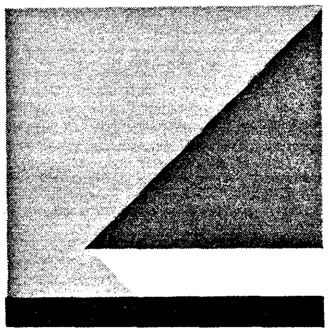
IBM

NetView™

SC31-6015-0

Customization: Writing Command Lists

Release 3



File Number
S370/4300/30XX-50

Program Numbers
5665-362 (MVS/XA)
5664-204 (VM)

Fifth Edition (May 1989)

This edition applies to Release 3 of the NetView™ licensed program, which runs under the following operating systems:

MVS/XA (NetView program number 5665-362)
MVS/ESA (NetView program number 5665-362)
VM (NetView program number 5664-204).

Contents

Figures

Tables

About This Book

NetView Customization: Writing Command Lists describes how to write command lists for the NetView™ program using either the Restructured Extended Executor language (REXX) or the NetView command list language.

This book is intended to aid the customer in writing command lists. It primarily contains general-use programming interfaces, which allow the customer to write programs that use the services of the NetView program. However, this book also provides the following types of information, which are explicitly identified where they occur: Other product information, such as defining command lists within NetView and implementing message automation is provided to allow the customer to use the NetView program. This information should never be used as programming interface information.

Who Should Use This Book

NetView Customization: Writing Command Lists is designed for system programmers and network operators who are either using command lists or learning how to write command lists. Before you read this book, you should be familiar with how the NetView program is used in your network and what the operators' tasks are. This book does not provide descriptions of NetView operator commands. If a command is unfamiliar, refer to *NetView Operation.*

How to Use This Book

This section includes information about the organization, the terms, and the coding conventions used in this book.

How This Book Is Organized

This book is organized into the following sections:

"Part One. Basic Command List Topics" contains an overview of basic command list topics that are common to command lists written in either REXX or the NetView command list language.

"Part Two. Writing Command Lists in the Restructured Extended Executor Language" contains information about how to write command lists using REXX.

"Part Three. Writing Command Lists in the NetView Command List Language" contains information about how to write command lists using the NetView command list language.

"Part Four. Advanced Command List Topics" contains information on advanced topics that pertain to command lists written in either REXX or the NetView command list language.

™ NetView is a trademark of International Business Machines Corporation.

Appendix A, "REXX Command List Reference Summary" contains summary charts of the instructions and functions provided by the NetView program for use in REXX command lists.

Appendix B, " NetView Command List Language Reference Summary" contains summary charts of all control keywords, control variables, and built-in functions for the NetView command list language.

Appendix C, "Comparison of REXX and NetView Command List Language," contains a comparison between the features of REXX and the NetView command list language.

Appendix D, "Converting Command Lists Written in the NetView Command List Language to REXX," contains information about converting command lists written in the NetView command list language to REXX.

Notes on Terms Used in This Book

Following is a list of terms and the meanings they have in this book. Unless otherwise noted, the abbreviations for products refer to the latest version and release of the product.

Term	**Meaning**
command lists	command lists written in REXX and command lists written in the NetView command list language
member	member in the specified data set (MVS) and file name with a file type of NCCFLST (VM)
MVS	MVS/XA, and MVS/ESA (compatability mode)
REXX	Restructured Extended Executor language (see note 1)
REXX Reference	*VM/SP System Product Interpreter Reference* or *TSO/E REXX Reference*
REXX User's Guide	*VM/SP System Product Interpreter User's Guide* or *TSO/E REXX User's Guide*
VM	VM/SP, VM/SP HPO, and VM/XA (see note 2)
VTAM	VTAM V3R1.1, VTAM V3R1.2, and VTAM V3R2.

Notes:

1. NetView does not support REXX on VM/XA systems.
2. VM/XA runs in compatability mode.

Coding Conventions Used in This Book

The model statements are formatted according to a set of coding conventions which are described in this section.

Braces { }

When braces enclose operands, this indicates that you must choose one of the operands. Any accompanying commas or equal signs must be included. Do not include braces when coding.

Brackets [ ]
: When brackets enclose an operand, this indicates a completely optional specification. Any accompanying commas or equal signs are optional. Do not include brackets when coding the specification.

Ellipsis ...
: An ellipsis replaces a repetition of an operand or variable in syntax statements. Replace ellipses with the appropriate operand or variable when you code.

OR-sign |
: The OR-sign separates choices for an optional or required specification. If a group of options is enclosed by brackets, and the individual options are separated by OR-signs, none of the options in the group has to be chosen. Do not include the OR-sign when coding the specification.

UPPERCASE Characters
: You must enter command names or operands shown in UPPERCASE, **BOLD** characters exactly as they appear. These names are program keywords.

lowercase **Characters**
: Lowercase, *italic* characters describe the kind of program variable information that must be supplied, rather than the literal information. The actual value replaces the lowercase description.

underscored Characters
: Underscored characters indicate **default** values. These values are automatically assigned unless you specify a different value.

What Is New In This Book

In previous releases of NetView, this book was titled *NetView Command Lists*.

Major changes made to this book include information about the Restructured Extended Executor (REXX) language. Release 3 of the NetView program has been enhanced to support REXX command lists. As a result, some existing chapters in this book have been changed to reflect new REXX information, and several new chapters have been added.

Chapter 2, "Restructured Extended Executor Language Overview" on page 23 contains an introduction to REXX and explains how to write REXX command lists for NetView.

Chapter 3, "REXX Instructions Provided by NetView" on page 33 contains detailed information about using the new REXX instructions provided by NetView.

Chapter 4, "REXX Functions Provided by NetView" on page 51 contains detailed information about using the new REXX functions provided by NetView.

Appendix A, "REXX Command List Reference Summary" on page 171 contains a summary of the functions and instructions provided by the NetView program for use in REXX command lists for NetView.

Appendix C, "Comparison of REXX and NetView Command List Language" on page 185 contains a comparison between the features of REXX and the NetView command list language.

Appendix D, "Converting Command Lists Written in the NetView Command List Language to REXX" on page 209 contains information about how to convert command lists written in the NetView command list language to REXX.

Where To Find More Information

Table 1 shows all of the publications in the NetView Release 3 library, arranged according to related tasks. For more information on these and other related publications, see "Bibliography" on page 241.

Table 1. The NetView Library

Evaluation and Education	
Network Program Products General Information	GC30-3350
Bibliography and Master Index for NetView, NCP, and VTAM	GC31-6081
Learning about NetView: Operator Training (PC Diskettes)	SK2T-0292
Planning	
Network Program Products Planning	SC30-3351
NetView Storage Estimates (PC Diskettes)	SK2T-1988
Console Automation Using NetView: Planning	SC31-6058
Installation and Administration	
NetView Installation and Administration Guide	SC31-6018
NetView Administration Reference	SC31-6014
Network Program Products Samples	SC30-3352
NetView Tuning Guide	SC31-6079
Customization	
NetView Customization Guide	SC31-6016
NetView Customization: Writing Command Lists	SC31-6015
NetView Customization: Using PL/I and C	SC31-6037
NetView Customization: Using Assembler	SC31-6078
Operation	
NetView Operation Primer	SC31-6020
NetView Operation	SC31-6019
NetView Command Summary	SX75-0026
Diagnosis	
NetView Problem Determination and Diagnosis	LY43-0001
NetView Resource Alerts Reference	SC31-6024
NetView Problem Determination Supplement for Management Services Major Vectors 0001 and 0025	LD21-0023

Part One. Basic Command List Topics

Chapter 1. Command List Overview

This chapter is for those readers who need to understand what command lists for the NetView™ program are, how to use them, how to create them, and how to run them.

This chapter is intended to provide customers with an overview of using command lists. It contains information on how to use, create, and run command lists. The information in this chapter must not be used for programming purposes.

What Is a Command List

A command list is a set of commands and special instructions that are grouped under one name like a computer program. For NetView, a command list can be written in either REXX or the NetView command list language. When you type a command list name at a terminal, the commands and instructions in that command list are interpreted and executed. There are several ways to run command lists besides entering a command list name at a terminal. For example, you can issue a timer command to run a command list at a specified time or time intervals. You can also run more than one command list at the same time under different tasks. See "How to Run Command Lists" on page 10 for more information on how to run command lists.

Command lists help you control your network and make the operators' jobs easier. Command lists obtain information from operators, other tasks, system resources, or the contents of messages. The command list uses this information to perform processing or to decide the next action. This flexibility lets you automate repetitive or complex routine operations, perform resource recovery, and handle operations consistently among different operators.

How Command Lists Can Help You

Command lists help you automate your system and network in the following ways:

- A command list can ask the operator questions and take action based on the answers.
- A command list can display information on an operator's screen.
- A command list can reword, delete, or reply to a message before the operator sees it.
- A command list can wait for NetView to receive a message or group of messages and take action based on the message content.
- A command list can speed backup and recovery procedures (for example, automatic recovery of a failing resource).
- A command list can tailor operator commands and procedures for your network.

™ NetView is a trademark of International Business Machines Corporation.

- A command list can monitor and restart subsystems and programs (for example, VTAM, CICS, and DB2).

System programmers or operators can write command lists to:

- Simplify entry of operator commands
- Ensure completeness and correct order when a sequence of commands must be issued
- Provide for commands to be issued automatically when specific messages are received during the operation of systems, networks, and applications
- Implement specialized operator dialogs that extend the operator's role or increase the efficiency and productivity of operators.

Command lists can save time and make the operator's job easier in the following ways:

- A command list can combine complex or multiple routine jobs or both. The operator can do all the jobs by entering the command list name at the terminal.
- Complex or lengthy functions can be performed consistently among operators by using the same command list.

Before you write a command list, it is important that you analyze your system and network operating procedures and the tasks that your operators regularly perform. Decide which of these jobs you want to automate using command lists. Start by writing simple command lists and add the more complex functions as you gain experience. This book does not describe how to use NetView operator commands. If you need information about an unfamiliar command, see *NetView Operation*.

The following are examples of command lists that simplify network operation.

Examples of Common Startup Command Lists

If the operators need to set up terminal access facility (TAF) sessions with the Information Management System (IMS) and the Host Command Facility (HCF), a command list can be used instead of entering individual commands.

The STARTUP1 command list in Figure 1 is an example of a REXX command list that can be used to establish terminal access facility (TAF) sessions with IMS and HCF.

```
/* STARTUP1 */
'BGNSESS OPCTL,APPLID=IMS1,SRCLU=TAF11,LOGMODE=OPCTLLOG,SESSID=IMS'
'BGNSESS OPCTL,APPLID=HCF1,SRCLU=TAF11,LOGMODE=OPCTLLOG,SESSID=HCFA'
'BGNSESS OPCTL,APPLID=HCF1,SRCLU=TAF12,LOGMODE=OPCTLLOG,SESSID=HCFB'
EXIT
```

Figure 1. Common REXX Startup Command List

The STARTUP2 command list in Figure 2 on page 5 is an example of a command list written in the NetView command list language that can be used to establish the same terminal access facility sessions.

```
STARTUP2  CLIST
BGNSESS OPCTL,APPLID=IMS1,SRCLU=TAF11,LOGMODE=OPCTLLOG,SESSID=IMS
BGNSESS OPCTL,APPLID=HCF1,SRCLU=TAF11,LOGMODE=OPCTLLOG,SESSID=HCFA
BGNSESS OPCTL,APPLID=HCF1,SRCLU=TAF12,LOGMODE=OPCTLLOG,SESSID=HCFB
&EXIT
```

Figure 2. Common NetView Command List Language Startup Command List

Instead of having to remember and enter three commands, operators can now simply enter the command list name STARTUP1 or STARTUP2. The command list starts the three sessions and operators receive the same messages that they would usually receive if they had entered all three commands.

Examples of Activating a Network Control Program

For operators who need to activate a Network Control Program (NCP), you can write a command list to simplify the activation of the NCP. Figure 3 is an example of a REXX command list that activates the NCP.

```
/* NCP1 */
'V NET,ACT,ID=NCP1,LOAD=YES,LOADSTA=LINK1'
EXIT
```

Figure 3. REXX Command List to Activate a Network Control Program

Figure 4 is an example of a command list written in the NetView command list language that activates that same NCP.

```
NCP2  CLIST
V NET,ACT,ID=NCP1,LOAD=YES,LOADSTA=LINK1
&EXIT
```

Figure 4. NetView Command List Language Command List to Activate a Network Control Program

The operator can now use the NCP1 or NCP2 command list to activate NCP1.

These are examples of simple command lists, illustrating some basic command list features. Both REXX and the NetView command list language provide the ability to perform additional functions. Detailed information about writing command lists in either language is provided in the following chapters.

How Command Lists are Created

You can create command lists before NetView is started or while it is running. Code each command list as a member of a command list data set. After you create the command list data set, you can use facilities such as ISPF (for VM and MVS), IEBUPDTE (for MVS), or XEDIT (for VM) to update the command list.

NetView supports command lists in data sets that are concatenated across volumes. The member name is the command list name unless another name was defined for the command list on a CMDSYN statement. For more information on

CMDSYN, see *NetView Administration Reference.* The command list name can be from 1- to 8-characters (0-9, A-Z, @, $, #). The command list name must begin with a non-numeric character.

Note: When NetView is operating on an MVS system and you plan to update or create command lists while NetView is running, define your command list data sets without secondary extents. Otherwise, a command list might be filed in a new extent, and you will have to stop and restart NetView to use the command list.

Once a command list is created for MVS, the existence of the member is sufficient to allow an operator to run the command list. For VM, you must reaccess the minidisk to use the command list. This is all of the definition that is required to utilize the command list unless it is to be scope-protected or driven by a message.

For MVS

You must first create the data set that will be used to store the command lists. Code each command list as a separate member of a command list data set. To define the name of the command list data set to the NetView start procedure, code the JCL DD statement for the DSICLD as shown in Figure 5.

```
//DSICLD  DD  DSN=datasetname,DISP=SHR
```

Figure 5. JCL to Define a Command List Data Set

Data sets can be concatenated by coding the DSICLD statement as shown in Figure 6.

```
//DSICLD  DD  DSN=datasetname1,DISP=SHR
//        DD  DSN=datasetname2,DISP=SHR
//        DD  DSN=datasetname3,DISP=SHR
//        DD  DSN=datasetnamen,DISP=SHR
```

Figure 6. JCL to Define Concatenated Command List Data Sets

The first command list data set defined under DSICLD must have the largest block size of any concatenated command list data sets, or the first DD statement must have a DCB=(BLKSIZE=XXXX) statement where XXXX is equal to the largest block size of the concatenated data sets.

Note: To make sure your command lists are accessed when they have the same name as IBM-supplied command list data sets, concatenate your command list data sets before the IBM-supplied ones. Make sure the block size is 3920 or less to reduce paging caused by the block size exceeding the size of a page of memory.

For VM

You must create a file using the name of the command list as the file name with a file type of NCCFLST. Be sure to access the minidisk after you define the command list.

Note: If there is not a CMDMDL statement associated with a command list written in the NetView command list language, the CLIST statement must be the first record of the command list. See "General Coding Conventions" on page 72 for information on coding a CLIST statement.

Who Can Use Command Lists

Once a command list is created, the NetView operator can use that command list by entering the command list name.

