.

Exec Action Routines

Action routines may also be written as execs in the REXX, EXEC 2, or CMS EXEC languages. The programmable operator parameters are available in a different fashion for exec action routines. The method is described below.

1. Having determined that the action routine is an exec, the programmable operator facility calls the action routine accordingly.
2. The following information is passed as parameters (arguments) on the exec invocation, in this order:
 - Requester's user ID
 - Requester's node ID
 - Logical operator's user ID
 - Logical operator's node ID
 - Message type class
 - The programmable operator facility's user ID
 - The programmable operator facility's node ID
 - Networking machine user ID
 - RTABLE file name.
3. The following parameters are stacked LIFO for the exec in this order:
 - RTABLE PARAMETER field contents
 - Message text.

To ease handling by an exec, if the requester is CP, the requester's user ID and node ID are "CP".

Writing Action Routines

How an action routine is written using Basic Assembler Language depends on the functions that the action routine performs and the conditions under which it runs. Since the programmable operator can use message content and message origin to determine which action routine to call, it may not be necessary for the action routine to check any further conditions. However, by using the PARMLIST supplied by the programmable operator facility, the action routine may obtain additional information about the message. Each entry in the PARMLIST points to some item of data about the message just received or about the programmable operator environment.

As described in the section, "Action Routine Parameter Interface" on page 102, the information initially provided to the action routine is in the form of a CMS tokenized PLIST. By loading the second fullword of the second token of that PLIST into a register, the user can establish addressability to the PROP PARMLIST.

For example,

```
SAVE (14,12)          SAVE REGISTERS
LR   R12,R15          LOAD BASE REGISTER
USING ROUTINEX,R12    ESTABLISH ADDRESSABILITY
L    R2,12(,R1)       LOAD PROP PARMLIST ADDRESS
USING PARMLIST,R2     PARM ADDRESSABILITY
```

These instructions would be sufficient for many action routines to establish addressability for the action routine and the PROP PARMLIST. The following instructions could then be used to obtain the addresses of the requester's (message originator's) user ID and node ID.

```
L   R4,PARMRUSR      GET REQUESTER'S USER ID ADDRESS
L   R6,PARMRNOD     GET REQUESTER'S NODE ID ADDRESS
```

The Programmable Operator DSECTs, such as PARMLIST, define the above labels. To access the PROP DSECT, global the DMSGPI MACLIB. To include the PROP DSECT in the action routine insert the following assembler instruction in the source file for the routine:

```
COPY  PROP
```

In addition, it may be desirable to include the CMS REGEQU macroinstruction for register equates. When the action routine is complete, it is necessary to restore registers and branch to the address in register 14, or use the OS RETURN macro.

Action Routine Error Message and Response Handling

It is sometimes necessary to issue error messages and/or responses from an action routine that are to be sent to the message originator (requester). To do this, the action routine should simply TYPE the messages/responses (by LINEDIT, DMSERR, or WRTERM for action routines written in BAL, and &TYPE or SAY for EXEC action routines). Upon completion of the action routine, the programmable operator facility collects these VMCONIO messages (IUCV type 5 messages), as well as any error messages (IUCV type 6 messages), and sends them back to the requester. To prevent the programmable operator facility from sending error messages from CP commands back to the requester, the action routine should be coded to use the extended DIAGNOSE code X'08' to obtain the responses in a buffer.

Handling Console I/O in an Action Routine

The installation must determine how an action routine is to handle console I/O generated by the virtual machine and CP. Normal operation of the programmable operator facility sets VMCONIO to IUCV (by default) before calling an action routine and sends the VMCONIO to the message originator (requester). If it is desired that console I/O (VMCONIO) produced by the action routine be typed on the programmable operator virtual machine console, the action routine must SET VMCONIO OFF. However, because there would not normally be an operator at the programmable operator virtual machine console, an installation can code an action routine to receive and handle VMCONIO instead of allowing the programmable operator to receive it and send it to the requester (message originator). To accomplish this, the action routine can receive the VMCONIO that was generated by using the IUCV RECEIVE function with message type 5 specified as the IUCV target class (TRGCLS). For details on using IUCV, see the *z/VM: CP Programming Services*.

An example of this may be found in the subroutine CALLARTN of the IBM-supplied module DMSPOA. To use IUCV, it is necessary to include the CP COPY files, IPARML and EQU.

CP generated console I/O (CPCONIO) should be handled differently than above. The CPCONIO setting should not be changed because this could cause the programmable operator facility to miss some asynchronous CP messages.

If the action routine is to receive the responses from CP commands that it issues, it should use the DIAGNOSE code X'08' support with a command response buffer,

Programmable Operator Facility

rather than trying to receive it with IUCV. (See *z/VM: CP Programming Services* for information about DIAGNOSE code X'08'.) The reason for this is that other CP messages can be mixed in with the command response, and therefore the program cannot be assured of receiving its response in consecutive IUCV messages.

If the CP command response is to be typed on the programmable operator's virtual machine console, the action routine should use a CMS function, such as WRTERM, to write the lines in the program's CP command response buffer to the terminal.

PI end

Description of Supplied Action Routines

The action routines supplied with the programmable operator facility are DMSPOR, DMSPOS, and DMSPOL. A parameter must be supplied for module DMSPOR. This parameter is the name of the function or action that DMSPOR is to perform. DMSPOS may be invoked along with a parameter, which, in this case, is a user ID or nickname. DMSPOL may be invoked with a parameter, which is a routing table name.

Note that new action routines are not required to be in this format. The programmable operator facility supports any desired number of action routines. Each one is loaded separately when the programmable operator facility is initialized, or when a LOADTBL command is issued.

The following sections describe the action routines that are supplied with the programmable operator facility. These action routines (or subroutines in the case of DMSPOR) correspond to the programmable operator commands described in Chapter 6, "Programmable Operator Facility Commands," on page 113.

DMSPOR—Miscellaneous Action Subroutines

The DMSPOR action routine must be invoked with one of the following parameters, which is the name of the subroutine for the function or action that DMSPOR is to perform.

GET - Send the indicated file to an authorized user

This routine sends programmable operator files, such as log and feedback files, to requesting user. The files are sent using the CMS DISK DUMP command.

LGLOPR - Process the LGLOPR command

This routine processes the LGLOPR command to assign (ASN), release (RLS), or replace (RPL) the logical operator as specified by an authorized user of the LGLOPR command.

QUERY - Return a response to a user query

This routine returns only one of the following information types:

- Name of the active routing table (RTABLE parameter)
- Status of PROPCHK node-checking (PROPCHK parameter)
- Status of HOSTCHK node-checking (HOSTCHK parameter)
- Name of the currently assigned logical operator (LOGOPR parameter)
- Status of the logging messages (LOGGING parameter)
- Message prefixes for virtual machines (OUTID parameter).

SET - Change the status of specific functions

This routine stops or resumes: the periodic checking of the distributed systems or the host system, the logging of messages in the log file, or the appending of the message prefix.

STOP - Stop the programmable operator facility

This routine stops the programmable operator operation after processing currently queued messages. The programmable operator virtual machine returns control to CMS.

TOFB - Write a message to the feedback file

This routine attaches the date and time received to the head of the incoming message and writes it to the feedback file. See “The Feedback File” on page 93 for more information.

TOVM - Execute a CP/CMS command

This routine is invoked when the programmable operator CMD command is issued. The text following CMD is regarded as the CP or CMS command to be executed in the programmable operator virtual machine. The command is treated as if it were entered at a CMS console (that is, such things as synonyms and the IMPCP and IMPEX settings apply to its interpretation as supported by the CMS COMMAND SUBCOM environment). The response to the executed CP or CMS command is returned to the authorized user who invoked the CMD command.

Authorized users of the CMD command should be aware of the following:

- Issuing commands that alter or overlay CMS storage, such as CP DEFINE STORAGE, CP IPL CMS, CP SHUTDOWN, and so on, has an adverse effect on the operation of the programmable operator facility.
- Reissuing the PROP command once the programmable operator facility is running causes the programmable operator facility to stop operating correctly. The user must re-IPL CMS and restart the programmable operator facility using the procedure described under “Invoking the Programmable Operator Facility” on page 62.
- Issuing commands that cause a VM READ or CP READ (interactive commands such as the DDR command) stop the operation of the programmable operator facility. The programmable operator facility must then be restarted in the manner described under “Invoking the Programmable Operator Facility” on page 62.
- Line-editing characters (pound sign (#), for example), as defined by the CP TERMINAL command, are not recognized as line-editing characters by the programmable operator facility.
- The CMS immediate commands (for example, HB, HI, HO, HT, HX, RO, RT, SO, TE, and TS) are not recognized by the programmable operator facility. If a user enters any of these commands, he receives an Unknown CP/CMS command response from the programmable operator facility.

In general, the programmable operator facility does no checking to ensure or prevent any of the above circumstances from occurring.

DMSPOS—Route a Message

DMSPOS sends (routes) a message to the user specified in the RTABLE PARAMETER field. The user is identified by a nickname from the CMS *userid* NAMES file or by a user ID. If the user is on another system, identification must be through a nickname. LGLOPR may be specified in the PARAMETER field of the routing table, which would indicate that DMSPOS uses the value of the currently assigned logical operator. If no logical operator has been explicitly assigned,

Programmable Operator Facility

DMSPOS uses the value specified in the LGLOPR statement in the routing table. This is the default if the parameter field is left blank.

A message longer than 94 characters (including the 19-character programmable operator origin ID) is split and sent as multiple messages. The first piece is no more than 94 characters. The remaining pieces are no longer than 91 characters, and preceded by a continuation mark (...). This splitting ensures that the message is small enough to be sent through an RSCS network. One exception to this message splitting is when the logical operator is NetView. For this case, DMSPOS does not split messages routed to the NetView operator.

If an error occurs because of an incorrect target ID, for example, the nickname was not in the *userid* NAMES file, the programmable operator tries to send the message to the logical operator.

Messages are sent with the CMS TELL command. If the programmable operator virtual machine is authorized (class B), the CP MSGNOH command is used. If the virtual machine is not authorized to use the CP MSGNOH command, then the CP MESSAGE command is used. For more information on the CMS TELL command, see the *z/VM: CMS Commands and Utilities Reference*.

Notes:

1. Using the CMS TELL command requires the user to have a SYSTEM NETID file set up. For more information, see "Use the SYSTEM NETID File to Configure Communication Across an NJE Network" on page 49.
2. DMSPOS must not be invoked if the logical operator virtual machine is the same as the programmable operator virtual machine. Also, a parameter should not be specified that directs the message to the programmable operator virtual machine.

To prevent a LOOP condition, a message being handled may not be sent on to the routing target by the DMSPOS routine. A message is not sent if it falls into any one of the following categories:

1. The preceding message could not be sent, and the current message is the same as the preceding message. In other words, the programmable operator receives an error return code when trying to send consecutive identical messages.
2. The programmable operator tries to route a message that originated from the networking virtual machine on the programmable operator's node. The message is identical with the last message from that virtual machine that the programmable operator tried to route. For example, a local network machine detects that a link is down.
3. The programmable operator tries to route a message that originated from the networking virtual machine on another system (node), and that message is identical with the last message that the programmable operator tried to route. For example, messages are being routed from system A to the logical operator who is supposed to be on another system (B), but is not logged on. The networking virtual machine on system B sends the programmable operator an error message each time it tries to route a message to the logical operator. This could cause a loop if not detected.