You can limit command list access to a specific group of operators by causing NetView to scope check each command list when it is run. For each command list you want scope checked, include the CMDMDL statement shown in Figure 7 in the DSICMD member.

```
cmdlistname    CMDMDL    MOD=DSICCP
```

Figure 7. CMDMDL Statement Syntax for Command Lists

Following the CMDMDL statement, enter the appropriate CMDCLASS statement to reflect the restrictions that apply to your NetView system. For more information on how to code CMDMDL and CMDCLASS statements, see *NetView Administration Reference.*

Note: You must restart NetView after the new definitions are included in DSICMD to put the appropriate scope checking into effect.

With NetView commands, you can scope check certain keywords that are entered with the commands using the KEYCLASS definition statement. However, you cannot use KEYCLASS to scope check parameters that are entered with a command list.

If you need to scope check the parameters that are entered with a command list, you can execute a command list from a PL/I or C program. The program can then execute the command list and pass the parameters to the command list. For more information on writing programs in PL/I or C, see *NetView Customization: Using PL/I and C.*

Loading Command Lists Into Storage

NetView provides the ability to load command lists into main storage prior to execution.

Although it is not mandatory that you load a command list into main storage before it is executed, pre-loading promotes improved performance of your computer system. If you invoke a command list that has not been pre-loaded, it is loaded into main storage, executed, and then dropped from main storage. Therefore, every time the command list is executed, it must be retrieved from the auxiliary storage device where it resides. By pre-loading the command list, it can be executed multiple times without having to be retrieved from auxiliary storage each time.

There are three NetView commands that allow you to move command lists into and out of main storage, and list command lists that are currently in main storage:

LOADCL	loads command lists into main storage shared by all operators.
DROPCL	drops a command list that was previously loaded into main storage using the LOADCL command.
MAPCL	lists command lists that currently reside in main storage.

A description of each command is provided on the following pages. For additional information on the LOADCL, DROPCL, or MAPCL command, refer to *NetView Operation*.

The NetView program provides a sample REXX command list (CNMS8003) that can help you manage the number of command lists that have been loaded into storage using the LOADCL command. The sample uses the MAPCL and DROPCL commands to conditionally drop commands from main storage. Browse the sample command list for more information on how it works.

LOADCL Command

Use the LOADCL command to load command lists into main storage. Figure 8 shows the syntax of the LOADCL command. The operands can be entered in any order.

LOADCL *cmdlistname*[,...] [(**REPLACE**)]

Figure 8. LOADCL Command Syntax

cmdlistname[,...]

for MVS, the names of the members within the DSICLD MVS data set that contain the command lists to be loaded into storage.

for VM, the names of the VM files that contain the command lists to be loaded into storage. The file type must be NCCFLST.

Note: Any synonyms defined for a command list through the NetView CMDSYN command can be used with the LOADCL command.

(REPLACE)

indicates that you want to load new copies of any of the command lists that were previously loaded using LOADCL. When all of the current users have finished using the previously loaded copy, it is automatically dropped.

If a command list is already loaded and REPLACE is not specified, no load can occur for that command list.

Note: If you change a command list that has already been loaded into main storage, you must issue the LOADCL command with the (REPLACE) operand specified. This loads the updated version of the command list into main storage so that it is executed instead of the old version.

If you have command lists that are frequently executed, you can load them into main storage when NetView is initialized. For example, if the command lists STARTJOB and SETTERM are run often, you can load them into main storage by coding the statements shown in Figure 9 in your initialization command list. After initialization, the STARTJOB and SETTERM command lists will reside in main storage and are available for execution.

```
LOADCL STARTJOB
LOADCL SETTERM
```

Figure 9. Examples of LOADCL Commands

For more information on loading command lists into storage when NetView is initialized, refer to "NetView Initialization" on page 10.

DROPCL Command

You can remove a command list from main storage by using the DROPCL command. Figure 10 shows the syntax of the DROPCL command.

```
DROPCL *|cmdlistname[,...]
```

Figure 10. DROPCL Command Syntax

*

indicates that all storage-resident command lists should be removed from main storage.

cmdlistname[,...]

for MVS, the names of the members within the DSICLD MVS data set that contain the command lists to be removed from storage.

for VM, the names of the VM files that contain the command lists to be removed from storage. The file type must be NCCFLST.

Note: Any synonyms defined for a command list through the NetView CMDSYN command can be used with the DROPCL command.

MAPCL Command

The MAPCL command can be used to list all command lists currently residing in main storage or to determine if a specific command list resides in main storage. For information on output displayed for MAPCL, see *NetView Operation.* Figure 11 shows the syntax of the MAPCL command.

```
MAPCL  [*|cmdlistname[,...]]
```

Figure 11. MAPCL Command Syntax

<u>*</u>

indicates that all storage-resident command lists should be listed. This is the default if MAPCL is entered with no parameters.

cmdlistname[,...]

for MVS, the names of the members within the DSICLD MVS data set that contain the command lists that can reside in main storage.

for VM, the names of the VM files that contain the command lists that can reside in main storage. The file type must be NCCFLST.

If the command lists are storage-resident, they are listed.

Note: Any synonyms defined for a command list through the NetView CMDSYN command are not supported by this command.

How to Run Command Lists

You should design command lists that run with little outside help from operators. Some of the ways command lists can be run are:

- By NetView initialization
- By operator logon to NetView
- By message automation
- By an operator command (including timer commands)
- By another command list
- By a user-written command processor.

NetView Initialization

You can define a command list to run automatically when NetView is started. The NetView initialization command list runs under the PPT task. See "Primary POI Task Restrictions" on page 17 for information about PPT restrictions.

You can run only one command list at initialization, but this command list can call other command lists. "Another Command List" on page 13 explains the rules that apply to calling another command list.

Code the name of the command list you want to run on the NCCFIC definition statement in DSIDMN. For example, if you want to run the SETUP command list, code the NCCFIC statement as shown in Figure 12.

```
NCCFIC    IC=SETUP
```

Figure 12. Example of NCCFIC Definition Statement

The default NCCFIC statement coded in the sample DSIDMN shipped with NetView is:

```
NCCFIC IC=CNME1034 DSIMSG01
```

This invokes command list CNME1034 (the default initialization command list) with the parameter DSIMSG01. The parameter DSIMSG01 is the name of the message automation table that will be in effect when NetView is initialized. If you want to use another table, change this parameter.

For more information on NCCFIC, see *NetView Administration Reference*.

You can include many types of commands in your initialization command list. The following list describes some of the commands you may want to include:

- To route unsolicited messages, include ASSIGN commands. ASSIGN commands allow you to automatically set up unsolicited message routing for the operators.

- To start message automation, include AUTOMSG commands.

  NetView already includes the DSIMSG01 member to set up message automation, and for running PDFILTER when the hardware monitor is initialized. The AUTOMSG command is coded as follows:

  ```
  AUTOMSG MEMBER=DSIMSG01
  ```

- To establish authorized operators for the IBM-supplied message automation command lists, include the following statements:

```
&CGLOBAL CGAUTHID1
&CGAUTHID1=operid
```

- To restore AT, EVERY, and AFTER commands that were entered with the SAVE option, include a RESTORE TIMER command. By putting the RESTORE TIMER command in your initialization command list, you ensure that the saved TIMER commands are restored at NetView initialization. By restricting use of the RESTORE TIMER command to only the initialization command list, you ensure that the timers are restored only once.
- To start operator tasks that handle your system and network automation, include AUTOTASK commands.
- To load command lists into main storage, include LOADCL commands.

Operator Logon

You can define a command list to run automatically after the operator successfully logs on. Only one command list can be defined to run when an operator logs on, but this command list can activate other command lists. Refer to "Another Command List" on page 13 for rules that apply when calling another command list. The name of the activating command list is coded in the operator's profile.

Code the name of the command list you want to run in the operator's profile using the IC operand of the PROFILE statement. For example, if you want to run the HELLO command list every time an operator logs on; and if the operator has a profile of PROFBEG, the IC operand (as shown in Figure 13) should be added to this profile.

```
PROFBEG  PROFILE  IC=HELLO
```

Figure 13. Example of PROFILE Definition Statement

Note: Some operator IDS are assigned by using the AUTOTASK command, which runs network automation tasks. Operator tasks started by the AUTOTASK command do not have a terminal attached (as specified in the PROFILE statement for its operator). Therefore, if an initial command list is to be run after the operator successfully logs on (as specified on the IC operand of the operator's PROFILE statement), the initial command list should not set any PF keys or invoke any NetView full-screen displays.

For more information on the PROFILE definition statement, see *NetView Administration Reference.*

Message Automation

A command list can be initiated by NetView upon receipt of a message. These command lists can send a command as an automatic response to the message, or they can use the GENALERT command to represent the event as an alert in the hardware monitor data base. For the format of the GENALERT command, see *NetView Operation.*

Command lists initiated by NetView upon receipt of a message contain a series of commands to perform a function as a result of the message. For example, if the message reported that an NCP failed, the command list can issue the VTAM command to reactivate the NCP. See Chapter 9, "Message Automation" on page 135.

Operator Command

The operator can enter a command list name from the terminal in the same way any other command and operands are entered. When the name of the command list is entered, the command list starts processing. Message responses and other information can be sent to the operator, depending on how the command list is written.

NetView operators can activate, stop, suspend, or restart command list processing by entering the NetView commands GO, RESET, STACK, or UNSTACK. For command lists written in REXX, the commands are entered when the command list is waiting for a response to a PARSE EXT, PARSE PULL, or WAIT instruction. For command lists written in the NetView command list language, the commands are entered during command list pauses or command list waits. The GO command must precede any data entered in response to a PARSE EXT or PARSE PULL or in response to an &PAUSE. For more information about the GO, RESET, STACK, and UNSTACK commands, see *NetView Operation*.

Operators can use the following NetView commands to run command lists at a specified time or time interval:

AFTER	instructs NetView to run the command list after a specified period of time.
AT	instructs NetView to run the command list at a particular time.
DELAY	instructs NetView to wait the specified amount of time and then run the command list once.
EVERY	instructs NetView to run the command list repeatedly at a certain time interval.

You can set up the AT, DELAY, EVERY, and AFTER commands so the command list runs even if the operator is not logged on at the time. This is done with the PPT operand. However, some commands cannot be used in a command list running under the PPT. Read "Primary POI Task Restrictions" on page 17 for more information.

Operators can use the CMD command to queue a command list at a different priority than its default.

Command lists can be defined so that they always interrupt the processing of other command lists. This is done using the TYPE= parameter of the CMDMDL statement in the DSICMD. For more information, see *NetView Administration Reference.*

To learn more about the AT, DELAY, EVERY, AFTER, and CMD commands, see *NetView Operation.*

Another Command List

One command list can activate another command list. When a command list is running under the control of another command list, it is nested within the calling command list. To nest a command list within another command list, code the name of the called command list as a command within the controlling command list. When NetView reaches a statement with the name of a command list, NetView starts running the nested command list. When NetView reaches the end of the nested command list, NetView returns control to the calling command list and processes the next statement.

Command lists written in REXX and command lists written in the NetView command list language can call each other. A REXX command list can be invoked as a REXX command, subroutine, or function. A REXX command list can call a command list written in the NetView command list language as a command but not as a subroutine or a function. A command list written in the NetView command list language can call another command list written in the NetView command list language or a REXX command list as a command. For information about REXX subroutines and functions, see *REXX User's Guide* and *REXX Reference.*

When REXX command lists and command lists written in the NetView command list language call each other, parameters can be passed from the calling command list to the nested command list. However, when the nested command list is finished, only a return code is returned to the calling command list. To pass variables between the calling command list and the nested command list, use NetView global variables. "REXX GLOBALV Instruction" on page 44 provides information about setting and retrieving global variables in REXX command lists. For information on defining global variables in command lists written in the NetView command list language, see Chapter 8, "NetView Command List Language Global Variables" on page 123.

You can have 250 levels of externally nested command lists. This means that you can write a command list which activates another command list. The nested command list can activate a third command list. The third command list can then activate a fourth, and so on. To visualize how this process works, see Figure 14 on page 14.

Note: Only REXX command lists invoked as commands, external subroutines, or external functions count as one of the 250 levels of externally nested command lists. You can invoke up to 250 REXX command lists as internal subroutines and functions but they do not count toward the 250 levels of externally nested command lists.

You should test each command list by itself before running the command list as part of a nested chain of command lists. If a nested command list encounters an unrecoverable error, the command list ends and passes the error back to the command list from which it was called. If the calling command list is written in REXX, it might be able to take action to recover from the error passed to it from the nested command list. For information on coding REXX command lists that can recover from errors, see "Recovering from Errors in REXX Command Lists" on page 31. If the calling command list is written in the NetView command list language, and an error occurs in the nested command list, the calling command list also ends. If the calling command list was called by another command list, it continues to pass the error back to the command list from which it was called.

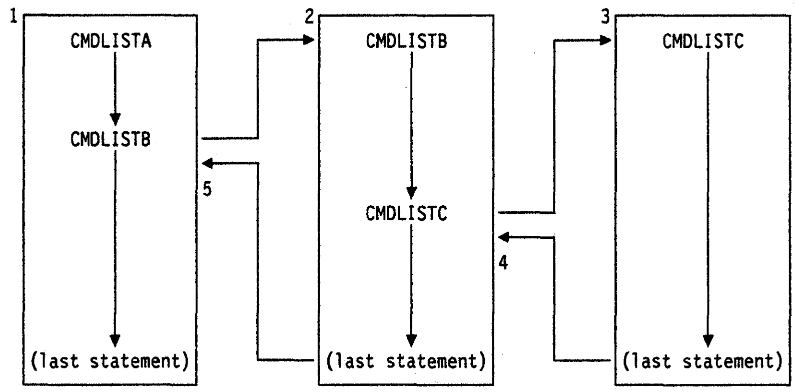


Figure 14. Nested Command Lists

User-Written Command Processor

You can write a command processor that activates a command list. Command processors are programs written in languages such as Assembler, PL/I, or C. For information on how to write command processors, see *NetView Customization: Using Assembler.*

Using Network Commands in Command Lists

You can use network commands in a command list. The following is a partial list of some of the types of network commands you can include:

- NetView commands
- User-written NetView commands
- VTAM commands.

The commands used within command lists are still limited by the operator's span of control and the scope of the commands.

Notes:

1. You cannot use the NetView RETURN command.
2. You can only use NetView and user-written commands that are defined on the CMDMDL statement as regular or both (TYPE=R or TYPE=B).
3. You must use the appropriate prefix with session monitor (NLDM), hardware monitor (NPDA), and status monitor (STATMON) commands.

The following sections describe how you can use NetView commands in command lists.

Using System Commands

System commands can be used in command lists. The NetView command MVS is available to enter MVS commands in command lists. For example, 'MVS S *jobname*' or 'MVS D A,L'. See *NetView Operation* for more information about the MVS command.

Using Long Running Commands

You can use long running commands in your command lists. There are two types of long running commands: minor and major. The type of long running command, minor or major, and whether the command list uses the CMD command to queue the command, determines whether the long running command or the issuing command list receives execution priority.

Using Minor Long Running Commands

The NetView BGNSESS (FLSCN) and NCCF commands are minor long running commands. When issued from a command list, a minor long running command performs syntax checking and other synchronous error tests. The value of the return code (RC in REXX command lists or &RETCODE in command lists written in the NetView command list language) contains the result of these tests. When the issuing command list is complete, the minor long running command is executed. Any errors that occur while the long running command is executing are reported in messages. To access these messages, use message automation, the TRAP and WAIT instructions (REXX), or the &WAIT control statement (NetView command list language).