If the DMSPOS routine tries to send a message to the logical operator, but for some reason the logical operator's network node is unavailable for messages (not logged on, SMSG and/or MSG off), DMSPOS detects this condition and stops any

further attempts to send that message. The unsent message, although logged in the current day's log file, is not displayed at the logical operator's console.

DMSPOL—Load a Routing Table

This routine dynamically loads the routing table indicated by the programmable operator LOADTBL command. The routing table name must be *filename* RTABLE, where *filename* can be any name that conforms to CMS file naming conventions. Although the routing table name specified with the LOADTBL command takes precedence, it is also possible to specify in a routing table the file name of the table to be loaded as a parameter to the action routine. (This can be used as a default.) Therefore, any message selected by the system programmer can cause a new RTABLE to be loaded. Also, the programmer can change the LOADTBL default of "PROP" to whatever is desired without changing the LOADTBL action routine.

Note: With the loading of the routing table done by a separate action routine, it is possible for the other routines, DMSPOR and DMSPOS and any user-written routines, to be replaced when a LOADTBL occurs. This permits changes to action routines other than DMSPOL to be made dynamically without stopping the programmable operator.

Exit Execs

Exit execs receive control when the programmable operator facility encounters certain error and communication status conditions. PROPLGER EXEC is invoked when there is a log error. PROPHCHK EXEC or PROPPCHK EXEC is invoked when there is a change in communication status. The programmable operator facility provides sample exit execs which are used in their sample form. These sample exit execs can be modified or replaced with user-written execs with the same naming convention.

Exit Exec Interface

PI

The programmable operator facility exit execs have the same parameter list provided as an exec action routine, with the exception that no RTABLE parameter field value and no message text are stacked for the exec. See "Exec Action Routines" on page 104 for a list of these parameters. When an exit exec is called, contents of the program stack depend upon which exit is being invoked. Descriptions of the stack contents for the different types of exits follow.

Notes:

1. Some of the parameter values have no meaning for a particular exit exec and their use is left to the discretion of the exec writer. For example, the requester's user ID and node ID, and the message class have no meaning for the log error exec, PROPLGER, and the communication exit execs, PROPPCHK and PROPHCHK.
2. The programmable operator facility does not trap VMCONIO-type or CP MSGS produced by exit execs as it does for action routines.

Log Error Exit (PROPLGER)

If a virtual machine resource limit is reached, such as *disk full* or *file space full*, the programmable operator may not be able to write another record in the log file. If this happens, the programmable operator facility stacks (LIFO) the error code received from the CMS FSWRITE function and calls the PROPLGER EXEC for recovery.

The IBM-supplied PROPLGER EXEC:

Programmable Operator Facility

1. Closes the current log file.
2. Sends the last two log files to the logical operator.

Note: PROPLGER does not try to send the log files to the logical operator if the logical operator is an NCCF or NetView operator. The files are instead sent to the programmable operator's virtual reader.

3. Erases the last two log files.
4. Issues one of the following return codes:

RC=0 Recovered from error. The programmable operator facility should retry logging. If it is still unable to log, the programmable operator sends an error message to the logical operator.

RC=4 Unable to do recovery. The programmable operator should send an error message to the logical operator.

If the PROPLGER EXEC cannot be found, the programmable operator facility acts as if RC=4 has been returned and sends an error message to the logical operator. Whatever action is taken, the programmable operator continues operation.

If the same logging error occurs on two successive logging attempts (for example, two consecutive incoming messages cause the same logging error), the programmable operator sets LOGGING to OFF. This prevents unpredictable looping in some situations. Note, though, that the logical operator may receive only two error messages when logging errors occur.

When using an SFS directory accessed as file mode A, PROPLGER EXEC is invoked whenever the file space threshold limit is exceeded. However, if the threshold is set too high or other users are writing to this storage group, the programmable operator facility may not detect certain resource limits until it tries to close the log file. At that point the programmable operator will not be able to invoke the PROPLGER EXEC, and some log data could be lost. If it is important for you to minimize the potential loss of log data, consider using a minidisk for the log file instead of an SFS directory.

Communication Exit (PROPHCHK/PROPPCHK)

The programmable operator facility does communication checking (node-checking) based on the checking intervals and response wait intervals specified on the HOSTCHK statement or PROPCHK statement in the routing table. When a change in communication status is detected, the programmable operator facility stacks (LIFO) an entry having the following format:

nodeid UP|DOWN

nodeid is the RSCS node ID of a node that has changed communication status.

UP indicates that the node had not been responding and now has resumed responding to acknowledgement requests.

DOWN

indicates that the node had been responding and has ceased to respond.

Also stacked will be the total number of node ID entries that are stacked.

When the communication checking is complete, if any entries were placed in the stack, the programmable operator calls one of the following execs:

- PROPHCHK EXEC if the programmable operator was checking the logical operator's node as specified in the HOSTCHK statement.

The IBM-supplied PROPPCHK EXEC types a message on the programmable operator console and sends a message to the MAINT user ID indicating that communication with the logical operator (the *host*) has been broken or restored.

- PROPPCHK EXEC if the programmable operator was checking the nodes specified in the PROPCHK statement.

The IBM-supplied PROPPCHK EXEC notifies the logical operator about each node that has failed to respond or has resumed responding. A return code of 0 should be returned by the exec.

Note: PROPPCHK does not try to send a message to the logical operator if the logical operator is an NCCF or NetView operator.

PI end

Problem Determination—Debug Mode

Debug mode determines problems with the programmable operator facility itself. It allows responses to commands issued from the programmable operator virtual machine console to be returned to the console without being intercepted by the programmable operator facility. This permits any CP command to be issued without having its response trapped by the programmable operator facility.

SET DEBUG ON may be used after the programmable operator facility responds with the message:

```
PROP running - enter STOP to terminate
```

indicating that the programmable operator facility is running and operational. The programmable operator facility then responds with the message:

```
PROP is running in DEBUG mode
```

which is also written to the log file. In debug mode, the programmable operator facility waits to receive messages from another virtual machine, or for the system programmer to enter input from the console. Because only two commands, STOP and SET are accepted from the programmable operator virtual machine console, the system programmer must enter the CP environment (using the PA1 key) to enter any CP commands. Otherwise, the commands are intercepted and rejected as incorrect programmable operator commands.


Pressing the PA1 key or entering the BEGIN command returns control to the programmable operator facility. From this environment, entering SET DEBUG OFF returns the programmable operator facility to its normal function of trapping messages.

Chapter 6. Programmable Operator Facility Commands

The Programmable Operator Facility increases the efficiency of system operation by allowing systems to be operated remotely. To do this, the Programmable Operator Facility handles all messages and requests sent to its virtual machine according to preprogrammed actions. For further information on the Programmable Operator Facility, refer to page 57.

This chapter contains commands used with the Programmable Operator Facility. These commands can be executed by anyone who is authorized by the active routing table. An *authorized user* must have a routing table entry for the command he or she wants to issue. If an unauthorized user issues a programmable operator facility command, the command is not executed but is routed to the logical operator.

All programmable operator facility commands are sent by users as messages; these messages should be in English to ensure that they get routed properly.

CMD


Purpose

Use the CMD command to execute CP or CMS commands in the programmable operator facility virtual machine. The command is accepted or rejected by CP based on the CP user class defined for the programmable operator facility virtual machine. Also, the programmable operator facility may reject or accept the command based on the authorization granted in the active routing table. The user class assigned to the programmable operator facility virtual machine is determined by the installation. If accepted, the response from the command is returned to the issuer of the command.

Operands

vmcmd

is the z/VM command sent to the programmable operator facility virtual machine for execution according to the CMS IMPCP and IMPEX settings.

Usage Notes

1. Any commands that alter or overlay CMS storage (CP DEFINE STORAGE, CP IPL CMS, CP SHUTDOWN, and so forth) will have an adverse effect on the operation of the programmable operator facility and should not be issued under the programmable operator facility.
2. Reissuing the PROP command after the programmable operator facility is running will cause it to stop operating correctly. The user must then re-IPL CMS and restart the programmable operator facility using the procedure described in “Invoking the Programmable Operator Facility” on page 62.
3. Entering commands that cause a VM READ or a CP READ (such as the DDR command) will stop the programmable operator facility. It must then be restarted using the procedure described in “Invoking the Programmable Operator Facility” on page 62, or the read must be answered from the console of the programmable operator facility virtual machine. Commands of this type should not be sent to the programmable operator facility.
4. Line-editing characters (CHARDEL, LINEDEL, LINEND, and ESCAPE), although interpreted by your terminal when you have SET LINEDIT ON in effect, are not interpreted as line-editing characters when sent to the programmable operator facility. In other words, they are interpreted as the characters they are (that is, @, ¢, #, :). For example, the string:

```
m op cmd access 191 a"#receive
```

will cause an Invalid filemode message to be returned to the issuer. The default line-editing characters may be defined by the installation. You may also define your own line-editing characters by using the CP TERMINAL command.

5. The programmable operator facility does not recognize the CMS immediate commands (HB, HI, HO, HT, HX, RO, RT, SO, TE, and TS). If you issue any of these commands, the programmable operator facility issues the message, Unknown CP/CMS command.

Examples

The following is a list of examples that show how the programmable operator CMD command could be issued from a virtual machine console:

```
m op cmd indicate
m op cmd query files
m op cmd query printer all
m op cmd erase lg861015 node1 a5
m op cmd query search
m op cmd listfile
```

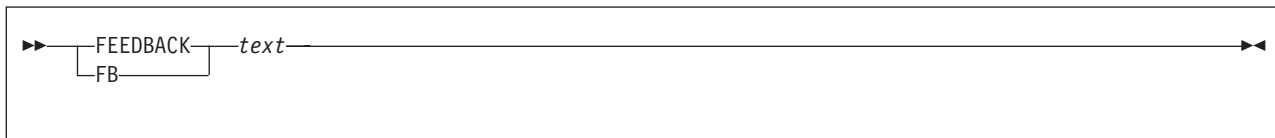
The following is a list of examples that show how the programmable operator CMD command could be issued from an NCCF or NetView[®] operator terminal:

```
prop cmd indicate
prop cmd query files
prop cmd query printer all
prop cmd erase lg861015 node1 a5
prop cmd query search
prop cmd listfile
```

Responses

The response returned by CP or CMS is sent to the issuer of the command. After the response from CP or CMS, the programmable operator facility responds with:
Command complete

FEEDBACK



Purpose

Use the FEEDBACK command to place comments about the operation of the system and/or the programmable operator facility in the feedback file. These comments are available for review by personnel responsible for maintenance of the programmable operator facility. The comment (preceded by the date and time it was received, and the sender's user ID and node ID) is placed in a file named *FEEDBACK nodeid A5*. (If the node ID is not available, the file type for the feedback file will default to PROFILE.)