Notes:

1. When a task receives a message, a check is first made to determine if a command list is waiting for a message. If not and if message automation is being used, then the message is checked against the message automation table. Once a message is used by a command list for wait processing (TRAP and WAIT or &WAIT), that same message cannot be used by a message automation table.
2. You do not need to issue the NCCF minor long running command from a command list because NetView ensures that the command facility screen is displayed whenever line mode messages are presented.

To define a user-written command as a minor long running command, use the DSIPUSH macro. See *NetView Customization: Using Assembler* for information on DSIPUSH.

Using Major Long Running Commands

With the exception of the BGNSESS (FLSCN) and NCCF commands, all other long running commands are major long running commands. When a major long running command is issued from a command list, execution of the command list is suspended while the command executes. It may be necessary for the operator to indicate that the major long running command is complete by issuing a RETURN or END command before the calling command list resumes processing.

If a command list issues a major long running command, and while the command is executing, the same major long running command is entered, the first command is canceled. The major long running command then passes control to the issuing command list:

- When the issuing command list is written in REXX, it is recommended that you code SIGNAL ON HALT. If you do not code SIGNAL ON HALT, the operator will see inappropriate termination messages. You should code EXIT -5, and you should not generate any messages in the HALT subroutine. See "Recovering from Errors in REXX Command Lists" on page 31 for more information on coding SIGNAL ON HALT.

- When the issuing command list is written in the NetView command list language, the command list is also canceled.

You can also cancel the calling command list with the UNIQUE command. See *NetView Operation* for information on UNIQUE.

Queuing Long Running Commands

You can control the execution of long running commands by using the NetView CMD command to queue them. When queued, all long running commands are processed in the same manner, regardless of whether the command is minor or major. Queuing a long running command causes it to be processed independently of your command list. The result of the long running command does not influence the result of the command list. When you queue a long running command, the return code indicates the result of the queuing operation only. You cannot get a return code from the queued command.

To ensure that TAF command output is displayed before the command list resumes processing, use CMD HIGH BGNSESS FLSCN. If the operator ROLLs from the current long running command, the command list continues. If the long running command is canceled, the cancel is not passed back to the issuing command list. For more information on TAF, see *NetView Operation*.

To delay the execution of NLDM until your command list is finished executing, is stacked, is canceled, or is otherwise interrupted, use CMD LOW NLDM.

Using the VIEW Command

The VIEW command can be used in command lists to display panels. The VIEW command has access to local and global variables set in the command list that issues the VIEW command. See *NetView Customization Guide* for more information on the VIEW command.

Using Full-Screen Commands

If a command list that is executed from a full-screen processor issues a full-screen command, the NetView program can display the command facility screen before displaying the output of the full-screen command. The command facility screen is only displayed if the command list generates any other output that is displayed to the operator. Display of the command facility screen suspends any AUTOWRAP setting and prevents the full-screen output from being automatically displayed. To minimize the possibility of displaying command facility screen output, define and code the command list so that it does not generate any other output to be displayed. For example:

- Code a CMDMDL definition statement with ECHO=NO for the command list. See *NetView Administration Reference* for information on coding a CMDMDL statement.
- Code TRACE ERRORS or TRACE OFF at the beginning of a REXX command list or &CONTROL ERR at the beginning of a command list written in the NetView command list language. See *REXX Reference* or *REXX User's Guide* for information on the TRACE instruction.
- Do not code any SAY instructions in a REXX command list or any &WRITE or &BEGWRITE control statements in a command list written in the NetView command list language.
- Do not issue any commands that have line mode output.

Primary POI Task Restrictions

Command lists run under the primary POI task (PPT) when they meet *any* of the following criteria:

- Routed to the PPT for execution as a result of message automation
- Coded on an NCCFIC definition statement to run when NetView is initialized
- Called with an AT, EVERY, AFTER, or EXCMD command that uses the PPT as an operand. (PPT on AT, EVERY, and AFTER allows the command to be run even when the operator who scheduled it is not logged on.)

The following restrictions apply to command lists run under the PPT:

- In general, full-screen commands and immediate commands cannot be used. Do not use the following NetView commands:
  - AUTOWRAP
  - BGNSESS
  - CLOSE
  - GO
  - INPUT
  - LOGOFF
  - MOVE
  - ROUTE
  - SET
  - START
  - STOP
  - SWITCH
  - WTO
  - WTOR.
- Do not use the following REXX instructions:
  - FLUSHQ
  - MSGREAD
  - PARSE EXT
  - PARSE PULL
  - TRAP
  - WAIT.
- Do not use the following NetView command list language control statements:
  - &PAUSE
  - &WAIT.
- Do not execute command processors that use the MVS/GCS STIMER macro.

Note: Command lists running under the PPT should not generate messages containing non-Latin characters (double-byte character sets, such as Kanji) that will be routed to the system console.

AUTOTASK OST Restrictions

Some command lists run under an OST which is started by an AUTOTASK command that sets up a subtask called an automation task. Command lists running under an automation task can handle message automation.

Because an automation task handles message automation, it does not have a terminal logged on to it. Therefore, neither full-screen commands nor commands that support specific keyboard functions (such as SET PF keys) are useful in automation task command lists.

Writing Bilingual Command Lists

A command list can be written in REXX, the NetView Command List language, or both. A command list written in both languages is referred to as a *bilingual* command list.

Bilingual command lists help to ensure that consistent results are achieved when a common command list is executed by operators at multiple installations. For instance, you may have one installation that has the REXX interpreter installed and another that does not. If you create a bilingual command list that can be executed in both installations, you help to ensure that the results are consistent. All command lists reside on the same fixed record length library.

NetView determines the language in which a command list is written by checking the first record of the command list. A REXX command list starts with a comment, so its first record must contain "/*" as the first two non-blank characters. The comment must end with the "*/" characters. A command list written in the NetView command list language must have the character string CLIST in the first 71 characters of its first record. The structure of the first record of a bilingual command list is as follows:

1. Columns 1 and 2 must contain the characters "/* ".
2. Beginning in column 3, there can be optional non-blank characters.
3. Following any non-blank characters, there must be one or more blank characters.
4. The character string CLIST must follow the one or more blank characters.
5. If any parameter variables are being passed, there must be one or more blank characters between the CLIST character string and the parameter variables.

Note: You must code &EXIT at the end of the NetView command list language portion of a bilingual command list. You must code the characters "*/" before the beginning of the REXX portion.

When processing a bilingual command list, NetView determines whether to execute the REXX portion or the NetView command list language portion based on the following criteria:

REXX Active?	Command List First Record	Action
Yes	/*...	The REXX interpreter is invoked. The REXX portion of the command list is processed. The NetView command list language portion of the command list is treated as a comment.
Yes	/*...CLIST	The REXX interpreter is invoked. The REXX portion of the command list is processed. The NetView command list language portion of the command list is treated as a comment.
No	/*...	NetView issues an error message.
No	/*...CLIST	The NetView command list language interpreter is invoked. The NetView command list language portion of the command list is executed. The REXX portion of the command list is ignored.
Yes|No	Does not start with /*...	The NetView command list language interpreter is invoked.

Figure 15 provides an example of how bilingual command lists should be structured. Because the first line of the command list contains "/*" in columns 1 and 2 and ends with the character string CLIST, NetView recognizes the command list as bilingual.

```
/*SAMPLE CLIST
*   AFTER THE FIRST RECORD WOULD BE
*   THE COMMAND LIST WRITTEN IN THE
*   NETVIEW COMMAND LIST LANGUAGE
    .
    .  (NETVIEW COMMAND LIST LANGUAGE PORTION OF THE COMMAND LIST)
    .
*   THE NETVIEW COMMAND LIST LANGUAGE PORTION OF THE COMMAND LIST
*   WOULD END WITH &EXIT
&EXIT
*/
/*  OPTIONALLY, YOU MAY HAVE A COMMENT HERE THAT
    IDENTIFIES THE BEGINNING OF THE REXX PORTION OF
    THE COMMAND LIST */
    .
    .  (REXX PORTION OF THE COMMAND LIST)
    .
```

Figure 15. Example of a Bilingual Command List

For information on converting command lists written in the NetView command list language into REXX or bilingual command lists, see Appendix D, "Converting Command Lists Written in the NetView Command List Language to REXX" on page 209.

What an Operator Sees when a Command List Runs

You can control the amount of data displayed to the operator during the execution of a command list. Responses to commands in the command list or messages the command list sends to the terminal screen can be displayed to the operator.

To control the amount of data displayed to the operator during the execution of a REXX command list, use the TRACE instruction (see *REXX Reference*), the TRAP instruction, (see "REXX TRAP Instruction" on page 34), or the suppression character (see "Suppressing Display of Non-REXX Commands" on page 25).

To control the amount of data displayed to the operator during the execution of a command list written in the NetView command list language, use the &CONTROL control statement (see "&CONTROL Control Statement" on page 92), the &WAIT SUPPRESS control statement (see "Customizing the &WAIT Statement" on page 118), or the suppression character (see "Conventions for Suppression Characters" on page 74).

The commands and messages displayed during execution of a command list appear in the message area of the NetView screen. Output from the command list is preceded by a type code of C. For a complete description of the NetView screen layout and the format of messages sent to the screen, see *NetView Operation*.

Part Two. Writing Command Lists in the Restructured Extended Executor Language

Chapter 2. Restructured Extended Executor Language Overview

This chapter offers a brief introduction to REXX. Not all of the features and syntax rules of REXX are described in this manual. This manual focuses primarily on the REXX instructions and functions provided by the NetView program. For more detailed information about REXX, see your *REXX Reference* or *REXX User's Guide*.

Notes:

1. In this book, *REXX Reference* refers to *TSO/E REXX Reference* for MVS users or *VM/SP System Product Interpreter Reference* for VM/SP users.
2. In this book, *REXX User's Guide* refers to *TSO/E REXX User's Guide* for MVS users or *VM/SP System Product Interpreter User's Guide* for VM/SP users.
3. NetView does not support REXX on VM/XA systems.
4. For NetView to support REXX on MVS systems, TSO/E must be installed but does not have to be active.

Introduction to the Restructured Extended Executor Language

REXX is an *interpretive* language. This means that the REXX interpreter operates directly on the program as it executes, line-by-line and word-by-word. An interpreted language is different from other programming languages, such as COBOL, because it does not have to be compiled before it is executed.

Each REXX command list must begin with a comment. A comment is marked with "/*" at the beginning and "*/" at the end. You can insert comments in your REXX command list wherever necessary.

A REXX command list consists of a series of clauses, each having a separate purpose. In a simple REXX command list, the clauses are interpreted in the sequence in which they are coded. You can control the sequence in which clauses are executed by using specific commands that alter the processing order.

A REXX instruction tells the REXX interpreter to do something. A REXX instruction is identified by its keyword, which must be the first item in the clause.

When an equal sign (=) is the second item in a clause, the clause is identified as an assignment clause. Assignment clauses allow you to give a value to a variable. Variables allow you to define different values for the clauses within a command list.

When the second item in a clause is a colon (:), the clause is interpreted as a label. Labels serve to identify the target statement for a transfer of control.

The REXX language allows you to call internal or external routines, called functions. REXX function names must always be followed by parentheses. There can be up to ten expressions, separated by commas, between the parentheses. An expression is something that can be computed. The REXX interpreter performs the computation named by the function and returns a result. The result is then used in the expression in place of the function call. To use a function, place the function name in the command list at the location where you want the result to be accessed. There are also several built-in functions included in the REXX language that perform pre-defined operations.

See *REXX Reference* or *REXX User's Guide* for a complete description of the features of the REXX language.

Coding Conventions for REXX Command Lists

This section describes the syntax rules that apply when coding REXX command lists for NetView.

Record Size

The records in REXX command lists for NetView can be up to 80 characters in length. If the first record of a REXX command list contains a sequence number in columns 73 through 80, then all records in that command list will be truncated to 72 characters.

Using Quotes

To avoid variable substitution on a string in a REXX command list, enclose the string in either single quotes (') or double quotes ("). The quotes signify that you do not want REXX to perform variable substitution on the string. That is, you do not want the REXX interpreter to interpret the string. When REXX encounters a quote (single or double) on a command list statement, it stops interpreting until it reaches a matching quote.

Do not enclose REXX instructions in quotes. REXX recognizes its own instructions and does not perform variable substitution on them. Following are some examples showing how quotes are used to prevent variable substitution with the REXX SAY instruction:

```
SAY 'THIS IS A STRING WITH SINGLE QUOTES'
SAY "THIS IS A STRING WITH DOUBLE QUOTES"
```

These two instructions would display the following at your terminal:

```
THIS IS A STRING WITH SINGLE QUOTES
THIS IS A STRING WITH DOUBLE QUOTES
```

To use an apostrophe or double quotes within the text of a string enclosed in quotes, you can do the following:

```
SAY "IT'S EIGHT O'CLOCK.  TIME TO BRING UP CICS."
SAY 'IT''S EIGHT O''CLOCK.  TIME TO BRING UP CICS.'
SAY 'PLEASE ENTER "GO NODENAME" OR "GO STOP"'
SAY "PLEASE ENTER ""GO NODENAME"" OR ""GO STOP"""
```

The first two instructions would both display the first line below, the last two instructions would both display the second line:

```
IT'S EIGHT O'CLOCK.  TIME TO BRING UP CICS.
PLEASE ENTER "GO NODENAME" OR "GO STOP"
```

Generally, you should enclose any NetView commands, or system commands recognized by NetView, in quotes. The exception is when you want variable substitution to take place on an operand of such a command. If you want variable substitution to take place, leave the operand outside of the quotes.

For example, if you want to use the NetView INACT command in a command list to deactivate a node named NODE1, you would code:

```
'INACT NODE1'
```

However, if the command list contains a variable named NODE and you want to deactivate the node whose name is the current value of the NODE variable, you would code:

```
'INACT ' NODE
```

The following is another example of using quotes to have REXX perform variable substitution on only part of a command:

```
ARG DDNAME
ADDRESS MVS 'EXECIO 1 DISKR ' DDNAME ' ( STEM LINE'
```

This example would first parse the user's input into a variable called DDNAME. The TSO/E EXECIO command is then used to read a line of that DDNAME. ADDRESS MVS is a REXX instruction, so it is not enclosed in quotes. The quotes begin before EXECIO because it is a TSO/E command. The quotes end before DDNAME to allow REXX to substitute the current value of the DDNAME variable into the EXECIO command. The rest of the EXECIO command is enclosed in quotes so that variable substitution does not take place on the STEM and LINE operands.

Suppressing Display of Non-REXX Commands

Use the REXX TRACE command to control the suppression or echoing of non-REXX commands. The SUPPCHAR command of the NCCFID statement does not influence the echoing of non-REXX commands.

When issuing a command that returns its status in the return code, you can enhance the performance of your command list by suppressing synchronous output from the command. To suppress synchronous output, code the suppression character defined on the NCCFID statement twice. For example, if the suppression character is defined as a question mark and you coded the following in a REXX command list:

```
'??SET PF24 IMMED RETRIEVE'
```

no synchronous output from the command is displayed to the operator.

Use the double suppression character to enhance performance of commands that produce line mode messages synchronously and when sufficient status is provided by the return code. Using the double suppression character does not affect output that is scheduled by a command (for example, D NET,APPLS) nor does it reliably reduce output from a long running command (for example, NLDM).

NetView Restrictions on REXX Instructions

This section describes the restrictions that apply when coding REXX instructions in REXX command lists for NetView.