Operands

text

is the user's comments to be placed in the feedback file

Usage Notes

1. Authorized z/VM users can retrieve the feedback file using the GET FEEDBACK command (this does not include NCCF or NetView users).
2. The length of the message is limited by the maximum length of the command used to send the message to the programmable operator facility. If the user desires to send a longer message, the command must be used multiple times.

Examples

The following example shows how the programmable operator FEEDBACK command could be issued from a virtual machine console:

```
m op feedback system response was slow during the morning shift.
```

The following example shows how the programmable operator FEEDBACK command could be issued from an NCCF or NetView operator terminal:

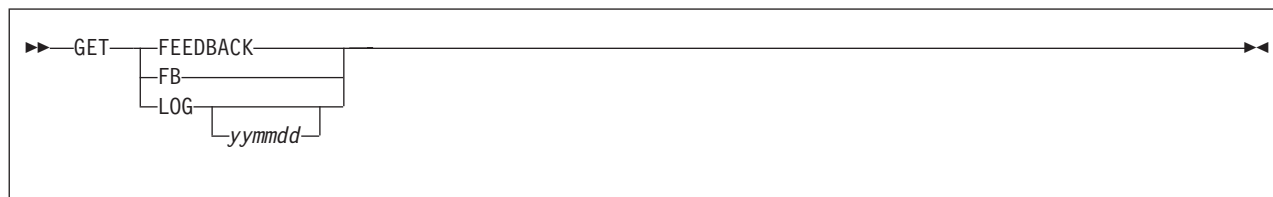
```
prop feedback system response was slow during the morning shift.
```

Responses

The programmable operator facility responds with:

```
Command complete
```

GET



Purpose

Use the GET command to retrieve one of the programmable operator facility files; the feedback file (FB or FEEDBACK) or the log file (LOG). The file is sent to the requesting user if he is authorized in the active routing table to receive it. If LOG is specified, the user will receive the log file for either the current day or the specified day.

Operands

FEEDBACK or FB

indicates that the feedback file is to be retrieved.

LOG

indicates that the log file for date *yymmdd* is to be retrieved. If no date is given, the log file for the current day is retrieved.

Usage Notes

1. An NCCF or NetView operator cannot use the GET command to retrieve the LOG and FEEDBACK files. Use CMS commands (and the programmable operator CMD command) to type the file(s) or portions of the files, or send the file(s) to some user ID where you can process them.
2. The file appears in the requesting user's virtual reader in DISK DUMP format. The user must execute a DISK LOAD or RECEIVE command to read the file.

Examples

The following examples show how the programmable operator GET command could be issued from a virtual machine console:

```
m op get fb
```

or

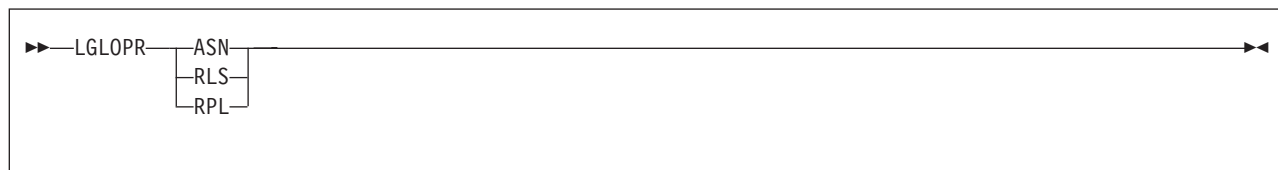
```
m op get log 920206
```

Responses

The programmable operator facility responds with:

```
Command complete
```

LGLOPR



Purpose

Use the LGLOPR command to assign or release yourself as the logical operator for the programmable operator facility under which the command is executed. This programmable operator command can be used by a z/VM user or an NCCF or NetView operator.

Users are authorized in the active programmable operator routing table.

Operands

ASN

assigns the issuer of the command as the logical operator, if a logical operator is not currently assigned (that is, the current LGLOPR is the default). If a logical operator is already assigned to the programmable operator facility, an error response is given.

RLS

releases the issuer from being the logical operator, if he currently is the logical operator and assigns the default LGLOPR. If the issuer is not the logical operator, no operation is performed and the system gives the following response:

DMS763E Not currently assigned as LGLOPR, cannot be released

RPL

replaces the current logical operator with the issuer of the command. The programmable operator determines if a logical operator is currently assigned. If there is, an implicit release is done. Then the issuer of the command becomes the logical operator.

Usage Notes

1. The system keeps the logical operator that is specified on the LGLOPR statement in the routing table as a default. You cannot release this logical operator. The LGLOPR ASN and LGLOPR RPL commands override the default.
2. To ensure that messages are not lost when changing logical operators, the new logical operator should issue a LGLOPR RPL command, rather than the current logical operator issuing a LGLOPR RLS command and the new logical operator issuing a LGLOPR ASN command.
3. If the default logical operator is the current logical operator and he issues the LGLOPR ASN command, CP will send a message stating that he already assigned as the logical operator
4. The HOSTCHK function is suspended when an NCCF or NetView operator or a local z/VM user is assigned as the logical operator. It is resumed when a remote z/VM user is assigned as the logical operator.
5. PROP uses the CP MSG facility to send messages to the logical operator. If the operator ID is OPERATOR, the message is sent to the system operator

regardless of the user ID. If the system operator has been changed with the SET SYSOPER command, be aware that the logical operator still can not have an ID of OPERATOR.

Examples

The following example shows how the programmable operator LGLOPR command could be issued from a virtual machine console:

```
msg op lglopr asn
```

The following example shows how the programmable operator LGLOPR command could be issued from an NCCF or NetView operator terminal:

```
prop lglopr asn
```

Responses

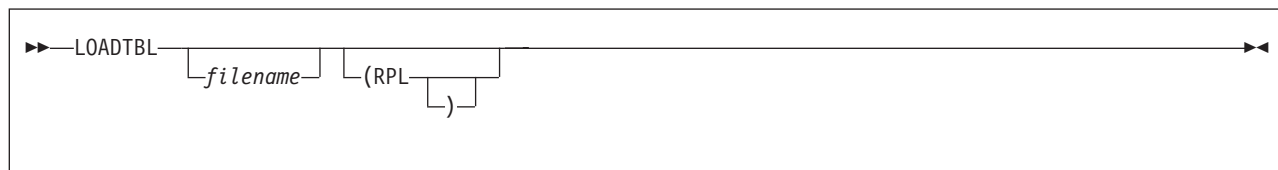
Both the new and old logical operators receive the message:

```
DMSPOR758I {NCCF|VM} user userid [nodeid] is now LGLOPR for PROP on node nodeid
```

to notify them of the change of logical operators. The requester also receives the response:

```
Command complete
```

LOADTBL



Purpose

Use the LOADTBL command to dynamically load a new routing table. You, as a z/VM user or an NCCF or NetView operator, can specify whether the currently assigned logical operator should be replaced by the logical operator specified in the new routing table.

Operands

filename

is the name of the routing table to be loaded. If the file name is not given, the default routing table (file name = PROP) is used.

RPL

replaces the currently assigned logical operator with the logical operator specified in the new routing table. The logical operator is replaced only after the new routing table has been successfully loaded.

If RPL is not specified, the logical operator in the new routing table simply becomes the new default logical operator, and any explicitly assigned logical operator (that is, a logical operator assigned by the LGLOPR command) remains the logical operator.

Usage Notes

1. If any action routines named in the specified routing table can not be located or loaded, an error message is issued, and the programmable operator facility drops any action routine modules associated with the specified routing table. It then tries to reload the action routine modules associated with the routing table that was active before the LOADTBL command was issued. If it cannot load these modules, the programmable operator facility terminates operation.

Note: If any of the action routines associated with the previous routing table were modified (that is, replaced in LOADLIB) between the time the programmable operator facility dropped the specified routing table modules and reloaded the previously active routing table modules, the modified version of the action routines are used when the previous routing table is reloaded.

2. Only the file name is used to identify the routing table file to the LOADTBL command.
3. Because DMSPOL is the action routine module executing the LOADTBL command, it is not dropped and reloaded as are the other action routine modules listed in the routing table. If you want to alter or replace DMSPOL, you must stop the programmable operator facility (using the STOP command), make the desired changes or replacement, and invoke the programmable operator facility again. See “Invoking the Programmable Operator Facility” on page 62 for information about starting the programmable operator facility and “Stopping the Programmable Operator Facility” on page 67 for information about stopping the

programmable operator facility. For information about writing a new programmable operator facility action routine, see page 104.

If the programmable operator virtual machine is set up so that the programmable operator is started automatically when CMS is IPLed in that virtual machine, it is sufficient to do the replacement and then IPL CMS again.

4. With the loading done by DMSPOL, it is possible for the other routines in DMSPOR to be replaced when a LOADTBL occurs. This permits changes to action routines other than DMSPOL to be made dynamically, without stopping the programmable operator. It is also possible to specify the name of a table to be loaded as a parameter to the action routine. The logical operator will be notified of the loading.
5. If the current logical operator is the default logical operator (not explicitly assigned), then the current logical operator will be replaced even if the RPL option is not specified.
6. When you issue LOADTBL and replace the logical operator, both the old and new logical operators receive the following message:

```
DMSPOR758I {NCCF|VM} user userid [nodeid] is now LGLOPR
for PROP on node nodeid
```

Both operators receive this message, even if you have not specified the RPL option.

Examples

The following example shows how the programmable operator LOADTBL command could be issued from a virtual machine console:

```
msg op loadtbl route3 (rp1)
```

The following example shows how the programmable operator LOADTBL command could be issued from an NCCF or NetView operator terminal:

```
prop loadtbl route3 (rp1)
```

Responses

The programmable operator facility responds with:

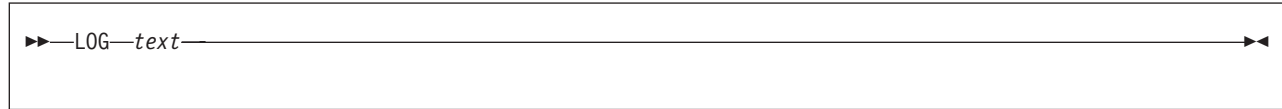
```
New RTABLE not loaded
```

```
PROP terminated
```

```
PROP running with routing table fn ft fm
```

LOG

LOG



Purpose

Use the LOG command to write a message to the log file. Use of the LOG command allows messages to be placed in the log file with no action taken by the programmable operator facility.

Operands

text...

is the message text to be placed in the log file.

Usage Notes

1. All messages are logged whether the LOG command is explicitly used.
2. Authorized z/VM users can retrieve the log file using the GET LOG command (this does not include NCCF or NetView users). The log file has a file ID of *LGyymmdd nodeid A5* where *yy* is the year, *mm* is the month, *dd* is the day, and *nodeid* is the node ID of the system on which the programmable operator facility is running. (If the node ID is not available, the file type for the feedback file will default to PROPFIL).
The user ID and node ID of the sender is recorded along with the message.
3. The length of the message is limited by the maximum length of the command used to send the message to the programmable operator facility. If you want to send a longer message, you must use the command multiple times.