Pausing for Operator Input

The REXX instructions PARSE EXT, PARSE PULL, PULL, and TRACE ? cause a command list to pause for operator input.

Using the PARSE EXT or PARSE PULL instructions along with other instructions, you can code command lists that ask the operator questions and pick up entered responses. Use the REXX SAY instruction to describe what the operator should enter. Code the PARSE EXT or PARSE PULL instruction after the SAY instruction to temporarily stop the command list (unless, in the case of PARSE PULL, there is data on the data stack). After the command list has temporarily stopped, the operator must enter the NetView GO command before it will continue. Any data to be passed to the command list must be entered as an operand or operands on the GO command. For example, to have the command list process a yes or no answer from the operator, you could code the following SAY and PARSE PULL instructions:

```
SAY 'ENTER "GO YES" OR "GO NO" TO CONTINUE'
PARSE UPPER PULL ANSWER
```

The operator could respond to the command list with either GO YES or GO NO. The GO command causes the command list to continue processing, and the YES or NO value is picked up by the PARSE PULL instruction.

Using the SAY Instruction

The REXX SAY instruction can have a character string of any length; however, NetView can output only 32,728 characters at a time.

When you issue a REXX SAY instruction in a REXX command list for NetView, a 12-character header precedes the data displayed on the operator's screen. The header contains the one-character NetView message type of the message (HDRMTYPE()), followed by three blanks and the identifier of the domain under which the command list is running (APPLID()). For more information on HDRMTYPE() and APPLID(), see Chapter 4, "REXX Functions Provided by NetView" on page 51.

Do not use MSGID() as the first item of output from a SAY instruction because the message will be processed as a regular NetView message. This can cause the message to be trapped by a TRAP instruction and can incorrectly satisfy a WAIT instruction.

Using the CALL Instruction

When you use the CALL instruction in REXX command lists for NetView, it is recommended that you enclose the command list you want to call within single quotes. You can call only REXX command lists with the CALL instruction. Any parameters to be passed to the called command list must be outside the quotes enclosing the name of the command list. If you want to avoid variable substitution for a parameter, you must enclose the parameter in quotes. For example, if you code the following CALL instruction to call an external command list named CLIST2:

```
CALL 'CLIST2' P1,P2,'RESOURCE PU1 INACTIVE'
```

and CLIST2 contained the following ARG statement,

```
ARG RES1 RES2 STATUS
```

then the RES1 and RES2 variables are assigned the current values of P1 and P2 when CLIST2 is called.

If you execute CLIST2 as a command from another command list, for example:

```
'CLIST2' P1,P2,'RESOURCE PU1 INACTIVE'
```

then CLIST2 receives the same values for the variables on the ARG statement, but the value of the ARG() function is set to 1.

NetView Restrictions on REXX Functions

This section describes the restrictions that apply when coding REXX functions in REXX command lists for NetView.

Note: Some REXX functions return different values depending on the operating system that the command list containing them is running under. For example, DATE() returns the current date in different formats depending on the operating system. The REXX functions provided by NetView return the same values regardless of the operating system.

Using the REXX LINESIZE Function

The REXX LINESIZE() function always returns the value 32,728 when used in REXX command lists for NetView.

Using the REXX STORAGE Function

REXX command lists for NetView cannot use the REXX STORAGE() function.

Using VM REXX Compression Tools

NetView does not support the use of any VM REXX compression tools. If you experience a problem with a command list that you compressed or optimized with a compression tool, test the same command list without using the tool before you report a problem to IBM.

Writing REXX Function Packages

You can write your own REXX function packages for NetView. The NetView program supplies two dummy directories to help you write function packages. One directory is for a user function package (DSIRXUFP), and the other directory is for a local function package (DSIRXLFP). See *REXX Reference* for instructions on coding a real directory and coding the interface to your function code. Link-edit the real directory and function code into load module DSIRXUFP for a user function package or DSIRXLFP for a local function package. As part of coding the interface to your function code, you need to use the NetView DSIRXEBS macro to obtain a new EVALBLOCK. See *Customization: Using Assembler* for information on the DSIRXEBS macro.

See *NetView Installation and Administration Guide* and *NetView Tuning Guide* for information on improving the performance of REXX function packages for NetView.

Changing the Environment Addressed by REXX Command Lists

REXX command lists for NetView use NetView as the default addressing environment. If you want to change the environment, use the REXX ADDRESS instruction. For example, if you want your command list to execute MVS subcommands, you must first change the addressing environment with an ADDRESS MVS instruction.

In ADDRESS MVS, you can use the following TSO/E REXX commands:

- DELSTACK
- NEWSTACK
- QSTACK
- QBUF
- QELEM
- EXECIO
- MAKEBUF
- DROPBUF
- SUBCOM
- TS
- TE.

See *TSO/E REXX Reference* for more information on these commands.

Note: REXX command lists for NetView do not support ADDRESS ATTACH or ADDRESS LINK.

Using the TSO/E EXECIO Command

If you use the TSO/E REXX EXECIO command in a command list, code the command list so that it issues an EXECIO command with the FINIS option before the command list completes its processing. If the command list using EXECIO is part of a nested chain of command lists, code the chain so that one of the command lists issues EXECIO with the FINIS option before the chain of command lists completes processing. This enables you to use SIGNAL ON HALT to try to recover if EXECIO with the FINIS option encounters an error closing a file. If the EXECIO command encounters an error, it sets the RC variable to a non-zero return code. See *TSO/E REXX Reference* for information on return codes used by the EXECIO command.

See "TYPE Example" on page 58 and "PRINT Example" on page 60 for examples of how EXECIO can be used in a REXX command list.

Note: NetView running on a VM system does not support EXECIO.

Using the NetView ALLOCATE and FREE Commands

The NetView program provides the ALLOCATE and FREE commands to enable you to dynamically allocate and deallocate data sets from NetView. NetView supports these commands on MVS systems only. The commands closely resemble the TSO/E commands for allocating and deallocating data sets. However, because these commands are provided by the NetView program, you do not need to use the ADDRESS MVS instruction when using these commands in a command list. Simply enclose the commands in quotes as you do for other NetView commands. The TYPE, TYPEIT, and PRINT examples in Chapter 5, "Examples of REXX Command Lists for NetView" on page 57 use the NetView ALLOCATE command. See *NetView Operation* for the syntax of the NetView ALLOCATE and FREE commands.

Nesting REXX Command Lists from Assembler, C, or PL/I

Each time a REXX command list is nested by an Assembler, C, or PL/I command processor, a unique REXX environment is created for that REXX command list. The data stacks from any previous REXX command lists in the nested chain are not passed to the additional unique environment. For example, if a REXX command list calls a PL/I command processor and the PL/I command processor calls another REXX command list, then an additional unique REXX environment is created for the second REXX command list.

The number of unique REXX environments that can be created at one time is limited by TSO/E REXX. Therefore, your nested chains are also limited in the number of REXX command lists that can be called by the Assembler, C, or PL/I command processors. See *REXX Reference* for information on the maximum number of environments in an address space.

Parsing in REXX Command Lists

In a REXX command list, you can parse character strings using either the REXX PARSE instruction or the NetView PARSEL2R command.

See *REXX Reference* for information on the REXX PARSE instruction.

See "Parsing Variables with PARSEL2R" on page 144 for information on using the NetView PARSEL2R command. When you use PARSEL2R in a REXX command list, enclose the command in quotes to avoid variable substitution. For example:

```
TITLE = 'DON''T TREAD(ROUGHLY) ON ME, PLEASE'
'PARSEL2R TITLE A1 A2 A3'
```

Tracing REXX Command Lists

During the creation of a REXX command list for the NetView program, you can see how the REXX interpreter evaluates an expression using the TRACE START (TS) command. The TS command sets an indicator that is checked by the REXX interpreter when it starts to interpret a command list or when control is returned to a command list after a nested command list completes execution. Figure 16 shows the syntax of the TS command.

```
TS
```

Figure 16. TS Command Syntax

After receiving the following message on an MVS system:

```
CNM431I REXX INTERACTIVE TRACE.  ENTER 'GO TRACE OFF' TO END TRACE,
ENTER 'GO' TO CONTINUE.
```

or, after receiving the following message on a VM system:

```
+++Interactive trace.  TRACE OFF to end debug, ENTER to continue.+++
```

enter GO to continue tracing, or enter GO TRACE OFF to end the trace. Also, after receiving one of the messages indicating a trace point has been reached, you can enter GO followed by a command or instruction you want to execute at that point in the command list. For example, to set a variable to a certain value at that point in the command list you could enter:

```
GO X=5
```

Or, to display the current value of a variable you could enter:

```
GO SAY 'VAR1 CURRENTLY IS 'VAR1
```

If you enter a TS command but decide that you do not want to run the trace before it begins, use the TE command to cancel the trace. You can also use the TE command to end a trace that is not interactive. Figure 17 shows the syntax of the TE command.

```
TE
```

Figure 17. TE Command Syntax

For more information on TS and TE, see *NetView Operation.*

Return Codes in REXX Command Lists

The REXX return code variable, RC, is set after execution of each instruction, command, or nested command list. You can use the EXIT statement in a nested command list to end the command list and set RC to a value that is passed back to the calling command list. RC is not given an initial value when a command list begins.

Possible RC values and their meanings are:

Values	**Meaning**
0	No error. The command, instruction, or nested command list completed successfully.
-1	The command, instruction, or nested command list encountered an error. The -1 return code passes control to the FAILURE label if SIGNAL ON FAILURE is coded.
-3	The command or nested command list is not in the operator's scope of commands. The -3 return code passes control to the FAILURE label if SIGNAL ON FAILURE is coded.
-5	The command list has been canceled. The -5 return code passes control to the HALT label if SIGNAL ON HALT is coded.
Others	Other return codes are set by individual commands, instructions, or nested command lists.

Note: See "Recovering from Errors in REXX Command Lists" for more information on using the SIGNAL instruction with NetView.

Recovering from Errors in REXX Command Lists

When an error occurs in a REXX command list, you can use the SIGNAL instruction to cause processing to continue at a certain point. A command list can encounter an error for the following reasons:

- An error exists in the coding of the command list itself.
- The command list is part of a nested chain, and one of the other command lists in the chain contains an error that is passed back to the command list.
- An operator enters a command that causes an error in the command list.

If an error occurs, the SIGNAL instruction passes control to another part of the command list. Depending on the error condition, the SIGNAL instruction can pass control to three different labels in the command list:

- SIGNAL ON FAILURE passes control to a label named FAILURE when the error condition results in a negative return code. The only negative return codes returned by NetView are −1 and −3. However, if your command list calls user written commands, control is passed to FAILURE when any negative return code, except −5, is returned.

  If your command list recovers from the error, you can return the appropriate return code to the calling command list. If your command list does not recover from the error, pass the failure to the calling command list with EXIT −1.

  Note: Regardless of whether SIGNAL ON FAILURE is coded, NetView only passes the halt condition to the calling command list if you code EXIT −1.

- SIGNAL ON ERROR passes control to a label named ERROR when any command or function in your command list returns a positive return code. Control is also passed to ERROR when SIGNAL ON FAILURE is not coded and a command or function returns any negative return code except −5.

  The return code you pass to any command list that nested your command list should reflect the severity of the error. A zero (0) return code is recognized by all NetView commands as an indication of successful completion, while all positive return codes indicate that an error occurred. The higher the return code, the greater the severity of the error.

- SIGNAL ON HALT passes control to a label named HALT when the command list is canceled. A command list is canceled when:

  - A RESET NORMAL command is executed on the current operator task while your command list is running.
  - A CLOSE IMMED command is executed on any task in your NetView while your command list is running. The command list continues processing as long as it does not issue NetView commands.
  - During SNA sessions, an operator presses the ATTN key while your command list is running.
  - A command issued by your command list is canceled or returns a return code of −5.
  - The operator's terminal session is lost for any reason, including the operator entering the LOGOFF command, while the command list is running. The command list continues processing as long as it does not issue NetView commands.

  To pass the HALT condition to any command list that nested your command list, end the command list with EXIT −5.

  Notes:

  1. If you do not code SIGNAL ON HALT, NetView passes the halt condition to the command list that nested your command list.
  2. Whenever you call another REXX command list as a function or subroutine, the following statement of the command list should test the RESULT variable for the −5 cancel condition.

For more information on the SIGNAL instruction, see *REXX Reference*.

Chapter 3. REXX Instructions Provided by NetView

This chapter describes the instructions used in REXX command lists for NetView. These instructions are provided as part of the NetView program so REXX command lists can perform specific NetView activities. Because these instructions are provided by NetView and are not standard REXX instructions, they can only be used in command lists that execute in a NetView environment. These instructions do not function in any REXX EXECs that are executing in non-NetView environments. The instructions provided by NetView are internal commands, which means they can only be used in command lists, and are not available for entry at operator consoles.

This chapter contains a description of each REXX instruction provided by the NetView program, how the instruction works, and how to code the instruction in a REXX command list. For more information on REXX syntax rules, as well as information on other REXX instructions, see *REXX User's Guide* or *REXX Reference*.

See Appendix C, "Comparison of REXX and NetView Command List Language" on page 185 for a complete list of the REXX instructions that are equivalent to NetView command list language control statements. This list includes both instructions provided by NetView and instructions provided by REXX itself. See "Examples Comparing REXX and NetView Command List Language" on page 193 for examples of command lists written in the NetView command list language and the equivalent REXX command lists.

The REXX instructions provided by the NetView program are:

- TRAP
- WAIT
- WAIT CONTINUE
- MSGREAD
- FLUSHQ
- GLOBALV.

The TRAP, WAIT, WAIT CONTINUE, and MSGREAD instructions are used in a command list to monitor the operator station task (OST) for specific messages or wait for a specified period of time.

You can use the TRAP instruction to define the messages for which the command list should wait. When a TRAP instruction is issued, NetView begins monitoring the operator task for an occurrence of a specified message. If the message is received, it is stored in a message queue.

When a WAIT instruction is issued, the command list stops processing until one or more of the messages specified on the TRAP instruction are received or until the specified period of time has elapsed.

The WAIT CONTINUE instruction causes the command list to wait for additional messages or the remainder of the specified period of time before resuming command list processing.

If the operator task receives any of the messages specified on a TRAP instruction, you can use the MSGREAD instruction to read the trapped messages from the message queue. The command list can then take action based on the content of each message.

The FLUSHQ instruction is used to remove all trapped messages from the message queue.

"Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE" on page 42 contains a command list that shows how the TRAP, WAIT, MSGREAD, and WAIT CONTINUE instructions are used.

The GLOBALV instruction allows you to set and retrieve task and common global variables in REXX command lists. "Examples of Command Lists that Set, Retrieve, and Update Task Global Variables" on page 47 contains two command lists that show how the GLOBALV instruction is used.

REXX TRAP Instruction

Use the TRAP instruction to define messages that are to be trapped and to specify whether the messages should be displayed to the operator once they are trapped. The TRAP instruction can also be used to remove all messages from the list of messages to be trapped.

The TRAP instruction causes NetView to monitor the operator task for specified messages. If the messages occur, they are trapped and added to the message queue. Trapped messages can then be read using the MSGREAD instruction and can satisfy a WAIT instruction. See "REXX MSGREAD Instruction" on page 40 and "REXX WAIT Instruction" on page 36 for information on how these instructions are used with the TRAP instruction.

The TRAP instruction does not clear the queue of messages trapped by the previous TRAP. To clear the message queue, issue a FLUSHQ instruction. See "REXX FLUSHQ Instruction" on page 42 for more information on FLUSHQ.

Figure 18 shows the syntax of the TRAP instruction.

Notes:

1. The operands must be entered in the order shown in Figure 18.
2. The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'TRAP  [[AND ]SUPPRESS|DISPLAY]
       [MORE|ONLY]
       MESSAGES token [,...]'

                 OR

'TRAP  NO MESSAGES'
```

Figure 18. TRAP Instruction Syntax

AND

can be used to make the TRAP instruction syntax more readable. AND can only be used between TRAP and SUPPRESS or TRAP and DISPLAY.

SUPPRESS|DISPLAY

SUPPRESS	indicates that any messages matching the specified *tokens* should not be displayed on the operator's screen when received by NetView.
DISPLAY	indicates that any messages matching the specified *tokens* should be displayed on the operator's screen when received by NetView.

MORE|ONLY

MORE	indicates that the specified *tokens* should be added to the list of *tokens* that was specified on a previous TRAP instruction. **Note:** Each message in the resulting list retains its own individual setting of the SUPPRESS|DISPLAY option. This allows some messages in the list to be suppressed while others are displayed.
ONLY	indicates that the specified *tokens* replace the list of *tokens* specified on a previous TRAP instruction.

MESSAGES

indicates that the trapped items are messages.

token [,...]

1 to 10 characters that identify the first token of the message or messages to be trapped. Optionally, you can identify the domain of a message to be trapped. If a domain identifier is specified, it precedes the token and is separated from the token by a period (*domainid.token*). You can also use an asterisk (*) to indicate that you are specifying a partial domain identifier or token. If you do not specify a domain identifier, the message being trapped can be from any domain.

Note: If you specify SUPPRESS and use an asterisk when specifying a domain identifier or token, no messages matching the specified domain or token will be displayed to the operator. For example, if you have a command list that traps and suppresses all messages, no REXX messages are displayed to the operator when the command list is run. This includes any REXX trace output and messages if you are using the REXX TRACE instruction to debug the command list.

Following are examples of how you can specify the messages you want to trap:

domainid.token	The command list traps any message whose domain identifier matches the 1- to 5-character *domainid* and whose first token matches *token*.
dom.token*	The command list traps any message whose domain identifier matches the partial domain identifier specified by *dom** and whose first token matches *token*. For example, NCCF*.DSI463I means trap a DSI463I message from any domain with an identifier that starts with NCCF (such as NCCFA or NCCFB).
**.token*	The command list traps any message whose first token matches *token*. The message can be from any domain.

token	The command list traps any message whose first token matches *token*. The message can be from any domain.
*tok**	The command list traps any message whose first token matches the partial token specified by *tok**. For example, DSI* means trap any messages whose first token begins with DSI (such as DSI463I or DSI386I).
*	The command list traps all messages. **Note:** Use caution when coding * or *.* with SUPPRESS. This causes no messages to be displayed to the operator.

Multi-line messages such as multi-line-write-to-operator (MLWTO) are treated as one message. Therefore, only the message identifier of the first message in a multi-line message is available to the TRAP, and the TRAP can be satisfied only by that message identifier. Use GETMSIZE, GETMTYPE, and GETMLINE to access the other messages in a multi-line message. See "Working with Multi-Line Messages" on page 151 for more information on multi-line messages.

NO
indicates that NetView should stop trapping the messages that were specified on the previous TRAP instruction.

TRAP sets the value of RC to indicate the processing results, as follows:

Code	Meaning
0	Successful completion
4	TRAP only allowed from HLL or REXX command lists
12	Syntax error
144	Not in OST or NNT
18004	DSIMRBLD invalid parameter
18008	DSIMRBLD storage failure.

A sample command list, showing how the TRAP instruction can be used, appears in "Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE" on page 42. Also, several of the examples in Chapter 5, "Examples of REXX Command Lists for NetView" on page 57 use TRAP, WAIT, and MSGREAD.

Using TRAP in Nested REXX Command Lists

You can code a TRAP instruction in a REXX command list that contains nested command lists. Nested REXX command lists can also contain a TRAP instruction. However, trapped messages are available only to the command list that issued the TRAP instruction.

REXX WAIT Instruction

The WAIT instruction causes a command list to temporarily suspend processing until a specified event occurs. The event can be one or more messages, a certain period of time, or both. The first event that occurs satisfies the WAIT

Note: There are times when you cannot use WAIT. Before coding WAIT, read "Primary POI Task Restrictions" on page 17. Do not code WAIT with service point service commands. For additional information, see Chapter 10, "Service Point Command Service Commands" on page 165.

Figure 19 on page 37 shows the syntax of the WAIT instruction.

Notes:

1. The operands must be entered in the order shown in Figure 19 on page 37.
2. You must code a time interval (*n* SECONDS or *n* MINUTES), MESSAGES, or both.
3. The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'WAIT    [n [SECONDS|MINUTES]]
         [FOR [MESSAGES]]'
```

Figure 19. WAIT Instruction Syntax

n

means the command list waits *n* SECONDS or MINUTES before resuming command list processing.

When SECONDS is specified, the value of *n* can be from 0 to 2,678,400. When MINUTES is specified, the value of *n* can be from 0 to 44,640. The equivalent of 2,678,400 seconds or 44,640 minutes is 31 days.

SECONDS|MINUTES

SECONDS means the command list waits *n* seconds before resuming processing.

MINUTES means the command list waits *n* minutes before resuming processing.

FOR

can be used to make the WAIT instruction syntax more readable.

MESSAGES

means the command list waits for a trapped message to be added to the message queue before resuming processing. The specific messages for which the command list should wait are defined using the TRAP instruction. If a time interval is also specified, the command list waits for up to that amount of time and then resumes even if one of the specified messages has not been received. If one of the specified messages is already on the message queue, the command list resumes without waiting. See "REXX TRAP Instruction" on page 34.

When NetView encounters a WAIT instruction in a REXX command list, the letter W is displayed in the upper right-hand corner of the current command facility panel if the screen is refreshed as the result of a message being received or the ENTER key being pressed. This notifies the operator that the command list has halted its processing and is waiting for a message or group of messages or for a specific period of time. The first event that occurs satisfies the WAIT.

Checking the Result of a WAIT Instruction

The NetView event that satisfied the WAIT is determined by the value of the REXX EVENT() function. The REXX command list can check EVENT() and take appropriate action based on its value. The possible values for EVENT() are:

M The message the command list is waiting for has arrived. The message can be read using the MSGREAD instruction.

T The time period for which the command list was waiting has expired, and processing is resumed.

G The operator entered the GO command, and processing is resumed.

E The WAIT or TRAP instructions were not coded correctly. For example, the operands were not entered in the correct order or a WAIT instruction was issued without a matching TRAP instruction. The command list resumes processing.

If a WAIT instruction is never issued in a command list, the value of the EVENT() function is set to null.

WAIT sets the value of RC to indicate the results of processing, as follows:

Code	Meaning
-1	DSIGET failure
0	Successful completion
4	WAIT only allowed from HLL or REXX command lists
8	Too many operands
12	Syntax error
144	Not in OST or NNT
152	WAIT issued without a previous TRAP
248	WAIT CONTINUE issued without a previous valid WAIT.

A command list, showing how the WAIT instruction can be used, appears in "Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE" on page 42. Also, several of the examples in Chapter 5, "Examples of REXX Command Lists for NetView" on page 57 use TRAP, WAIT, and MSGREAD.

Continuing to Wait for Additional Messages

You can code a WAIT CONTINUE instruction in your REXX command list to cause the command list to continue waiting before resuming command list processing.

Figure 20 shows the syntax of the WAIT CONTINUE instruction.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'WAIT CONTINUE'
```

Figure 20. WAIT CONTINUE Instruction Syntax

The options specified on the previous TRAP and WAIT instructions remain in effect for the WAIT CONTINUE instruction. When processing resumes, the next instruction after the WAIT CONTINUE is executed.

For example, if you code the following instructions in a command list and one of the messages specified on the TRAP instruction is received in 5 seconds:

```
'TRAP AND SUPPRESS MESSAGES MSG1, MSG2, MSG3'
'WAIT 20 SECONDS FOR MESSAGES'
'MSGREAD'
   •
   •
   •
'WAIT CONTINUE'
'MSGREAD'
```

then the WAIT instruction is satisfied. The WAIT CONTINUE instruction waits up to 15 seconds (the difference between the 20 seconds specified on the WAIT instruction and the 5 seconds already used to satisfy the WAIT instruction) to receive one of the messages specified on the TRAP instruction before resuming command list processing. When processing resumes, the MSGREAD instruction following WAIT CONTINUE is executed.

A sample command, showing how the WAIT CONTINUE instruction is used, appears in "Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE" on page 42. Also, several of the examples in Chapter 5, "Examples of REXX Command Lists for NetView" on page 57 contain examples of command lists that use WAIT CONTINUE.

Using NetView Commands with WAIT

When a REXX command list is in a wait or pause state, operator commands that are entered can be deferred. Whether the commands are deferred is based on the NetView DEFAULTS, OVERRIDE, and CMD commands. See *NetView Operation* for information on these commands.

The GO, STACK, UNSTACK, and RESET commands affect the processing of command lists in a wait state as follows:

- GO ends a WAIT.
- STACK suspends command list processing and causes any commands that have been deferred to be processed. You can enter any command or command list for normal processing while a command list is suspended. The WAIT is not suspended, and events are still matched as they occur. The W, if present, does not remain in the upper right corner of the NetView screen. The GO command is rejected until the command list resumes processing.
- UNSTACK resumes command list processing. The WAIT resumes processing events that were matched while the command list was suspended. The WAIT does not resume after expiration of a specified period of time if, while the command list was suspended, you ran another command list that issues a WAIT or &WAIT with a specified period of time.
- RESET ends a command list, as well as all command lists related to it by nesting. RESET also drives HALT when SIGNAL ON HALT is coded.

For more information on the GO, STACK, UNSTACK, and RESET commands, see *NetView Operation*.

Using WAIT in Nested Command Lists

REXX command lists that call other command lists or are called by other command lists can issue a WAIT instruction. The following considerations apply when using WAIT with nested command lists:

- Messages that arrive for the waiting command list are queued until the nested command list has finished processing.
- If you specify the same message number on TRAP instructions in both the waiting and nested command lists, the message satisfies the WAIT in the nested command list.

REXX MSGREAD Instruction

The MSGREAD instruction causes NetView to read a trapped message from the message queue. The command list can then take action based on the message. See "REXX TRAP Instruction" on page 34 and "REXX WAIT Instruction" on page 36 for information on how these instructions are used with the MSGREAD instruction.

When a MSGREAD instruction is issued, the oldest message in the queue is read and removed from the queue. The message that is read is used to set MSGID(), MSGCNT(), MSGORIGN(), and MSGSTR(). The message text is used to set the parameter variables MSGVAR(1) - MSGVAR(31). See "Functions Set by MSGREAD" for information about the variables set by MSGREAD.

Figure 21 shows the syntax of the MSGREAD instruction.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'MSGREAD'
```

Figure 21. MSGREAD Instruction Syntax

MSGREAD sets the value of RC to indicate the results of processing, as follows:

Code	Meaning
-2	Syntax error
-1	DSIGET failure
0	Successful completion
4	No messages in queue.

A sample command list, showing how the MSGREAD instruction is used, appears in "Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE" on page 42. Also, several of the examples in Chapter 5, "Examples of REXX Command Lists for NetView" on page 57 use TRAP, WAIT, and MSGREAD.

Functions Set by MSGREAD

NetView sets the values of the MSGCNT(), MSGID(), MSGORIGN(), MSGSTR(), MSGTYP(), and MSGVAR(*n*) functions based on the information contained in a message read by a MSGREAD instruction.

MSGCNT()
becomes the number of elements in the text of MSGSTR().

MSGID()
becomes the message identifier of the last message read. The message identifier is the first token of the message (up to 10 characters). If the first token is longer than 10 characters, MSGID() uses only the first 10 characters.

MSGORIGN()
becomes the name of the domain from which the message was sent.

MSGSTR()
becomes the message text exactly as it is received by NetView. MSGSTR() does not include the message identifier (the token used by the MSGID() function).

MSGTYP()
becomes the system message type of the last message read.

MSGVAR(*n*)
NetView changes the values of the MSGVAR(1) - MSGVAR(31) functions to reflect the text of the message. Each MSGVAR(*n*) function is set to a token of the message. Tokens are delimited by commas, apostrophes, or blanks. MSGVAR(1) is set to the first token following the message identifier (the token used by the MSGID() function). MSGVAR(2) is set to the next token to the right of MSGVAR(1), and so on up to a maximum of 31 variables.

For example, if MSGREAD is used to read the following message:

```
DSI008I SPAN1  NOT ACTIVE
```

the functions are set as follows:

Variable	Value
MSGORIGN()	DOM01
MSGID()	DSI008I
MSGSTR()	SPAN1 NOT ACTIVE
MSGCNT()	3
MSGVAR(1)	SPAN1
MSGVAR(2)	NOT
MSGVAR(3)	ACTIVE
MSGVAR(4)-MSGVAR(31)	null

Notes:

1. If MSGREAD reads a a multi-line message, the functions are set according to the first line of the message. See "Working with Multi-Line Messages" on page 151 for information concerning working with multi-line messages.
2. The MSGVAR(1) - MSGVAR(31) functions can be given values when a command list is invoked in the same way the &1 - &31 NetView command list language parameter variables can. (See "Parameter Variables" on page 77.) If MSGVAR(1) - MSGVAR(31) are given values when the command list is invoked, save those values in variables before issuing a MSGREAD instruction. This lets you use the values after MSGREAD changes them.
3. After using MSGREAD, save the values of the message functions in variables before issuing another MSGREAD instruction. This lets you use the values after another MSGREAD changes them.
4. Before a MSGREAD instruction is issued, the values of MSGID(), MSGORIGN(), MSGSTR(), and MSGTYP() are null. The value of MSGCNT() is 0. The MSGVAR(*n*) functions retain any values they were given when the command list was run.
5. If you issue a MSGREAD instruction when the message queue is empty, the values of MSGID(), MSGORIGN(), MSGSTR(), MSGTYP(), and MSGVAR(*n*) are set to null. The value of MSGCNT() is 0.

REXX FLUSHQ Instruction

The FLUSHQ instruction is used to remove all trapped messages from the message queue, including the message currently being processed.

Figure 22 shows the syntax of the FLUSHQ instruction.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'FLUSHQ [MESSAGES]'
```

Figure 22. FLUSHQ Instruction Syntax

Because the TRAP instruction does not clear the queue of messages trapped by a previous TRAP, you should issue a FLUSHQ instruction between multiple TRAP instructions coded in the same command list.

FLUSHQ sets the value of RC to indicate the results of processing, as follows:

Code	**Meaning**
-1	Syntax error
0	Successful completion.

Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE

This section contains an example of a command list that illustrates how the TRAP, WAIT, MSGREAD, and WAIT CONTINUE instructions can be used in REXX command lists.

Figure 23 is a REXX command list named ACTLU. ACTLU issues a VTAM command to activate a node specified by the user. Messages sent as the result of the activate command are trapped, and the operator is notified of the result of the command list. The comments in the example explain how the command list works.