Examples

The following example shows how the programmable operator LOG command could be issued from a virtual machine console:

```
m op log this message is to be logged.
```

The following example shows how the programmable operator LOG command could be issued from an NCCF or NetView operator terminal:

```
prop log this message is to be logged.
```

Responses

The programmable operator facility responds with:

```
Command complete
```

QUERY



Purpose

Use QUERY RTABLE to find the name of the active routing table.

Use QUERY PROPCHK and QUERY HOSTCHK to query the status of the programmable operator node-checking.

Use QUERY LOGGING to query the status of the logging messages.

Use QUERY LGLOPR to find the name of the currently assigned logical operator.

Use QUERY OUTID to find out the message prefix(s) for a virtual machine(s).

Operands

RTABLE

displays the name of the active routing table.

PROPCHK

displays a message with node-checking status. The message will state whether PROPCHK is set ON or OFF. If a *nodeid* is supplied, only the node specified is checked.

HOSTCHK

displays a message with node-checking status. The message will state whether HOSTCHK is set ON or OFF.

LOGGING

displays a message with logging status. The message will state whether messages and responses are being logged (LOGGING ALL), incoming messages and special programmable operator messages are being logged (LOGGING ON), or there is no log (LOGGING OFF). Also, if messages are being logged, it will state whether the messages are logged in uppercase format (UPCASE) or in their original format (LOWCASE).

LGLOPR

for a z/VM logical operator, displays a message with the user ID and node ID of the operator. For an NCCF or NetView logical operator, LGLOPR displays only the user ID (that is, operator ID) of the operator.

OUTID

displays one or more output identifiers (which will be prefixed to command output responses) and the associated virtual machine. If you specify the *userid* option, only the output identifier for that virtual machine will be displayed.

QUERY

Examples

The following is a list of examples that show how the programmable operator QUERY command could be issued from a virtual machine console:

```
m op query rtable
m op query propchk node1
m op query hostchk
m op query logging
m op query lglopr
m op query outid crrserv1
```

The following is a list of examples that show how the programmable operator QUERY command could be issued from an NCCF or NetView operator terminal:

```
prop query rtable
prop query propchk vmsys1
prop query hostchk
prop query logging
prop query lglopr
prop query outid crrserv2
```

Responses

Programmable operator facility responses:

```
PROP running with routing table fn
```

is received if the programmable operator facility is running, where *fn* is the file name of the active routing table.

```
{PROPCHK|HOSTCHK} is ON
```

is received if node-checking is in effect.

```
{PROPCHK|HOSTCHK} is OFF
```

is received if node-checking was specified in the RTABLE but is currently off.

```
PROPCHK is {ON|OFF} for nodeid nodeid
```

is received if a specified node ID is being queried.

```
DMSPOR762E Host checking is suspended -- LGLOPR not on a checkable node
```

is received if the logical operator is an NCCF or NetView operator or a local z/VM user.

DMSPOR690E {PROPCHK|HOSTCHK} not specified in RTABLE

is received if node-checking was not specified in the current RTABLE.

DMSPOR709E PROPCHK not specified in RTABLE for nodeid *nodeid*

is received if a node ID was specified on the QUERY command and node-checking was not specified in the current RTABLE for that node.

Logging ALL {UPCASE|LOWCASE}

is received if incoming messages and all programmable operator responses are being logged.

Logging ON {UPCASE|LOWCASE}

is received if incoming messages and special programmable operator responses are being logged.

Logging OFF

is received if no logging is being done.

DMSPOR758I {NCCF|VM} user *userid* {*nodeid*} is now LGLOPR for PROP on node *nodeid*

is received when the logical operator is being queried.

userid set output identifier for user *userid* to *msgprefix*

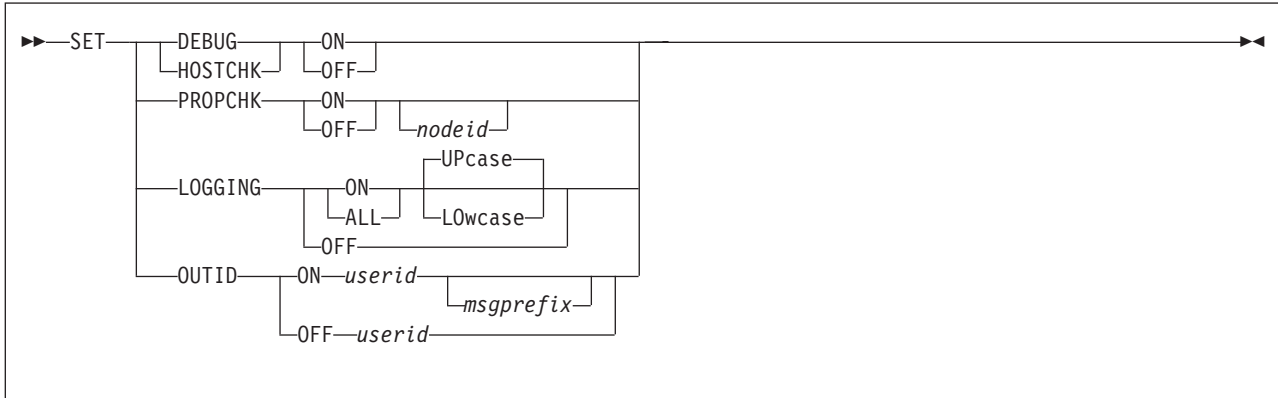
is received when the QUERY OUTID is successful.

No output identifier is defined.

is received from QUERY OUTID when no output identifiers are active.

SET

SET



Purpose

Use SET DEBUG to enter or exit the programmable operator facility DEBUG mode. DEBUG mode is used to do problem determination on the programmable operator facility.

Use SET PROPCHK to set the periodic checking of the programmable operator on the distributed systems ON or OFF. The distributed systems are identified by the PROPCHK statements in the routing table of the host programmable operator. The programmable operator facility with the PROPCHK statement (for example, the host system) does the checking.

Use SET HOSTCHK to set the periodic checking of the link to the host system ON or OFF. HOSTCHK must be specified in the routing table of the programmable operator at the distributed system. The programmable operator facility with the HOSTCHK statement (for example, the distributed system) does the checking.

Use SET LOGGING to control messages going to the programmable operator log file. SET LOGGING lets the message sender control the logging level: no logging, logging incoming messages and special programmable operator messages, or incoming messages plus response messages.

Use SET OUTID to specify that each message generated by a virtual machine using the CP MESSAGE command be appended with a message prefix. The message prefix will be used by NetView to package any command output and route it to the proper administrator. This option is useful when administrators are trying to debug a server across systems, since all of the messages from the server are grouped together and sent to the administrator at the same time.

Operands

DEBUG

SET DEBUG ON stops the programmable operator facility from intercepting responses to CP commands. SET DEBUG OFF allows the programmable operator facility to return to its normal function of intercepting messages and responses from CP. SET DEBUG OFF is the initial setting.

HOSTCHK

SET HOSTCHK OFF halts checking of the host system by the distributed

system. SET HOSTCHK ON restarts the checking. The initial setting is determined by the existence of a HOSTCHK statement in the RTABLE (ON if it exists, OFF if it does not).

PROPCHK

SET PROPCHK OFF halts checking of the programmable operators on the distributed systems until this command is reissued to set the checking back on, or until the programmable operator is stopped and restarted, or the PROP LOADTBL command is issued. SET PROPCHK ON restarts the checking. If *nodeid* is specified, PROPCHK applies to the specified node only. The initial setting is determined by the existence of the PROPCHK statement(s) in the RTABLE (ON if they exist, OFF if they do not).

An error message is received if the SET PROPCHK command is given and the routing table does not contain a PROPCHK statement or if the node ID specified in the SET PROPCHK command is not found in a PROPCHK statement in the routing table.

LOGGING

SET LOGGING OFF causes the programmable operator facility to stop writing any messages to the log file. SET LOGGING ON or ALL causes logging to be resumed. SET LOGGING ALL causes logging of all programmable operator command responses, such as virtual machine console I/O generated by action routines. Specifying UPCASE causes the incoming messages to be logged in uppercase while specifying LOWCASE means the incoming messages are to be logged with no change to the original message text. UPCASE is the default.

The LOGGING statement in the configuration portion of the routing table determines the initial setting. If no LOGGING statement appears in the routing table, the default is ON and UPCASE.

OUTID

SET OUTID ON causes a message prefix to be appended to any command output from *userid*. If *msgprefix* is specified, then it is used as the message prefix; otherwise, the *userid* is used as the message prefix. SET OUTID OFF stops the appending of the message prefix.

Usage Notes

1. The SET DEBUG command is only valid from the console of the programmable operator facility virtual machine.
2. The SET DEBUG ON command permits an authorized user to stop the programmable operator facility from intercepting messages associated with CP commands entered from the console of the programmable operator facility virtual machine. The programmable operator facility responds to CP messages (MSG), warnings (WNG), and special messages (SMSG), and messages sent using the Single Console Image Facility (SCIF), but not to responses from CP.
3. When SET DEBUG OFF is in effect, all responses to CP commands are intercepted by the programmable operator facility. SET DEBUG OFF is in effect when the programmable operator facility is initialized.
4. If the logical operator is on a noncheckable node (for example an NCCF or NetView operator) when he turns HOSTCHK ON, the issuer of the SET command receives an error message.

Examples

The following is a list of examples that show how the programmable operator SET command could be issued from a programmable operator virtual machine console:

SET

```
set debug on  
set debug off
```

The following is a list of examples that show how the programmable operator SET command could be issued from a virtual machine console:

```
m op set hostchk off  
m op set propchk on sys2  
m op set logging off  
m op set logging on upcase  
m op set outid on roverk9 xxx  
m op set outid off roverk9
```

The following is a list of examples that show how the programmable operator SET command could be issued from an NCCF or NetView operator terminal:

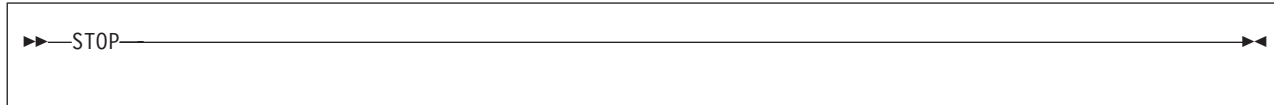
```
prop set hostchk off  
prop set propchk on sys2  
prop set logging off
```

Responses

The programmable operator facility responds with:

- For commands sent by message:
 - Command complete
 - Output identifier set {ON|OFF} successfully.
 - Authorization failure: *userid* set output identifier for user *userid* to *msgprefix*.
- For commands issued at the programmable operator console:
 - {PROPCHK|HOSTCHK} has been started
 - {PROPCHK|HOSTCHK} has been stopped
 - Logging has been started
 - Logging has been stopped
 - PROP running in DEBUG mode
 - PROP has exited DEBUG mode

STOP



Purpose

Use the STOP command to stop the operation of the programmable operator facility. When the STOP command is issued, the programmable operator facility processes all outstanding messages, closes files, stops operation, and returns control to CMS. The STOP command is logged in the log file for the current day.