```
/***********************************************************************/
/*  ACTLU COMMAND LIST                                                 */
/*                                                                     */
/*  FUNCTION : TO ACTIVATE A VTAM NODE.                                */
/*  INPUT    : 1 PARAMETER, THE NAME OF THE NODE.                      */
/***********************************************************************/
IF MSGVAR(1) = '' THEN                 /* NO FIRST PARAMETER  ?        */
  DO                                   /*  THEN ISSUE REQUEST          */
   SAY 'PLEASE ENTER "GO NODENAME"', /* REQUEST NODENAME FROM USER   */
     'TO CONTINUE OR "GO STOP" TO',  /* OR, ALLOW USER TO STOP       */
     'STOP'
   PARSE PULL NODE                     /* NODE = NODENAME OR STOP      */
  END                                  /*  THEN ISSUE REQUEST          */
ELSE                                   /* FIRST PARAMETER EXISTS       */
  NODE = MSGVAR(1)                     /* ASSUME IT IS A NODE NAME     */
```

Figure 23 (Part 1 of 2). Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE

```
IF NODE¬='STOP' THEN                /* IF USER DID NOT CHOOSE STOP   */
  DO                               /* PROCESS NODENAME              */
    'TRAP AND SUPPRESS ONLY MESSAGES IST* ' /* TRAP ALL VTAM MSGS   */
    'V NET,ACT,ID='NODE            /* ISSUE VTAM ACTIVATE FOR NODE  */
    RCSAVE=RC                      /* SAVE RETURN CODE IN RCSAVE    */
    'WAIT 30 SECONDS FOR MESSAGES' /* WAIT FOR 30 SECONDS           */
    SELECT
      WHEN (EVENT()='M') THEN      /* OUT OF WAIT - IS THERE A MSG? */
        DO                         /* PROCESS TRAPPED MESSAGE       */
          'MSGREAD'                /* READ IN 1ST MESSAGE           */
          DO WHILE (RC=0)          /* IF RC¬=0 THEN NO MORE MSGS    */
            SELECT                 /* DETERMINE WHICH MESSAGE HIT   */
              WHEN (MSGID() = 'IST061I')          /* NODE NOT FOUND */
                THEN SAY '==> LU UNKNOWN',        /* INFORM USER    */
                         'TO YOUR VTAM <=='
              WHEN (MSGID() = 'IST093I')          /* NODE NOW ACTIVE */
                THEN SAY '==> TERMINAL',          /* INFORM USER    */
                  MSGVAR(1)' NOW',
                  MSGVAR(2) '<=='
                OTHERWISE          /* IGNORE THE VTAM MESSAGE       */
                  IF RCSAVE=0 THEN
                    'WAIT CONTINUE'  /* CONTINUE WAITING            */
                  ELSE DO
                    SAY 'ERROR PROCESSING', /* ERROR ENCOUNTERED ?  */
                        'ACTIVATE COMMAND'  /* INFORM USER          */
                    SAY MSGSTR()            /* DISPLAY MESSAGE TEXT */
                  END
            END                    /* OF SELECT FOR IST061I/IST093I */
            'MSGREAD'              /* READ IN THE NEXT MESSAGE      */
          END                      /* DO WHILE RC=0, LOOP BACK      */
        END                        /* PROCESS TRAPPED MESSAGE DO    */
                                   /* OUT OF DO WHILE               */

      WHEN (EVENT()='E'|SAVERC¬=0) THEN /* CHECK FOR ERROR OR       */
                                   /* TIMEOUT EVENTS                */
        SAY 'ERROR PROCESSING',    /* ERROR ENCOUNTERED?            */
            'ACTIVATE COMMAND'     /* INFORM USER                   */
      WHEN (EVENT()='T') THEN      /* WAIT TIMEOUT ENCOUNTERED ?    */
        SAY 'NO RESPONSE TO',      /* INFORM USER                   */
            'ACTLU CLIST FOR 'NODE
      OTHERWISE                    /* NO-OP                         */
    END                            /* OF SELECT FOR ERROR/TIMEOUT   */
  END                              /* IF NODE¬='STOP' PROCESSING    */
```

Figure 23 (Part 2 of 2). Sample Command List Using TRAP, WAIT, MSGREAD, and WAIT CONTINUE

REXX GLOBALV Instruction

The REXX GLOBALV instruction allows you to set and retrieve global variables in REXX command lists. Global variables allow multiple command lists, regardless of their language, to share a common set of values. There are two types of global variables: task and common. Task global variables let you set and retrieve any number of global variables in command lists running under a particular NetView task. Common global variables let you set and retrieve global variables in command lists running under any NetView task.

When you set a global variable using the GLOBALV instruction, NetView places that variable in a global variable dictionary. If the variable is a task global variable, it is stored in the global variable dictionary for the particular task under which the command list is running. Only command lists running under the same task can access that task's global variable dictionary.

When you want to use a global variable in a command list, the GLOBALV instruction retrieves the global variable from the appropriate dictionary and places it in a REXX variable of the same name. You can then perform arithmetic operations or other manipulations on the REXX variable to suit the needs of your command list without affecting the value of the global variable in the dictionary.

Command lists written in the NetView command list language use the &CGLOBAL and &TGLOBAL control statements to access global variables.

When setting global variables using the GLOBALV instruction, follow these guidelines:

- The global variable name can be 1 to 31 alphanumeric characters. Valid alphanumeric characters are A-Z, 0-9, #, @, ¢, $, _, !, ?, and period (.). The first character cannot be a number or a period (.).

  Note: The NetView command list language does not allow variable names to contain a period, _, ¢, !, or ? and limits global variables to a length of 11 characters. Therefore, if you want global variables you create in a REXX command list to also be accessible to command lists written in the NetView command list language, make sure the global variable names are from 1 to 11 characters in length and do not contain a period, _, ¢, !, or ?.

- If more than one global variable is specified on the GLOBALV instruction, the variable names must be delimited by commas or blanks.

- The value of the global variable can be 255 characters long. For double-byte character sets the maximum number of double-byte characters between the shift-out and shift-in is 126.

- You can give global variables a numerical value between -2147483647 and 2147483647.

See "Scope of Variables in Command Lists" on page 127 for information about the scope of global variables in command lists.

Setting Task Global Variables in REXX Command Lists

To set a task global variable from a REXX command list, use the GLOBALV PUTT instruction. The GLOBALV PUTT instruction creates a task global variable with the specified variable name and places it in the task global variable dictionary for the task under which the command list is running.

Figure 24 shows how to code a GLOBALV instruction to set a task global variable.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'GLOBALV        PUTT variable [,...]'
```

Figure 24. GLOBALV PUTT Instruction Syntax

PUTT
: indicates that task global variables with the specified variable names should be set.

variable [,...]
: the 1- to 31-character names of the task global variables to be set. See page 44 for a list of the characters that can be used for a variable name.

When a GLOBALV PUTT instruction is processed, the NetView program determines whether a REXX variable already exists with the specified variable name. If a REXX variable with the same name exists, the task global variable is created, and its value is set to the value currently assigned to the REXX variable.

If no REXX variable exists with the variable name specified, a task global variable with that name is created. The value of the task global variable is set to null.

If a task global variable already exists with the specified variable name, the value of the task global variable is updated in the task global dictionary.

The command lists in "Examples of Command Lists that Set, Retrieve, and Update Task Global Variables" on page 47 show how to set, retrieve, and update a task global variable.

Retrieving Task Global Variables in REXX Command Lists

To access a task global variable in a REXX command list, use the GLOBALV GETT instruction. The GLOBALV GETT instruction retrieves the value of the specified task global variable and assigns it to a REXX variable of the same name.

Figure 25 shows how to code a GLOBALV instruction to retrieve a task global variable.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'GLOBALV          GETT variable [,...]'
```

Figure 25. GLOBALV GETT Instruction Syntax

GETT
indicates that the task global variables with the specified variable names should be retrieved.

variable [,...]
the 1- to 31-character names of the task global variables to be retrieved.

When a GLOBALV GETT instruction is processed, the NetView program retrieves the current value of the specified task global variable from the task global variable dictionary and places it in a REXX variable of the same name. If a REXX variable with the same name does not already exist, the REXX variable is created, and its value is set to the current value of the task global variable.

If no task global variable with the specified variable name exists, the value of the REXX variable is set to null.

The command lists in "Examples of Command Lists that Set, Retrieve, and Update Task Global Variables" on page 47 show how to set, retrieve, and update a task global variable.

Examples of Command Lists that Set, Retrieve, and Update Task Global Variables

Figure 26 and Figure 27 are examples of REXX command lists that show how to set, retrieve, and update a task global variable. The first command list is named INITIALIZE1, and it executes the nested command list UPDATE1.

```
/* INITIALIZE1 COMMAND LIST */
/* THE FOLLOWING ASSIGNMENT STATEMENT GIVES THE REXX
    VARIABLE, TOM, A VALUE OF 5.  */
TOM = '5'
/* THE FOLLOWING STATEMENT SETS A TASK GLOBAL VARIABLE
   NAMED TOM.  THE VALUE OF TASK GLOBAL VARIABLE TOM IS 5
   BECAUSE THE VALUE OF REXX VARIABLE TOM IS 5.  */
'GLOBALV PUTT TOM'
/* THE FOLLOWING STATEMENT EXECUTES A NESTED COMMAND LIST NAMED
    UPDATE1.  */
UPDATE1
/* THE FOLLOWING STATEMENT RETRIEVES THE VALUE OF TASK
   GLOBAL VARIABLE TOM AND PLACES IT IN A REXX
   VARIABLE NAMED TOM.  */
'GLOBALV GETT TOM'
/* THE FOLLOWING STATEMENT WILL WRITE THE VALUE OF THE
   REXX VARIABLE TOM.  */
SAY 'TOM = 'TOM
EXIT
```

Figure 26. INITIALIZE1 Command List

```
/* UPDATE 1 COMMAND LIST */
/* THE FOLLOWING STATEMENT RETRIEVES THE VALUE OF TASK
   GLOBAL VARIABLE TOM INTO A REXX VARIABLE NAMED
   TOM.  */
'GLOBALV GETT TOM'
/* THE FOLLOWING STATEMENT CHECKS THE VALUE OF REXX
   VARIABLE TOM TO DETERMINE IF IT IS NULL.  */
IF TOM = '' THEN TOM = 0
/* THE FOLLOWING STATEMENT INCREMENTS REXX VARIABLE
   TOM BY 1.  */
TOM = TOM + 1
/* THE FOLLOWING STATEMENT SETS THE VALUE OF TASK
   GLOBAL VARIABLE TOM  */
'GLOBALV PUTT TOM'
EXIT
```

Figure 27. UPDATE1 Command List

Command list INITIALIZE1 creates a REXX variable called TOM and gives it a value of 5. A GLOBALV PUTT instruction is issued to set a task global variable named TOM. Since a REXX variable with the name TOM already exists with a value of 5, the value of task global variable TOM is also set to 5. INITIALIZE1 activates a nested command list named UPDATE1.

Command list UPDATE1 issues a GLOBALV GETT instruction to retrieve the current value of task global variable TOM into a REXX variable named TOM. The value of REXX variable TOM is checked to determine if it is null. If the value is null, its value must be set to 0. Because the current value of REXX variable TOM is 5, it is incremented by 1, making its current value 6. The GLOBALV PUTT instruction causes the value of task global variable TOM to be updated in the task global variable dictionary.

When UPDATE1 completes execution, control is returned to the INITIALIZE1 command list. INITIALIZE1 contains a GLOBALV GETT instruction to retrieve the current value of task global variable TOM and place it in a REXX variable named TOM. The value of REXX variable TOM is 6 because the current value of task global variable TOM is 6. The SAY instruction displays the current value of the REXX variable TOM.

Setting Common Global Variables in REXX Command Lists

To set a common global variable in a REXX command list, use the GLOBALV PUTC instruction. The GLOBALV PUTC instruction creates a common global variable with the specified variable name and places it in the common global variable dictionary.

Figure 28 shows how to code a GLOBALV instruction to set a common global variable.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

```
'GLOBALV      PUTC variable [,...]'
```

Figure 28. GLOBALV PUTC Instruction Syntax

PUTC
indicates that common global variables with the specified variable names should be set.

variable [,...]
the 1- to 31-character names of the common global variables to be set. See page 44 for a list of the characters that can be used for a variable name.

When a GLOBALV PUTC instruction is processed, the NetView program determines if a REXX variable already exists with the specified variable name. If a REXX variable with the same name already exists, the common global variable is created, and its value is set to the value currently assigned to the REXX variable.

If no REXX variable exists with the variable name specified, a common global variable with that name is created. The value of the common global variable is set to null.

If a common global variable already exists with the specified variable name, the value of the common global variable is updated in the common global dictionary.

Note: Be careful if you have more than one command list running under different tasks and accessing the same global variable. The last value that the variable is set to is the value that is retrieved by any command list accessing the variable. For example, a command list accesses a common global variable and then before that command list updates the variable, another command list running under a different task accesses the variable. If both command lists update the variable, the variable assumes the value given to it by the command list that updates it last. To avoid having a common global variable being used by different command lists at the same time, you should have all command lists that use the variable run under the same task.

You can use the NetView-supplied command lists UPDCGLOB and SETCGLOB to update and set common global variables under the PPT. See *NetView Operation* for information on using UPDCGLOB and SETCGLOB.

See "CHKOPNUM Example" on page 62 and "CHKRSTAT Example" on page 64 for examples of how to set common global variables.

Retrieving Common Global Variables in REXX Command Lists

To access a common global variable in a REXX command list, use the GLOBALV GETC instruction. The GLOBALV GETC instruction retrieves the value of the specified common global variable and assigns it to a local REXX variable of the same name.

Figure 29 shows how to code a GLOBALV instruction to retrieve a common global variable.

Note: The instruction is enclosed in single quotes to prevent variable substitution by REXX.

'GLOBALV **GETC** *variable* [,...]'

Figure 29. GLOBALV GETC Instruction Syntax

GETC
indicates that common global variables with the specified variable names should be retrieved.

variable [,...]
the 1- to 31-character names of the common global variables to be retrieved.

When a GLOBALV GETC instruction is processed, the NetView program retrieves the current value of the specified common global variable from the common global variable dictionary and places it in a REXX variable of the same name. If a REXX variable with the same name does not already exist, the REXX variable is created, and its value is set to the value currently assigned to the global variable.

If no common global variable with the specified variable name exists, the value of the REXX variable is set to null.

See "CHKOPNUM Example" on page 62, "CHKRSTAT Example" on page 64, and "DSPRSTAT Example" on page 66 for examples of how to retrieve common global variables.

Chapter 4. REXX Functions Provided by NetView

This chapter describes the functions used in REXX command lists for NetView. These functions are provided as part of the NetView program so that command lists written in REXX can perform specific NetView activities. Because these functions are provided by NetView and are not standard REXX functions, they can only be used in command lists that execute in a NetView environment. These functions do not execute in any REXX EXECs that are executing in non-NetView environments.

Note: You can improve the performance of your REXX command list by limiting the use of REXX functions provided by NetView. If the same function, provided by NetView, is used several times in the command list without a change in value, use the function once to set a local variable to the value of the function. After setting the REXX function provided by NetView to a local variable, use the local variable in place of the function. If the value of the function changes during execution of the command list, you need to use the function each time to access its current value. For more information on setting REXX variables, see "REXX GLOBALV Instruction" on page 44.

Included in this chapter is a description of each function provided by NetView, how it works and how to code the instruction in a REXX command list. For more information on REXX syntax rules, as well as information on other REXX functions, see *REXX User's Guide* or *REXX Reference*.

The functions provided by the NetView program are set based on system information. To use a function, you must place the function name in the REXX command list at the location where you want the information to be accessed. When the command list runs, NetView gives the correct values to each function.

The functions let you obtain information about the operating environment, test conditions in a command list, and take actions based on the results.

See Appendix C, "Comparison of REXX and NetView Command List Language" on page 185 for a complete list of the REXX functions that are equivalent to NetView command list language control variables. This list includes both functions provided by NetView and functions provided by REXX itself. See "Examples Comparing REXX and NetView Command List Language" on page 193 for examples of command lists written in the NetView command list language and the equivalent REXX command lists.

Session Information

APPLID()
becomes the application program identifier for the task under which the command list is running. APPLID() is always the NetView domain ID appended with a 3-character alphanumeric value assigned by NetView. For example, if your domain ID is PARIS, APPLID() might be PARIS001.

OPSYSTEM()
becomes a character string that indicates the operating system for which NetView was compiled. OPSYSTEM() can contain the following character values:

MVS/XA
VM.

TASK()

becomes the 3-character string PPT (primary POI task), OST (operator station task), or NNT (NetView-NetView task), depending on the task under which the command list is running. TASK() allows the same command list to run under any of these tasks, because the command list can test for the task type and process accordingly. For example, there are some restrictions for command lists running under the PPT. See "Primary POI Task Restrictions" on page 17.

VTAM()

becomes a character string that indicates the level of the access method used. The variable is returned in one of two formats depending on whether the level of the access method includes a modification number.

If the level of the access method does not include a modification number, the format of the variable is *VTvr*, where:

```
VT - indicates the access method is VTAM
v  - indicates the version number of the access method
r  - indicates the release number of the access method.
```

If the level of the access method does include a modification number, the format of the variable is *Vvrm*, where:

```
V - indicates the access method is VTAM
v - indicates the version number of the access method
r - indicates the release number of the access method
m - indicates the modification number of the access method.
```

For example, for VTAM Version 3 Release 2, the function would return a value of VT32. For VTAM Version 3 Release 1 Modification 1, the function would return a value of V311.

Note: The value of VTAM() is null if VTAM is not active.

Terminal Information

HCOPY()

becomes the name of the hard-copy log printer started by the operator. If there is no hard-copy printer for this operator, HCOPY() is null.

LU()

becomes the logical unit name for this operator terminal.

Operator Information

OPID()

becomes this operator's ID.

Command List Information

COMPNAME()

becomes the 16-byte name of the component running when the command list was initiated. For example, if command list HELP is initiated, COMPNAME() defines the active component so that the correct HELP command list is initiated. COMPNAME() can contain the following character values:

```
DSINCCF DSINCCF
DSINPDA
DSINLDM
DSIVIEW 1
DSIVIEW 2
DSIVIEW 3
DSIVIEW 4
DSIVIEW 5
DSIVIEW 6
DSIVIEW 7
DSIVIEW 8
DSIVIEW 9
DSISTATMONRESUME
DSILBROWSERESUME
DSIVIEW APPL1
```

PARMCNT()

becomes the number of parameter variables that were entered when a command list was initiated.

Message Processing Information

MSGCNT()

is the number of elements of text in the message string of the last message read by MSGREAD. MSGCNT() is used with MSGREAD and with the LINKPD command.

See "REXX MSGREAD Instruction" on page 40 for more information about using functions with MSGREAD.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

MSGID()

becomes the message identifier of the last message read by MSGREAD. The message identifier is the first token of the message (up to 10 characters). If the first token is longer than 10 characters, MSGID() uses only the first 10 characters. If a reply ID is sent with the message, it is not used as the first token. For an MLWTO message, MSGID() uses the first token of the first line of the first message. MSGID() is used in message automation, with MSGREAD, and with the LINKPD command. See Chapter 9, "Message Automation" on page 135 for more information about message automation.

See "REXX MSGREAD Instruction" on page 40 for more information about using functions with MSGREAD.

MSGORIGN()

is the domain where the last message read by MSGREAD originated. MSGORIGN() is used for message automation, with MSGREAD, and with the LINKPD command.

See Chapter 9, "Message Automation" on page 135 for more information about message automation.

See "REXX MSGREAD Instruction" on page 40 for more information about using functions with MSGREAD.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

MSGSTR()
is the message text of the last message read by MSGREAD. MSGSTR() does not include the message identifier (the token used by the MSGID() function). For an MLWTO message, MSGSTR becomes the message text of the first line of the message. MSGSTR() is used with MSGREAD and with the LINKPD command.

See "REXX MSGREAD Instruction" on page 40 for more information about using functions with MSGREAD.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

MSGVAR(*n*)
each MSGVAR(*n*) function is set to a token of the last message read by MSGREAD. MSGVAR(1) is set to the token following the message identifier (the token used by the MSGID() function). MSGVAR(2) is set to the next token to the right of MSGVAR(1), and so on, up to a maximum of 31 variables. MSGVAR(*n*) is used for message automation, with MSGREAD, and with the LINKPD command.

See Chapter 9, "Message Automation" on page 135 for more information about message automation.

See "REXX MSGREAD Instruction" on page 40 for more information about using functions with MSGREAD.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

The MSGVAR(*n*) functions can be given values when a command list is invoked in the same way the &1 - &31 NetView command list language parameter variables can. See "Parameter Variables" on page 77 for more information on NetView command list language parameter variables.

SESSID()
is the ID of the TAF session that sent the message. SESSID() is used in message automation and with WAIT. See Chapter 9, "Message Automation" on page 135 for more information about message automation.

Note: If TAF starts a session with a SESSID() equal to the domain ID, SESSID() will not be set correctly, and message automation may not work.

The remainder of this section contains descriptions of message information functions that are used only for message automation. More information about these functions can be found in *MVS Systems Programming Library: Systems Macros and Facilities, Vol. 2*.

Some of these message information functions are filled with values only after NetView receives a message through the subsystem interface (SSI). The message information functions that have null values until NetView receives a message through the SSI are:

- AREAID()
- DESC()
- JOBNAME()
- JOBNUM()

- MCSFLAG()
- MSGTYP()
- ROUTCDE()
- SMSGID()
- SYSCONID()
- SYSID().

AREAID()
provides a one-letter (A-Z) identifier for the area on the console screen that displays the message.

DESC()
provides the system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.

HDRMTYPE()
provides the 1-character NetView message type of the message. NetView message types are described in *NetView Customization: Using Assembler*.

Note: HDRMTYPE is a NetView-supplied message function.

JOBNAME()
provides the 1- to 8-character MVS JOB name identifier. Because the JOBNAME is the name of the job that originated the message, it may not always be the same as the name of the job to which the message is referring. For example, this can occur when MVS issues a message about the NetView job. Also, JOBNAME can contain the name of an initiator (instead of the actual jobname) when a job is started or terminated. If the message is issued during startup or termination, extract the job name from the message text rather than using the JOBNAME variable.

JOBNUM()
provides the 8-character MVS JOB number identifier. Depending on the MVS release, JOBNUM() can be a character string such as 'JOB   4', or simply a number such as '   4'.

Note: The appropriate number of blanks are imbedded within JOBNUM to ensure a total length of 8 characters.

LINETYPE()
provides the multi-line write-to-operator (MLWTO) line type, as follows:

C	The line is a message control line.
L	The line is a message label line.
D	The line is a message data line.
DE	The line is the last message data line.
E	The line is the last message line and contains no data.
blank	The message is a single-line message.

MCSFLAG()
provides the system message flags in a binary series of on (1) and off (0) codes corresponding to the following meanings:

first	Send message conditionally to console SYSCONID()
second	Send message unconditionally to console SYSCONID()
third	RESP
fourth	REPLY
fifth	BRDCST
sixth	HRDCPY only
seventh	NOTIME
eighth	NOCPY.

MSGTYP()
provides the system message type as three consecutive binary characters. An on character (1) in one of the positions corresponds to the following meanings:

first	SESS
second	JOBNAMES
third	STATUS.

REPLYID()
provides a 3-character reply identifier for WTOR command replies. See "WTOR" on page 140 for more information about the WTOR command.

ROUTCDE()
provides the system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.

SMSGID()
provides an 8-character value that identifies a particular instance of a message. This control variable is used by the DOM command to identify action messages to be removed from the display. See "DOM" on page 141 for more information about DOM.

SYSCONID()
provides the console number (in decimal) that is to receive the message.

SYSID()
provides an identifier for the MVS system that sent the message.

WTOREPLY()
is the reply sent by the operator in response to a WTOR command. See "WTOR" on page 140 for more information about the WTOR command.

Domain Information

NVCNT()
becomes the number of NetView domains with which you can establish a cross-domain session.

NVID(*n*)
returns the NetView domain identifier of a domain with which you can establish a cross-domain session. If an invalid domain identifier is specified in *n*, an error is returned. To obtain the local domain identifier, use the APPLID() function. APPLID() returns the local domain ID appended with a 3-character alphanumeric value assigned by NetView.

NVSTAT(*name*)
indicates whether you have an active session with a domain. If no *name* is specified or an invalid *name* is specified, an error is returned.

Chapter 5. Examples of REXX Command Lists for NetView

This section contains examples of REXX command lists written for NetView. These examples show how the instructions and functions provided by NetView and the standard REXX instructions and functions can be used together in REXX command lists executing in a NetView environment.

TYPE Example

```
/**********************************************************************/
/* TYPE COMMAND LIST                                                  */
/* -----------------                                                  */
/*                                                                    */
/* FUNCTION: THIS COMMAND LIST DISPLAYS MEMBERS OF A DATA SET AT THE  */
/*           (INVOKING) USER'S NETVIEW TERMINAL ONE LINE AT A TIME.   */
/*                                                                    */
/* INPUT PARMS: DATASETNAME = FULLY QUALIFIED DATA SET NAME           */
/*              (INCLUDING MEMBER NAME) TO DISPLAY AT THE TERMINAL.   */
/*                                                                    */
/* OUTPUT: LINE = EACH LINE WITHIN THE MEMBER SPECIFIED BY THE USER.  */
/**********************************************************************/
ARG DATASETNAME                              /* PARSE CLIST INPUT     */
IF DATASETNAME='' | PARMCNT() > 1 THEN       /* NO CLIST INPUT ?      */
  DO                                         /* NAME NOT SPECIFIED    */
    SAY 'INCORRECT SYNTAX USED.'             /* ISSUE ERROR MSG       */
    SAY 'CORRECT SYNTAX IS :  '              /* ISSUE HELP MSG        */
    SAY '     TYPE DATASET.NAME(MEMBER)  '   /* ISSUE HELP MSG        */
    RC=24                                    /* SET RETURN CODE       */
  END                                        /* NAME NOT SPECIFIED    */
ELSE                                         /* CORRECT NAME/SYNTAX   */
  DO                                         /* NAME WAS SPECIFIED    */
    'TRAP AND SUPPRESS ONLY MESSAGES *'      /* TRAP/SUPPRESS MSGS    */
    'ALLOCATE DA('DATASETNAME') SHR FREE'    /* ALLOC/CONNECT FILE    */
    'WAIT FOR MESSAGES'                      /* WAIT FOR MESSAGES     */
    'MSGREAD'                                /* READ A MESSAGE IN     */
    'TRAP NO MESSAGES'                       /* DISABLE TRAP MSGS     */
    IF (MSGID()¬='CNM272I') THEN             /* IS MSG CNM272I ?      */
      DO                                     /* ¬ CNM272I MSG         */
        SAY MSGID() MSGSTR()                 /* DISPLAY MESSAGE       */
      END                                    /* ¬ CNM272I MSG         */
    ELSE                                     /* MSG IS  CNM272I       */
      DO                                     /* PROCESS CNM272I MSG   */
        DDNAME = MSGVAR(1)                   /* SAVE DYNAMIC DDNAME   */
        ADDRESS MVS 'EXECIO 1 DISKR 'DDNAME  /* PUT 1ST LINE ON STACK */
        DO WHILE RC=0                        /* WHILE RC = 0          */
          PULL RECORD                        /* PULL LINE FROM STACK  */
          SAY SUBSTR(RECORD,1,68)            /* DISPLAY LINE TO USER  */
                                             /* PUT NEXT LINE ON STACK*/
          ADDRESS MVS 'EXECIO 1 DISKR 'DDNAME
        END                                  /* WHILE RC = 0          */
                                             /* PUT OUT COMPLETE MSG  */
        'MESSAGE 309I TYPE CLIST IS NOW FINISHED'
      END                                    /* PROCESS CNM272I MSG   */
  END                                        /* NAME WAS SPECIFIED    */
RETURN                                       /* RETURN TO CALLER/EXIT */
```

Figure 30. TYPE Example

TYPEIT Example

```
/************************************************************************/
/* TYPEIT COMMAND LIST                                                  */
/* -------------------                                                  */
/*                                                                      */
/* FUNCTION: THIS COMMAND LIST DISPLAYS MEMBERS OF A DATA SET AT THE    */
/*           (INVOKING) USER'S NETVIEW TERMINAL ONE LINE AT A TIME.     */
/*                                                                      */
/* INPUT PARMS DATASETNAME = FULLY QUALIFIED DATA SET NAME              */
/*           (INCLUDING MEMBER NAME) TO DISPLAY AT THE TERMINAL.        */
/*                                                                      */
/* OUTPUT: LINE = EACH LINE WITHIN THE MEMBER SPECIFIED BY THE USER.    */
/*                                                                      */
/* CLIST FLOW : (AFTER CHECKING FOR PROPER INPUT PARMS)                 */
/*  - ALLOCATE THE DATA SET & MEMBER SPECIFIED TO NETVIEW               */
/*  - READ ALL LINES OF MEMBER ONTO STACK WITH 'EXECIO * DISKR'         */
/*  - LOOP THROUGH STACK UNTIL NO MORE RECORDS EXIST <QUEUED()=0>       */
/*  - PUT OUT THE FIRST 68 CHARACTERS OF THE LINE (FOR READABILITY)     */
/*  - PUT OUT CNM309I AT THE END                                        */
/*  - CLOSE THE FILE WITH 'EXECIO 0 DISKR ... ( FINIS' WHETHER AN       */
/*    ERROR OCCURS OR NOT.                                              */
/************************************************************************/
SIGNAL ON HALT                              /* PROCESS TERMINATION   */
ARG DATASETNAME                             /* PARSE CLIST INPUT     */
IF DATASETNAME='' | PARMCNT() > 1 THEN      /* NO CLIST INPUT ?      */
  DO                                        /* NAME NOT SPECIFIED    */
    SAY 'INCORRECT SYNTAX USED.'            /* ISSUE ERROR MSG       */
    SAY 'CORRECT SYNTAX IS :  '             /* ISSUE HELP MSG        */
    SAY '      TYPE DATASET.NAME(MEMBER)  ' /* ISSUE HELP MSG        */
    RC=24                                   /* SET RETURN CODE       */
  END                                       /* NAME NOT SPECIFIED    */
ELSE                                        /* CORRECT NAME/SYNTAX   */
  DO                                        /* NAME WAS SPECIFIED    */
    'TRAP AND DISPLAY ONLY MESSAGES CNM272I' /* TRAP/DISPLAY CNM272I  */
    'ALLOCATE DA('DATASETNAME') SHR FREE'   /* ALLOC/CONNECT FILE    */
    'WAIT 5 SECONDS FOR MESSAGES'           /* WAIT FOR MESSAGES     */
    'MSGREAD'                               /* READ A MESSAGE IN     */
    'TRAP NO MESSAGES'                      /* DISABLE TRAP MSGS     */
    IF (MSGID()¬='CNM272I') THEN            /* IS MSG CNM272I ?      */
      DO                                    /* ¬ CNM272I MSG         */
        SAY MSGID() MSGSTR()                /* DISPLAY MESSAGE       */
      END                                   /* ¬ CNM272I MSG         */
    ELSE                                    /* MSG IS  CNM272I       */
      DO                                    /* PROCESS CNM272I MSG   */
        DDNAME = WORD(MSGSTR(),1)           /* SAVE DYNAMIC DDNAME   */
        ADDRESS MVS 'EXECIO * DISKR 'DDNAME /* READ DATA ONTO STACK  */
        DO UNTIL QUEUED() = 0               /* WHILE RC = 0          */
```

Figure 31 (Part 1 of 2). TYPEIT Example