Usage Notes

1. The STOP command is also valid from the console of the programmable operator facility virtual machine if an operator is logged on to the programmable operator facility virtual machine.

Responses

The programmable operator facility responds with:
PROP has terminated

STOP

Chapter 7. Managing the CMS Batch Facility

The CMS batch facility is a programming facility that runs under the CMS subsystem. The CMS batch facility runs in XC and XA virtual machines in z/VM. Any batch job submitted, which relies on 370-mode instructions, must first ensure that the CP 370 Accommodation Facility is enabled through the BATPROF EXEC, BATEXIT1 exit routine, or from within the batch job itself.

The CMS Batch Facility

The CMS batch facility allows VM users to run their jobs in batch mode by sending jobs either from their virtual machines or through the real (system) card reader to a virtual machine dedicated to running batch jobs. The CMS batch facility then executes these jobs freeing user machines for other uses.

If both the CMS batch facility and the Remote Spooling Communications Subsystem (RSCS) Networking Version 2 (or later version) are being executed under the same VM system, job input streams can be transmitted to the batch facility from remote stations by communication lines. Also, the output of the batch processing can be transmitted back to the remote station.

The CMS batch facility virtual machine is generated and controlled on a user ID dedicated to execution of jobs in batch mode. The system operator generates the *batch machine* either by:

- Entering the BATCH parameter in the PARM field of the IPL command, or
- Specifying the NOSPROF parameter of the IPL command and entering the CMSBATCH command when the VM READ status appears.

After each job is executed, the batch facility IPLs itself, thereby providing a continuously processing batch machine. The batch processor IPLs itself by using the PARM option of the CP IPL command followed by a character string that CMS recognizes as peculiar to a batch virtual machine performing its IPL. Jobs are sent to the batch machine's virtual card reader from users' terminals and executed sequentially. When there are no jobs waiting for execution, the CMS batch facility remains in a wait state ready to execute a user job.

The CMS batch facility is particularly useful for compute-bound jobs such as assemblies and compilations and for execution of large user programs, since interactive users can continue working at their terminals while their time-consuming jobs are run in another virtual machine.

The system programmer controls the batch facility virtual machine environment by resetting the CMS batch facility machine's system limits, by writing routines that handle special installation input to the batch facility, and by writing exec procedures that make the CMS batch facility easier to use.

Using the CMS Batch Facility

The batch facility is a z/VM programming facility that runs under CMS. It lets a z/VM user run jobs in batch mode by sending jobs from either his or her own virtual machine or the real card reader to a virtual machine dedicated to running batch jobs under the batch facility. This dedicated machine is generally set up at a terminal in the installation's computer room and controlled by the system operator.

Managing Batch

The batch facility virtual machine runs continuously, executing all jobs spooled to its virtual card reader from other virtual machines or from the real card reader. The batch operator need pay no attention to the batch machine after he or she has started and disconnected it.

Starting the Batch Virtual Machine

The system operator starts the batch virtual machine by logging on with a batch user ID and loading CMS using the CP IPL command.

Batch User ID

Every installation in which the CMS batch facility is available should establish one or more user IDs for the CMS batch virtual machines. You can then spool your files for execution to the card reader for that batch user ID.

It is the operator's responsibility to log on the z/VM system using the batch user ID established for his or her installation.

The user ID established for the batch facility virtual machine must have a read/write disk in its directory at virtual address 195. The 195 disk is erased when it is accessed as file mode A at the beginning of each job.

Invoking the Batch Facility

The batch facility virtual machine is invoked by the batch operator when he or she enters the CP IPL command with PARM BATCH or CP IPL followed by the CMSBATCH command. The latter method follows:

```
ipl cms parm nosprof
```

The system responds with:

```
z/VM V4.3.0 2002-05-31 15:55
```

Then enter:

```
cmsbatch
```

The system responds with:

```
Y/S (19E) R/O  
The following names are undefined:  
  BATEXIT1 BATEXIT2  
Ready; T=0.14/0.39 08:47:40  
Waiting for reader
```

The operator can now disconnect the batch machine terminal, if he or she wishes, using the #CP DISCONN command. The batch facility will IPL itself after each job is executed.

All virtual machine (CMS) console output is automatically spooled to a file to be printed after the program output at the real system printer. All commands entered through the virtual reader are displayed on the console to let them appear in the console output file. If the batch terminal is disconnected, only CP and batch initialization messages are displayed at the terminal.

If an installation wishes to use a saved system in running batch jobs, the operator must enter the name of the saved system in the CMSBATCH command line.

CMS Batch Output

The batch virtual machine spools output resulting from program execution to the system printer. Output is printed under the submitting user ID, with the submitting user ID's distribution code, a spool file name of CMSBATCH, and a spool file type of JOB (unless a job name was specified on the /JOB card).

The console output is always spooled. Therefore, if the console is disconnected, the CMS console output is spooled to a file that is printed following your program execution output at the real system printer, with the submitting user ID's distribution code, a spool file name of BATCH, and a spool file type of CONSOLE.

If the CP TAG command identifies spool files or directs these files to other virtual machines or remote work stations, BATCH resets the spooling devices for the next job.

A more complete description regarding control of the CMS Batch virtual machine is contained in the *z/VM: CMS Application Development Guide*. It describes the user control cards and suggests control techniques as well as how to control the batch machine using exec procedures.

Purging, Reordering, and Restarting Batch Jobs

When required, the spooling operator can control the execution of batch virtual machine jobs by purging, reordering, and restarting them; by the same token, because all the closed printer files are queued for system output under the submitting user ID, the submitting user can change, purge, or reorder these files before processing on the system printer.

To purge a job executing under the batch monitor, follow the procedure below:

1. Signal attention and enter the virtual machine environment.
2. Enter the HX (halt execution) immediate command.
3. Disconnect the virtual machine using the #CP DISCONN command.

The HX command causes the batch facility to abnormally terminate. This provides you with an error message and a CP dump of the batch facility virtual machine. The batch monitor then loads itself again and starts the next job (if any).

To purge an individual input spool file that is not yet executing, use the CP PURGE command:

```
# purge reader spoolid
```

spoolid is the spool file number of the job to be purged from the batch virtual machine's job queue. For example, the statement

```
# purge reader 123
```

would purge 123 from the batch virtual machine's job queue.

To reorder individual spool files in the job queue of the batch facility, use the CP ORDER command:

```
# order reader spoolid1 spoolid2...
```

spoolid1 and *spoolid2* are the assigned spool file identification of the jobs to be reordered.

Managing Batch

The operator can determine which jobs are in the queue by using the CP QUERY command:

```
# query reader all
```

This QUERY command lists the file names and file types of all the jobs in the batch virtual machine's job queue. The operator can then reorder them, using the ORDER command.

Stopping the Batch Virtual Machine

To stop batch virtual machine execution after completion of the current job, enter the HB immediate command and press the attention key or equivalent to cause an attention interruption at anytime during the job. This causes the batch virtual machine to be logged off at job completion.

When batch facility virtual machine execution is to be immediately stopped, but current files must be saved, you can use the #CP SPOOL command, in the form SPOOL READER HOLD, and then enter the #CP LOGOFF command. When you enter the SPOOL READER HOLD command, you may receive message HCPCSP424I. Ignore this message.

The HOLD option causes CP to retain the virtual machine's current card reader file, so that when the batch machine is logged on again, execution resumes at the beginning of the held reader file.

If an emergency should occur, all jobs in the batch reader and all spool files are saved.

Installing the CMS Batch Machine

Figure 18 shows the recommended CP directory entry for a virtual machine intended to be IPLed as a CMS batch machine. Consult the *z/VM: CP Planning and Administration* for the proper coding of the directory macro parameters.

```
USER CMSBATCH password 32M 32M X
ACCOUNT 13 SYSTEM
OPTION ACCT
IPL CMS PARM NOSPROF
CONSOLE 009 3215
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK MAINT 190 190 RR
MDISK 195 devtype cyl cyls volser
```

Figure 18. CP Directory entry for CMS Batch Machine

In place of the class G and class ANY commands available to all general users, the example above shows the CMSBATCH machine being authorized for privilege class X. Note that this class is not a standard class and must be created using the user class restructure feature. Privilege classes have been designed to allow the CMSBATCH machine to use all class G commands except the CP TRANSFER command. You should restrict the use of the CP TRANSFER command so that output created by CMSBATCH cannot be rerouted.

Note: There is no 191 MDISK for the CMS batch machine.

To have the CMS batch machine automatically logged on, you should have the following entry in the autolog virtual machine's (AUTOLOG1) PROFILE EXEC:

```
AUTOLOG CMSBATCH password CMSBATCH
```

Otherwise, the operator logs on to the CMS batch machine and at the Ready message follows the procedure described in “Invoking the Batch Facility” on page 132. Refer to the *z/VM: CP Commands and Utilities Reference* for more information on the AUTOLOG command.

Note: If your system has password suppression, the correct entry to have in the autolog virtual machine's PROFILE EXEC would be:

```
AUTOLOG CMSBATCH CMSBATCH
```

The ACCT option of the OPTION directory statement must be specified to create an accounting card. This accounting card assures that the proper user's account is charged for use of the CMSBATCH machine's time, plus, it transfers the output from the CMSBATCH machine back to the user who sent the job.

Resetting the CMS Batch Facility System Limits

Each job running under the CMS batch facility is limited by default to the maximum value of 32,767 seconds of virtual CPU time, 32,767 punched cards output, and 32,767 printed lines of output. You can reduce these limits by modifying the BATLIMIT MACRO file, which is found in the DMSGPI macro library, and then reassembling DMSBTP, the batch processor source file.

Exec Procedures for the Batch Facility Virtual Machine

You can control the CMS batch facility virtual machine using exec procedures. For example, you can use an exec:

- To produce the proper sequence of CP/CMS commands for users who do not know CMS commands and controls.
- To provide the sequence of commands needed to execute the most common jobs (assemblies and compilations) in a particular installation.

For information on how to use execs to control the batch facility virtual machine, see the *z/VM: CMS Application Development Guide*.

Data Security under the Batch Facility

After each job, the CMS batch facility loads (by IPL) itself destroying all nucleus data and work areas. All disks where links were established (using a CP LINK in the job stream) during the previous job are detached.

At the beginning of each job, the batch facility work disk is accessed and then immediately erased preventing the current user job from accessing files that might remain from the previous job. Because of this, execution of the PROFILE EXEC is disabled for the CMS batch facility machine. You may, however, create an exec procedure called BATPROF EXEC and store it on any system disk to be used instead of the ordinary PROFILE EXEC. The batch facility then executes this exec at each job initialization time.

To prevent the CMSBATCH machine from being able to reroute the output of jobs submitted by previous users, use of the CP TRANSFER command by the CMSBATCH machine should be restricted. To do this, you should redefine the

Managing Batch

privilege class of the CMSBATCH machine to a privilege class other than class G. The assigned class should allow the use of all of the general user commands except the TRANSFER command.

Improved IPL Performance Using a Saved System

Since the CMS batch processor goes through an IPL procedure after each user job, an installation may experience a more efficient IPL procedure by using a saved CMS system when processing batch jobs.

This can be accomplished by passing the name of the saved system to the CMS batch facility by the optional *sysname* operand in the CMSBATCH command line.

The batch facility saves the name of the saved system until the end of the first job. At this time it stores the name in the IPL command line both as the device address and as the PARM character string. The latter entry informs the CMS initialization routine that a saved system has been loaded and that the name is to be saved for subsequent IPL procedures.

Customizing the CMS Batch Facility

PI

The CMS batch facility provides installation-wide exits for:

- Processing a non-CMS control language
- Installation-specified processing of /JOB control cards.

You must write the routines that make use of these exits. The exit routines must be named BATEXIT1 and BATEXIT2, respectively. Each routine must have a file type of TEXT and a file mode number of 2 if placed on the system disk or an extension of the system disk. Your routines are responsible for saving registers, including general purpose register 12, which saves addressability for the batch facility. If you place your routines on the system disk, they are included with the CMS batch facility each time it is loaded.

BATEXIT1: Processing a Non-CMS Control Language

The BATEXIT1 entry point allows you to write a routine to check batch facility control cards for non-CMS control statements. For example, you can write a routine to scan for the OS job control language needed to compile, link edit, and execute a FORTRAN job. BATEXIT1 receives control after each read from the CMS batch facility virtual card reader is issued. General purpose register 1 contains the address of the batch facility read buffer, which contains the 80-byte card image to be executed by the batch facility. This enables the BATEXIT1 routine to scan each card it receives as input for the type of control information you specify.

After processing the card, the BATEXIT1 routine must place a return code in general purpose register 15 and return to CMS by branching to the address in register 14. If the return code from the BATEXIT1 routine is nonzero, the batch facility stops processing the card and reads the next card. If the return code from the BATEXIT1 routine is zero, the batch facility continues processing by passing the card to CMS for execution.

BATEXIT2: Special Processing of /JOB Control Cards

The BATEXIT2 entry point allows you to write a routine to do special processing of batch facility /JOB control cards. BATEXIT2 receives control after CMS has scanned the /JOB card and built the parameter list, but before the batch routine used to process the /JOB card begins its processing. General purpose register 1 points to the CMS parameter list buffer. This buffer consists of the following 8-byte fields containing data from the /JOB card:

1. /JOB
2. *userid*
3. *acctnum*
4. *jobname* (optional)

Your routine can redefine the contents of fields 2, 3 and 4.

After processing the /JOB card, the BATEXIT2 routine must place a return code in register 15 and return to CMS by branching to the address in register 14. If the return code from the BATEXIT2 routine is nonzero, the batch facility issues CMS error message 106E and flushes the job. If the return code from the BATEXIT2 routine is zero, the batch facility assumes that register 1 points to the /JOB control card information. The batch facility then does its normal checking of the /JOB control card information for a valid user ID and the existence of an account number and optional job name. Finally, execution of the job begins.

PI end

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in all 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
Armonk, New York 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, New York 12601-5400
U.S.A.
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information 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 the names and addresses used by an actual business enterprise 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.

Programming Interface Information

This book primarily documents information that is NOT intended to be used as Programming Interfaces of z/VM.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/VM. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

PI

<...Programming Interface information...>

PI end

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml

Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Glossary

For a list of z/VM terms and their definitions, see *z/VM: Glossary*.

The z/VM glossary is also available through the online z/VM HELP Facility. For example, to display the definition of the term “dedicated device”, issue the following HELP command:

```
help glossary dedicated device
```

While you are in the glossary help file, you can do additional searches:

- To display the definition of a new term, type a new HELP command on the command line:

```
help glossary newterm
```

This command opens a new help file inside the previous help file. You can repeat this process many times. The status area in the lower right corner of the screen shows how many help files you have open. To close the current file, press the Quit key (PF3/F3). To exit from the HELP Facility, press the Return key (PF4/F4).

- To search for a word, phrase, or character string, type it on the command line and press the Clocate key (PF5/F5). To find other occurrences, press the key multiple times.

The Clocate function searches from the current location to the end of the file. It does not wrap. To search the whole file, press the Top key (PF2/F2) to go to the top of the file before using Clocate.

Bibliography

See the following publications for additional information about z/VM. For abstracts of the z/VM publications, see *z/VM: General Information*.

Where to Get z/VM Information

z/VM product information is available from the following sources:

- z/VM Information Center at publib.boulder.ibm.com/infocenter/zvm/v6r1/index.jsp
- z/VM Internet Library at www.ibm.com/eserver/zseries/zvm/library/
- IBM Publications Center at www.elink.ibmink.ibm.com/publications/servlet/pbi.wss
- *IBM Online Library: z/VM Collection on DVD*, SK5T-7054

z/VM Base Library

Overview

- *z/VM: General Information*, GC24-6193
- *z/VM: Glossary*, GC24-6195
- *z/VM: License Information*, GC24-6200

Installation, Migration, and Service

- *z/VM: Guide for Automated Installation and Service*, GC24-6197
- *z/VM: Migration Guide*, GC24-6201
- *z/VM: Service Guide*, GC24-6232
- *z/VM: VMSES/E Introduction and Reference*, GC24-6243

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6167
- *z/VM: CMS Planning and Administration*, SC24-6171
- *z/VM: Connectivity*, SC24-6174
- *z/VM: CP Planning and Administration*, SC24-6178
- *z/VM: Getting Started with Linux on System z*, SC24-6194
- *z/VM: Group Control System*, SC24-6196

- *z/VM: I/O Configuration*, SC24-6198
- *z/VM: Running Guest Operating Systems*, SC24-6228
- *z/VM: Saved Segments Planning and Administration*, SC24-6229
- *z/VM: Secure Configuration Guide*, SC24-6230
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6236
- *z/VM: TCP/IP Planning and Customization*, SC24-6238
- *z/OS and z/VM: Hardware Configuration Manager User's Guide*, SC33-7989

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6176
- *z/VM: Performance*, SC24-6208

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6166
- *z/VM: CMS Pipelines Reference*, SC24-6169
- *z/VM: CMS Pipelines User's Guide*, SC24-6170
- *z/VM: CMS Primer*, SC24-6172
- *z/VM: CMS User's Guide*, SC24-6173
- *z/VM: CP Commands and Utilities Reference*, SC24-6175
- *z/VM: System Operation*, SC24-6233
- *z/VM: TCP/IP User's Guide*, SC24-6240
- *z/VM: Virtual Machine Operation*, SC24-6241
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6244
- *z/VM: XEDIT User's Guide*, SC24-6245
- *CMS/TSO Pipelines: Author's Edition*, SL26-0018

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6162
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6163
- *z/VM: CMS Application Multitasking*, SC24-6164
- *z/VM: CMS Callable Services Reference*, SC24-6165
- *z/VM: CMS Macros and Functions Reference*, SC24-6168

- *z/VM: CP Programming Services*, SC24-6179
- *z/VM: CPI Communications User's Guide*, SC24-6180
- *z/VM: Enterprise Systems Architecture/Extended Configuration Principles of Operation*, SC24-6192
- *z/VM: Language Environment User's Guide*, SC24-6199
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6202
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6203
- *z/VM: OpenExtensions Commands Reference*, SC24-6204
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6205
- *z/VM: OpenExtensions User's Guide*, SC24-6206
- *z/VM: Program Management Binder for CMS*, SC24-6211
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6220
- *z/VM: REXX/VM Reference*, SC24-6221
- *z/VM: REXX/VM User's Guide*, SC24-6222
- *z/VM: Systems Management Application Programming*, SC24-6234
- *z/VM: TCP/IP Programmer's Reference*, SC24-6239
- *Common Programming Interface Communications Reference*, SC26-4399
- *Common Programming Interface Resource Recovery Reference*, SC31-6821
- *z/OS: IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148
- *z/OS: Language Environment Concepts Guide*, SA22-7567
- *z/OS: Language Environment Debugging Guide*, GA22-7560
- *z/OS: Language Environment Programming Guide*, SA22-7561
- *z/OS: Language Environment Programming Reference*, SA22-7562
- *z/OS: Language Environment Run-Time Messages*, SA22-7566
- *z/OS: Language Environment Writing ILC Applications*, SA22-7563
- *z/OS MVS Program Management: Advanced Facilities*, SA22-7644
- *z/OS MVS Program Management: User's Guide and Reference*, SA22-7643

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6161
- *z/VM: CP Messages and Codes*, GC24-6177
- *z/VM: Diagnosis Guide*, GC24-6187
- *z/VM: Dump Viewing Facility*, GC24-6191
- *z/VM: Other Components Messages and Codes*, GC24-6207
- *z/VM: TCP/IP Diagnosis Guide*, GC24-6235
- *z/VM: TCP/IP Messages and Codes*, GC24-6237
- *z/VM: VM Dump Tool*, GC24-6242
- *z/OS and z/VM: Hardware Configuration Definition Messages*, SC33-7986

z/VM Facilities and Features

Data Facility Storage Management Subsystem for VM

- *z/VM: DFSMS/VM Customization*, SC24-6181
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6182
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6183
- *z/VM: DFSMS/VM Planning Guide*, SC24-6184
- *z/VM: DFSMS/VM Removable Media Services*, SC24-6185
- *z/VM: DFSMS/VM Storage Administration*, SC24-6186

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6188
- *z/VM: Directory Maintenance Facility Messages*, GC24-6189
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6190

Open Systems Adapter/Support Facility

- *System z10, System z9 and eServer zSeries: Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935
- *System z9 and eServer zSeries 890 and 990: Open Systems Adapter-Express Integrated Console Controller User's Guide*, SA22-7990

- *System z: Open Systems Adapter-Express Integrated Console Controller 3215 Support*, SA23-2247

Performance Toolkit for VM™

- *z/VM: Performance Toolkit Guide*, SC24-6209
- *z/VM: Performance Toolkit Reference*, SC24-6210

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6212
- *z/VM: RACF Security Server Command Language Reference*, SC24-6213
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6214
- *z/VM: RACF Security Server General User's Guide*, SC24-6215
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6216
- *z/VM: RACF Security Server Messages and Codes*, GC24-6217
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6218
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6219
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6231

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6223
- *z/VM: RSCS Networking Exit Customization*, SC24-6224
- *z/VM: RSCS Networking Messages and Codes*, GC24-6225
- *z/VM: RSCS Networking Operation and Use*, SC24-6226
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6227
- *Network Job Entry: Formats and Protocols*, SA22-7539

Prerequisite Products

Device Support Facilities

- *Device Support Facilities: User's Guide and Reference*, GC35-0033

Environmental Record Editing and Printing Program

- *Environmental Record Editing and Printing Program (EREP): Reference*, GC35-0152
- *Environmental Record Editing and Printing Program (EREP): User's Guide*, GC35-0151

Index

Special characters

*NCCF 80, 90, 91

Numerics

3350 DASD
 capacity for CMS minidisks 14
3375 DASD
 capacity for CMS minidisks 14
3380 DASD
 capacity for CMS minidisks 14
3390 DASD
 capacity for CMS minidisks 14
9345 DASD
 capacity for CMS minidisks 14

A

A-disk
 accessing 9
 CMS primary user disk 3
ABENDs
 programmable operator facility 101
abnormal termination (abend)
 programmable operator facility 61, 101
ACCESS command in CMS 9
access file modes in CMS files 9
access module, creating 54
access module, preparing 53, 55
accessing the message repository for SQL/DS 54
accessing the SQL/DS production minidisk 54
action routines 59, 100
 call interface 101
 DMSPOL 109
 DMSPOR 106
 GET 106
 LGLOPR 106
 QUERY 106
 SET 106
 STOP 107
 TOFB 107
 TOVM 107
 DMSPOS 107
 error message and response handling 105
 exec 104
 handling console I/O 105
 interface 101
 parameter interface 102
 writing 104
Ada program language 7
administration of CMS Pipelines 51, 55
 CMS Pipelines Repositories 51
 configuring capabilities 51
alternate CMS nucleus placement 41
AMS (Access Method Services)
 DASD devices supported 22
 DOS VSAM data set support 21

AMS (Access Method Services) (*continued*)
 OS data set support 21
 SAM data set support 21
 supported under CMS 21
APL2 program language 7
Assembler (High Level Assembler, Assembler H, and
 Assembler F) 7
assign
 logical operator 72, 106
automatic
 invocation of the programmable operator 63
auxiliary storage required by CMS 2

B

batch facility
 BATLIMIT MACRO file 135
 data security 135
 description 131
 exec procedures 135
 installing 134
 invoking 132
 IPL performance 136
 resetting system limits 135
 system limits 135
 using 131
batch jobs
 purging 133
 reordering 133
 restarting 133
batch virtual machine
 starting 132
 stopping 134
BATEXIT1 exit routine 136
BATEXIT2 exit routine 136
BATLIMIT macro 135

C

C program language 7
Callable Services Library 6
capacity, DASD
 CMS 14
 minidisks 14
changing a logical operator 72
 LGLOPR command 72
CKD (Count-Key-Data)
 devices
 CMS block 10
CMD command, programmable operator 114
CMS
 alternate nucleus placement 41
CMS (Conversational Monitor System)
 batch facility
 BATEXIT1 exit routine 136
 BATEXIT2 exit routine 136

- CMS (Conversational Monitor System) *(continued)*
 - batch facility *(continued)*
 - job control card, installation-specified processing 137
 - non-CMS control statements, processing 136
 - installation saved segment 37
 - multitasking 6
- CMS multitasking with CMS Pipelines 51
- CMS nucleus
 - adding segment for Y minidisk directory (Y-STAT) 41
 - alternate placement 41
 - increasing size 41
 - rebuilding 45
- CMS Pipelines
 - administration 51, 55
 - CMS multitasking considerations 51
 - compiling internals 51
 - configuring capabilities 51
 - module, rebuilding 55
 - repositories 51
 - shared segment 51
- CMS/DOS
 - ASSGN command 24, 25
 - CMSBAM saved segment 24
 - CMSDOS saved segment 24
 - disk label information area 25
 - DLBL command 24, 25
 - librarian programs 25
 - planning considerations 24
 - SET DOS ON command 24, 25
 - tape handling 25
 - VSE compilers 24
 - VSE sysres 24
 - VSE system and private libraries 24
 - VSE system generation considerations 24
 - when VSE system must be online 24
- CMSBAM DOSLIB 6
- CMSLIB MACLIB 5
- CMSLIB TXTLIB 6
- CMSSAA TXTLIB 6
- COBOL (OS/VS COBOL and VS COBOL II) 7
- commands
 - See also* CMS commands and CP commands
 - IPL CMS command 27
 - programmable operator 114
 - CMD 114
 - FEEDBACK 116
 - GET 117
 - LGLOPR 118
 - LOG 122
 - QUERY 123
 - SET 126
 - STOP 129
 - SAVEFD 40
- communication
 - between the programmable operator and the network 61
 - in a mixed environment 99
 - IUCV 65
 - programmable operator and NCCF or NetView 69
 - communication *(continued)*
 - programmable operator with IUCV 65
- configurations supported by CMS 12
- configuring a CMS Pipelines logon session 51
- control block mappings 102
- Control Program (CP)
 - device simulation 3
 - disk access 9
- controlling authorization 87
- Conversational Monitor System (CMS)
 - A-disk 3
 - assembler used with CMS 7
 - auxiliary storage requirements 2
 - batch facility 131
 - capacity of virtual disks 11
 - command language 1
 - DASD storage
 - BFS 14
 - SFS 14
 - default device addresses 2
 - devices supported 2
 - disk and file management 8
 - CMS 8
 - OS/DOS 8
 - VSAM 8
 - disks 3
 - access 9
 - capacity 11
 - file format 10
 - formatting 10
 - labels 10
 - linking 9
 - FB-512 blocks 10
 - file directory 11
 - File Pools
 - file pool server processing 15
 - top directory 13
 - files
 - access modes 9
 - format 10
 - identification 11
 - sharing 9
 - using SFS 9
 - FORMAT command 10
 - formatted disks volume label contents 10
 - immediate commands 107
 - libraries 4
 - limited support of OS and VSE 7
 - master file directory 11
 - nucleus storage requirements 2
 - partitioned data sets 4
 - planning considerations 1
 - program language facilities 7
 - program languages supported under CMS 7
 - records, maximum usable number per file 11
 - restrictions 16
 - Shared File System (SFS)
 - disk and file management 9
 - file pool 13, 15, 21
 - file pool server machine 13
 - file pool space 2

Conversational Monitor System (CMS) *(continued)*

Shared File System (SFS) *(continued)*

- file space 13
 - top directory 3
 - shared segments 12
 - simulated partitioned data sets 4
 - storage requirements 2
 - support of DL/I 8
 - symbolic names for devices 2
 - system disk (S-disk) 3
 - system libraries
 - macroinstruction 5
 - text 6
 - tailoring 27
 - tape support 11
 - unit record support 3, 11
 - virtual storage requirements 2
- Coordinated Resource Recovery (CRR)
- DASD storage requirements 15
- create
- access module 54

D

- DASD storage
- space needed by CMS 2
 - storage
 - for CMS minidisks 14
- data
- area mappings 102
 - security in batch facility 135
- data compression
- VSAM files 21, 23
- database name for SQL/DS 54
- debug mode, programmable operator 111
- default logical operator 59
- DEFINE command 9
- DEFNUC macro 35
- Device Support Facility
- formatting with 8
 - initializing with 21
- devices
- default addresses for CMS 2
 - supported by CMS 2
 - supported by CMS VSAM 22
- direct access storage device (DASD)
- space needed by CMS 2
 - storage
 - for CMS minidisks 14
- directory
- CMS file directory 11
 - Shared File System 4, 8
- disk access 9
- disk and file management within an SFS directory 8
- disk for SQL/DS production 54
- Disk Operating System (DOS)
- assembling VSE programs under CMS 5
 - macroinstruction library for CMS 5
 - support under CMS 7
- disk space
- space needed by CMS 2

disk space *(continued)*

- storage
 - for CMS minidisks 14
- disks 8
- CMS, access 9
 - formatting for CMS 10
 - labels, CMS 10
 - management
 - CMS 8
 - OS/DOS 8
 - VSAM 8
- DL/I support in CMS/DOS environment 8
- DMSAENV TXTLIB 6
- DMSAMT TXTLIB 6
- DMSBASE TXTLIB 6
- DMSCENV TXTLIB 6
- DMSCMT TXTLIB 6
- DMSGPI MACLIB 5
- DMSNGP text file 36
- DMSOM MACLIB 5
- DMSPIPE user ID 54
- DMSPOL programmable operator action routine 109
- DMSPOR programmable operator action routine 106
- DMSPOS programmable operator action routine 107
- DMSPQI ASMSQL file 54
- DMSPQI ASSEMBLE file 54
- DMSSP MACLIB 5
- DMSTOEAG module, invoking 29
- DOS (Disk Operating System)
- assembling VSE programs under CMS 5
 - macroinstruction library for CMS 5
 - support under CMS 7

E

- ensuring a complete log
- in a distributed system 92
 - in a mixed environment 93
 - in a single system 92
- establishing a SQL/DS database name 54
- establishing communication
- programmable operator and NCCF or NetView 69
 - programmable operator with IUCV 65
- estimating
- DASD storage requirements for CMS 13
 - DASD storage requirements for CMS minidisks 14
 - DASD storage requirements for file pools 15
- exec action routines 104
- exec procedures for CMS batch 135
- exec sharing 40
- exec, exit 59
- execution-time libraries 6
- exit exec 59
- exit execs, programmable operator 109
- interface 109
- exit EXECs, programmable operator
- communication error 110
 - PROPPCHK 110
- exits
- BATEXIT1 routine 136
 - BATEXIT2 routine 136

exits (*continued*)

CMS

- batch facility 136
 - BATEXIT1 routine 136
 - BATEXIT2 routine 136
- installation-wide
- BATEXIT1 routine 136
 - BATEXIT2 routine 136

F

FB-512

- capacity for CMS minidisks 14
- CMS block 10
- minidisk space allocation 14

FEEDBACK command, programmable operator 116

feedback file 93

file

pool

- planning 13
- server machine 13

sharing 9

file directory

- information 40
- updating 11

FILEDEF command

- to invoke the programmable operator 63

files

CMS

- file modes 9
- maximum number of records 11
- Shared File System (SFS) 9
- sharing 9

management

- CMS 8
- OS/DOS 8
- VSAM 8

filtering messages 86

FORMAT

- command (CMS)
- as used during disk formatting 10

FPLGPI MACLIB 5

FPLOM MACLIB 5

free storage required by CMS 2

G

GET

- command, programmable operator 117

GET action subroutine 106

granting authority to an access module 55

Group Control System (GCS)

- and the programmable operator facility 58, 68

H

hardware devices supported by VSE 22

HCPGPI MACLIB 5

HCPPSI MACLIB 5

HOSTCHK statement 76

I

IBM-supplied user message repository, CMS

Pipelines 52

identification of CMS files 11

identifying disk files 11

immediate commands in CMS 107

initialize

- programmable operator 64

install

- CMS batch machine 134

PMX 68

- programmable operator 62

internal message table, CMS Pipelines 52

invoking

- CMS batch facility 132

invoking

- commands 113

issuing commands 113

programmable operator facility 62

- automatically 63

- commands 95, 113

- manually 62

invoking DMSTOEAG 29

IPL CMS command

- BATCH parameter 131

description of 28

NOSPROF parameter 131

SAVESYS parameter 29

IPL command 27

IPL directory control statement 33, 107

IPL performance using saved system 136

IPL z/VM system directory control statement 28, 63

ISAM Interface Program (IIP) 22

IUCV (Inter-User Communication Vehicle)

- communication 65

- programmable operator 65

IXXOM MACLIB 5

L

languages supported by CMS 7

LGLOPR

- action subroutine 106

command 106

statement 74

libraries

Callable Services Library 6

CMSBAM DOSLIB 6

execution-time libraries 6

PROPLIB LOADLIB 6

System Macroinstruction Library 5

CMSLIB MACLIB 5

DMSGPI MACLIB 5

DMSOM MACLIB 5

DMSSP MACLIB 5

FPLGPI MACLIB 5

FPLOM MACLIB 5

HCPGPI MACLIB 5

HCPPSI MACLIB 5

IXXOM MACLIB 5

libraries (*continued*)

- System Macroinstruction Library (*continued*)
 - MVSXA MACLIB 5
 - OSMACRO MACLIB 5
 - OSMACRO1 MACLIB 5
 - OSVSAM MACLIB 5
- System Text Library 6
 - CMSLIB TXTLIB 6
 - CMSSAA TXTLIB 6
 - DMSAENV TXTLIB 6
 - DMSAMT TXTLIB 6
 - DMSBASE TXTLIB 6
 - DMSCENV TXTLIB 6
 - DMSCMT TXTLIB 6
 - POSXSOCK TXTLIB 6
 - TSOLIB TXTLIB 6
 - VMLIB TXTLIB 6
 - VMMTLIB TXTLIB 6
- VSE system and private libraries 24
- limited support of OS and VSE in CMS 7
- LINK command
 - use in disk access 9
- linking CMS disks 9
- linking the SQL/DS production minidisk 54
- load map
 - printing CMS 48
 - saving CMS 48
- LOADTBL command, programmable operator 120
- LOG command, programmable operator 122
- log file 90
- LOGGING statement 77
- logical operator 58, 70
 - action routine 106
 - assigning, releasing, replacing 72
 - command 72, 106, 118
 - compared to the CP system operator 70
 - default 59, 71
 - LGLOPR
 - command 118
 - NCCF or NetView 71
 - statement 74
- logical records, CMS 11

M

macroinstruction

- CMS macroinstructions 5
- DEFNUC 35
- OS macroinstructions under CMS, simulating 7
- OS/MVS macroinstruction libraries under CMS 5
- VSAM macroinstruction library under CMS 5
- VSE macroinstruction library under CMS 5

manual invocation of the programmable operator 62

master file directory, identifying CMS files 11

message

- comparison text used by programmable operator 79
- exchange (PMX) 58, 68
- format, programmable operator 66
- intercepted by CP 60
- system service 65
- to programmable operator 57

- message examples, notation used in xiii
- message repositories, user, CMS Pipelines 52
- message repository for SQL/DS accessing 54
- message repository, CMS Pipelines 51
- minidisk
 - file directory of 40
 - shared, EDF R/O 40
 - SQL/DS, production 54
- MSGLIMIT statement 77
- multitasking 6
- MVSXA MACLIB 5

N

NCCF 80, 90

NetView

- and the programmable operator facility 68
 - logging NetView messages 91
 - logical operator 71
- operator
 - command authorization 69
 - QUERY command authorization 72
 - station 96
- PMX 58, 68
 - when stopping programmable operator 68

Network Communications Control Facility (NCCF)

- and the programmable operator facility 68
 - logging NCCF messages 91
 - logical operator 71
- operator
 - command authorization 69
 - QUERY command authorization 72
 - routing messages to 86
 - station 96
- PMX 58, 68
 - when stopping programmable operator 68

notation used in message and response examples xiii

notices 139

O

object programs (TEXT files) under CMS 7

operator

- logical 58
- NCCF station 96
- NetView station 96

OS (Operating System)

- macroinstruction libraries for CMS 5
- support under CMS 7

OS PL/I program language 7

OSMACRO MACLIB 5

OSMACRO1 MACLIB 5

OSVSAM MACLIB 5

P

PA1 program function key

- with the programmable operator facility 64, 111

partitioned data set (PDS)

- CMS, limited support 7

- physical
 - record, CMS 10
- planning
 - CMS 1
 - for SFS, BFS and CRR 13
 - VSAM support 21
 - VSE simulation 21
- PMX (Programmable Operator/NCCF Message Exchange) 68
- preparing CMS Pipelines to use SQL/DS 53, 55
- preparing the access module 53, 55
- preprocessor for SQL/DS 54
- printing
 - CMS load map 48
- production minidisk for SQL/DS 54
- program languages supported by CMS 7
- programmable operator facility 61, 75, 87, 97, 109, 110
 - abend 61, 101
 - action routine interface
 - call interface 101
 - parameter interface 102
 - action routines 59
 - DMSPOL 109
 - DMSPOR 106
 - DMSPOS 107
 - error message and response handling 105
 - EXEC 101
 - handling console I/O 105
 - writing 104
- communication
 - with the network 61
- communication error exit exec 95
- debug mode 111
- default 71
- environments
 - distributed z/VM 97
 - mixed 99
- examples
 - in a distributed environment 98
- exit exec
 - communication error 110
 - interface 109
 - overview 59
 - PROPLGER EXEC 91
 - PROPPCHK EXEC 110
- initializing 64
- installing 62
 - PMX 68
- invoking
 - automatically 63
- issuing commands 97, 100
- LGLOPR
 - default 59
 - sample command entries in RTABLE 73
 - statement 59
- log file 90
- logical operator 60
- message format
 - distributed 96
 - local 95

- programmable operator facility (*continued*)
 - overview
 - how it works 59
 - in a distributed z/VM system 57
 - in a mixed environment 58
 - in a single system 57
 - logical operator 58
 - PMX 58
 - stopping 70
 - PROPLIB LOADLIB 62
 - PROPPROF EXEC 65
 - PROPST EXEC 62
 - relationship with RSCS 61
 - routing
 - entries to filter responses to commands 87
 - routing table 74
 - use 74
 - routing table (RTABLE) 59
 - routing table statements
 - TEXTSYM 75
 - starting 64
 - virtual machine 62
 - with NCCF
 - command authorization 69
- Programmable Operator Facility 6, 57, 111
- Programmable Operator/NCCF Message Exchange (PMX) 68
- programming interface information 140
- PROPCHK statement 76
- PROPEPIF EXEC 67
- PROPLGER EXEC 91
- PROPLIB LOADLIB 6, 62, 64
- PROPPCHK EXEC 110
- PROPPROF EXEC 65
- PROPST EXEC 62
- purging the batch facility 133

Q

- QUERY command
 - action subroutine 106
 - command, programmable operator 123

R

- rebuilding
 - CMS nucleus 45
- records, maximum usable number per CMS file 11
- releasing a logical operator 72
 - LGLOPR command 72
- Remote Spooling Communications Subsystem Networking (RSCS)
 - relationship with programmable operator facility 61, 94
- reordering the batch facility 133
- replace
 - a logical operator
 - LGLOPR command 72
- repositories, CMS Pipelines 51
- repository of SQL/DS messages, accessing 54
- response examples, notation used in xiii

- restarting the batch facility 133
- restriction
 - CMS 16
- REXX program language 7
- REXX run-time library, selecting 29
- ROUTE statement 77
- routines, action 59
- routing table (RTABLE) 59, 74
- routing text comparison example 83
- RTABLE (routing table) 59, 74
- running programs
 - programmable operator from NCCF or NetView 68
 - SQL/DS preprocessor 54
 - SQLPREP EXEC 54

S

- S-disk
 - CMS system disk
 - accessing 9
 - System Macroinstruction libraries 5
 - System Text libraries 6
- SAVEFD command 40
- saving
 - CMS load map 48
- selecting REXX run-time library 29
- SET
 - command, programmable operator 126
- SET action subroutine 106
- SET LANGUAGE command for SQL/DS message repository 54
- Shared File System (SFS)
 - directory access 9, 62
 - enrollment 3
 - planning 2
- shared segments, CMS 12
- sharing
 - execs and XEDIT macros 40
 - file directory information 40
- space
 - space needed by CMS 2
 - storage
 - for CMS minidisks 14
- specify
 - routing texts to an NCCF or NetView operator 85
- SQL
 - access module, preparing 53, 55
 - SQL/DS preprocessor 54
 - SQL/DS production minidisk 54
 - SQL/DS, preparing CMS Pipelines for 53, 55
 - SQLINIT command 54
 - SQLPREP EXEC, running 54
- start
 - programmable operator 64
- STOP, programmable operator
 - action subroutine 107
 - command 129
- stopping
 - batch virtual machine 134
 - programmable operator facility 67

- storage, DASD
 - space needed by CMS 2
 - storage
 - for CMS minidisks 14
- symbolic names for CMS devices 2
- syntax diagrams, how to read xi
- SYSPROF EXEC
 - bypassing 33
 - changing 31
 - description of 27
 - functions of 27, 33
 - inputs 31
 - modifying 28
 - output 32
- system
 - definition
 - considerations for VSE 24
 - macroinstruction libraries 5
 - profile 27
 - text libraries 6
- system NETID file 49

T

- table, routing 59
- tailoring
 - programmable operator routing table 82
 - your system 27
- tapes
 - handling, CMS/DOS 25
 - support for CMS 11
- terminating the programmable operator facility 67
- text libraries, CMS 6
- TEXTSYM statement 75
- TOFB action subroutine 107
- TOVM action subroutine 107
- transient area, CMS storage requirements 2
- TSO text library for CMS 6
- TSOLIB TXTLIB 6

U

- unit record device
 - support for CMS 3, 11
- user
 - ID
 - DMSPIPE 54
- user message repositories, CMS Pipelines 52

V

- virtual
 - disks 9
 - storage
 - CMS requirements 2
- virtual machine
 - programmable operator 59, 62
- VMLIB CSLLIB 6
- VMLIB CSLSEG 6
- VMLIB TXTLIB 6
- VMMTLIB logical saved segment 37

- VMMTLIB TEXT 6
- VMMTLIB TXTLIB 6
- volume label contents, CMS formatted disks 10
- VS FORTRAN program language 7
- VS Pascal program language 7
- VSAM (Virtual Storage Access Method)
 - CMS support 21
 - CMS support for OS and VSE users 7
 - DASD devices supported 22
 - data compression 21, 23
 - data set compatibility with CMS 22
 - DOS/VS SORT/MERGE support 21
 - ISAM Interface Program (IIP) 22
 - macroinstruction library for CMS 5
 - master catalog 21
 - minidisks 21
 - OS macroinstruction support 21
 - planning 21
 - planning considerations 23
 - VSE macroinstruction support 21
- VSE (Virtual Storage Extended)
 - compilers 24
 - macroinstruction library for CMS 5
 - planning 21
 - system and private libraries 24
 - system generation considerations 24

X

- XEDIT macro sharing 40

Y

- Y minidisk directory (Y-STAT)
 - adding segment for Y-STAT 41



Program Number: 5741-A07

Printed in USA

SC24-6171-00