```
        PULL RECORD                               /* PULL LINE FROM STACK  */
        SAY SUBSTR(RECORD,1,68)                   /* DISPLAY LINE TO USER  */
                                                  /* PUT NEXT LINE ON STACK*/
      END                                         /* WHILE RC = 0          */
                                                  /* PUT OUT COMPLETE MSG  */
      'MESSAGE 309I TYPEIT CLIST IS NOW FINISHED'
    END                                           /* PROCESS CNM272I MSG   */
  END                                             /* NAME WAS SPECIFIED    */

/* CLOSE THE FILE WHETHER OR NOT AN ERROR HAS OCCURRED                     */
HALT:                                             /* SIGNAL ON HALT LABEL  */
ADDRESS MVS 'EXECIO 0 DISKR 'DDNAME '( FINIS'/* CLOSE THE FILE             */
RETURN                                            /* RETURN TO CALLER/EXIT */
```

Figure 31 (Part 2 of 2). TYPEIT Example

PRINT Example

```
/**************************************************************************/
/* PRINT COMMAND LIST                                                     */
/* ------------------                                                     */
/*                                                                        */
/* FUNCTION: THIS COMMAND LIST PRINTS MEMBERS OF A DATA SET TO A          */
/*           SYSTEM PRINT FILE.                                           */
/*                                                                        */
/* INPUT PARMS: DATASETNAME = FULLY QUALIFIED DATA SET NAME               */
/*              (INCLUDING MEMBER NAME) TO DISPLAY AT THE TERMINAL.       */
/*                                                                        */
/* OUTPUT: A SYSTEM PRINT FILE.                                           */
/**************************************************************************/
ARG DATASETNAME                                   /* PARSE CLIST INPUT     */
IF DATASETNAME='' | PARMCNT() > 1 THEN            /* NO CLIST INPUT ?      */
  DO                                              /* NAME NOT SPECIFIED    */
    SAY 'INCORRECT SYNTAX USED.'                  /* ISSUE ERROR MSG       */
    SAY 'CORRECT SYNTAX IS :  '                   /* ISSUE HELP MSG        */
    SAY '      PRINT DATASET.NAME(MEMBER) '       /* ISSUE HELP MSG        */
    RC=24                                         /* SET RETURN CODE       */
  END                                             /* NAME NOT SPECIFIED    */
ELSE                                              /* CORRECT NAME/SYNTAX   */
  DO                                              /* NAME WAS SPECIFIED    */
    'TRAP DISPLAY ONLY MESSAGES *'                /* TRAP/SUPPRESS MSGS    */
    'ALLOCATE SYSOUT(A) FREE RECFM(FB) ',         /* ALLOC/CONNECT SYSTEM  */
      'LRECL(80) BLKSIZE(4000)'                   /* ... PRINTER FOR USAGE */
    'WAIT FOR MESSAGES'                           /* WAIT FOR RESULTS      */
    'MSGREAD'                                     /* READ A MESSAGE IN     */
```

Figure 32 (Part 1 of 2). PRINT Example

```
   IF (MSGID()¬='CNM272I') THEN               /* IS MSG CNM272I ?       */
     DO                                       /* ¬ CNM272I MSG          */
       SAY MSGID() MSGSTR()                   /* DISPLAY MESSAGE        */
     END                                      /* ¬ CNM272I MSG          */
   ELSE                                       /* MSG IS CNM272I         */
     DO                                       /* PROCESS 1ST CNM272I    */
       DDNAMEO = MSGVAR(1)                    /* SAVE OUTPUT DDNAME     */
       'TRAP AND DISPLAY  ONLY MESSAGES *'    /* TRAP/SUPPRESS MSGS     */
       'ALLOCATE DA('DATASETNAME') SHR FREE'/* ALLOC/CONNECT FILE     */
       'WAIT FOR MESSAGES'                    /* WAIT FOR MESSAGES      */
       'MSGREAD'                              /* READ A MESSAGE IN      */
       'TRAP NO MESSAGES'                     /* DISABLE TRAP MSGS      */
       IF (MSGID()¬='CNM272I') THEN           /* IS MSG CNM272I ?       */
         DO                                   /* ¬ CNM272I MSG          */
           SAY MSGID() MSGSTR()               /* DISPLAY MESSAGE        */
         END                                  /* ¬ CNM272I MSG          */
       ELSE                                   /* MSG IS  CNM272I        */
         DO                                   /* PROCESS 2ND CNM272I    */
           DDNAMEI = MSGVAR(1)                /* SAVE INPUT DDNAME      */
           ADDRESS MVS 'EXECIO 1 DISKR 'DDNAMEI /* READ 1ST LINE        */
           DO WHILE RC=0                      /* WHILE RC = 0           */
             ADDRESS MVS 'EXECIO 1 DISKW 'DDNAMEO /* WRITE LINE OUT     */
             ADDRESS MVS 'EXECIO 1 DISKR 'DDNAMEI /* READ NEXT LINE     */
           END                                /* WHILE RC = 0           */
                                              /* PUT OUT COMPLETE MSG   */
           'MESSAGE 309I TYPE CLIST IS NOW FINISHED'
         END                                  /* PROCESS 2ND CNM272I    */
     END                                      /* PROCESS 1ST CNM272I    */
 END                                          /* NAME WAS SPECIFIED     */

RETURN                                        /* RETURN TO CALLER/EXIT */
```

Figure 32 (Part 2 of 2). PRINT Example

CHKOPNUM Example

```
/***********************************************************************/
/*                                                                     */
/*  THE FOLLOWING REXX COMMAND LIST IS A FAIRLY SIMPLE EXAMPLE         */
/*  OF HOW SOME OF THE BASIC REXX FUNCTIONS AND NETVIEW-SPECIFIC       */
/*  FUNCTIONS CAN BE USED IN A COMMAND LIST.  IT ILLUSTRATES THE USAGE*/
/*  OF SUCH THINGS AS THE REXX 'PARSE' INSTRUCTION, AND THE NETVIEW   */
/*  SUPPLIED 'MSGTRAP', 'WAIT', 'MSGREAD', AND 'GLOBALV' COMMMANDS.   */
/*                                                                     */
/***********************************************************************/
/*                                                                     */
/*  COMMAND LIST NAME:  CHKOPNUM                                       */
/*                                                                     */
/*     THIS COMMAND LIST CAN BE USED PERIODICALLY TO CHECK THE         */
/*     NUMBER OF OPERATORS CURRENTLY LOGGED ON, AND WILL KEEP THE      */
/*     INFORMATION IN COMMON GLOBAL VARIABLES.  THE INFORMATION        */
/*     COLLECTED CAN LATER BE RETRIEVED BY USING THE 'DISPLAY'         */
/*     OPTION.                                                         */
/*                                                                     */
/*  INPUT:                                                             */
/*     ''          - WILL CHECK THE NUMBER OF OPERATORS LOGGED ON      */
/*                   AND UPDATE APPROPRIATE COMMON GLOBAL VARIABLES    */
/*     'DISPLAY'   - WILL ANALYZE THE VALUE IN THE COMMON GLOBAL       */
/*                   VARIABLES AND DISPLAY THE RESULTS                 */
/*     ANY OTHER                                                       */
/*     INPUT       - WILL DEFAULT TO ''                                */
/*                                                                     */
/*  USAGE EXAMPLE:                                                     */
/*  1. EXECUTE THE FOLLOWING TO CAUSE THE NUMBER OF OPERATORS          */
/*     TO BE CHECKED EVERY HOUR (COULD BE ANY TIME PERIOD):            */
/*     ->  'EVERY 01:00,PPT,CHKOPNUM'                                  */
/*  2. AT ANY TIME, EXECUTE THE FOLLOWING COMMAND TO DISPLAY THE       */
/*     RESULTS OF THE PREVIOUS EXECUTIONS:                             */
/*     ->  'CHKOPNUM DISPLAY                                           */
/*     RESULTS WILL BE DISPLAYED ON YOUR TERMINAL                      */
/*                                                                     */
/*  CHANGE CODE  DATE     DESCRIPTION                                  */
/*  -----------  -------- ---------------------------------------- */
/*                                                                     */
/***********************************************************************/
PARSE ARG OPTION                               /* GET INPUT, IF ANY    */

/* GET VALUES FOR COMMON GLOBAL VARIABLES USED BY THIS EXEC            */
'GLOBALV GETC CHKOPTIMES, CHKOPNUM, CHKOPMAX'

IF OPTION = 'DISPLAY' THEN DO;                /* WAS 'DISPLAY' REQSTD? */

  IF CHKOPTIMES = '' THEN                      /* ANYTHING TO DISPLAY?  */
    SAY 'NUMBER OF OPERATORS HAS NEVER BEEN CHECKED '/* NO, GIVE ERROR*/
```

Figure 33 (Part 1 of 2). CHKOPNUM Example

```
  ELSE DO;                                     /* YES, DISPLAY RESULTS  */
    SAY 'NUMBER OF OPERATORS HAS BEEN CHECKED 'CHKOPTIMES' TIMES'
    SAY 'AVERAGE NUMBER OF OPERATORS LOGGED ON IS: 'CHKOPNUM/CHKOPTIMES
    SAY 'MAXIMUM NUMBER OF OPERATORS LOGGED ON IS: 'CHKOPMAX
  END;                                         /* END DISPLAY RESULTS   */
  EXIT 0;                                      /* EXIT FROM COMMAND LIST*/
END;                                           /* END DISPLAY OPTION    */

CUROPNUM = 0
'TRAP AND SUPPRESS MESSAGES OPERATOR:,END'     /* TRAP LIST RESPONSE    */
'LIST STATUS=OPS'                              /* ISSUE LIST COMMAND    */
DO UNTIL MSGID()='END'                         /* LOOP TILL END OF MSGS */
  'WAIT FOR MESSAGES'                          /* WAIT FOR RESPONSE     */
  'MSGREAD'                                    /* YES, READ IN MESSAGE  */
  IF MSGID() = 'OPERATOR:' THEN                /* IS IT AN OPERATOR LINE*/
    CUROPNUM = CUROPNUM +1                     /* YES,INCREMENT # OPS   */
  ELSE NOP                                     /* NO, MUST BE 'END' MSG */
END
IF CHKOPTIMES = '' THEN CHKOPTIMES = 1         /* INCREMENT # TIMES CHKD*/
ELSE CHKOPTIMES = CHKOPTIMES + 1
IF CHKOPNUM = '' THEN CHKOPNUM = CUROPNUM      /* INCREMENT #OPS ON SYS */
ELSE CHKOPNUM = CHKOPNUM + CUROPNUM
IF CHKOPMAX = '' THEN CHKOPMAX = CUROPNUM      /* INCREMENT MAX IF NEED */
ELSE IF CHKOPMAX < CUROPNUM THEN CHKOPMAX = CUROPNUM

/* PUT NEW VALUE BACK INTO THE COMMON GLOBAL VARIABLE DICTIONARY        */
'GLOBALV PUTC CHKOPTIMES, CHKOPNUM, CHKOPMAX'

EXIT 0;                                        /* END OF COMMAND LIST   */
```

Figure 33 (Part 2 of 2). CHKOPNUM Example

CHKRSTAT Example

```
/***********************************************************************/
/*                                                                     */
/*  THE FOLLOWING REXX COMMAND LIST IS MORE COMPLEX THAN CHKOPNUM.    */
/*  IT ILLUSTRATES USAGE OF SUCH THINGS AS THE REXX 'INTERPRET'        */
/*  INSTRUCTION, AND THE NETVIEW 'WAIT' (FOR MESSAGES AND TIME),       */
/*  AND THE 'GETMLINE' /*  COMMAND (FOR MULTI-LINE MESSAGES)           */
/*                                                                     */
/***********************************************************************/
/*                                                                     */
/*  COMMAND LIST NAME:  CHKRSTAT                                       */
/*                                                                     */
/*     THIS COMMAND LIST CHECKS WHETHER A SPECIFIED RESOURCE           */
/*     IS ACTIVE, AND INCREMENTS A COMMON GLOBAL VARIABLE THAT         */
/*     REFLECTS THE NUMBER OF TIMES IT WAS IN THAT STATE. THIS         */
/*     COMMAND LIST SHOULD BE SCHEDULED TO RUN UNDER AN AUTOTASK       */
/*     AT REGULAR INTERVALS.                                           */
/*                                                                     */
/*  INPUT PARAMETERS:                                                  */
/*        RESNAME - NAME OF RESOURCE TO CHECK STATUS OF                */
/*                                                                     */
/*  CHANGE CODE  DATE      DESCRIPTION                                 */
/*  -----------  --------  ------------------------------------------ */
/*                                                                     */
/***********************************************************************/
PARSE UPPER ARG RESNAME                          /* GET INPUT, IF ANY  */

/* IF NO RESROUCE NAME GIVEN, DISPLAY ERROR MESSAGE AND EXIT           */
IF RESNAME = '' THEN DO;
  SAY 'RESOURCE NAME MUST BE PROVIDED'
  EXIT 99
  END

/* SET UP TRAP FOR POSSIBLE RESPONSES TO 'D NET,ID=' COMMAND, ISSUE   */
/* COMMAND, AND WAIT FOR MESSAGE TO ARRIVE                             */
'TRAP AND SUPPRESS MESSAGES IST097I IST075I IST453I'
'D NET,ID='RESNAME
'WAIT 60 SECONDS FOR MESSAGES'

/* IF MESSAGE DID NOT ARRIVE, THEN GIVE ERROR MESSAGE AND EXIT         */
IF EVENT() ¬= 'M' THEN DO
  SAY ' NO RESPONSE FROM VTAM - RESOURCE COUNT NOT UPDATED '
  EXIT 99
  END

/* READ MESSAGE. IF IT IS IST097I, ISSUE WAIT AGAIN, AND THE NEXT     */
/* MESSAGE READ SHOULD BE IST075I, WHICH HAS THE STATUS INFO           */
'MSGREAD'
IF MSGID() = 'IST097I' THEN DO;
  'WAIT CONTINUE'
  'MSGREAD'
```

Figure 34 (Part 1 of 2). CHKRSTAT Example

```
  /* IF THE MESSAGE IS NOT IST075I, DO NOTHING, AND THE STATUS WILL   */
  /* DEFAULT TO INACTIVE.  IF IT IS IST075I, GET THE 2ND LINE OF THE  */
  /* MULTI-LINE MESSAGE AND GET THE CURRENT STATE FROM THAT LINE      */
  IF MSGID() = 'IST075I' THEN DO
   'GETMLINE  STATLINE ' 2
   /* IF STRING CONTAINS IST486I THEN PARSE OUT RESOURCE STATUS       */
    IF INDEX(STATLINE,'IST486I') >0 THEN
       PARSE VALUE STATLINE WITH MSGTXT1 'CURRENT STATE =' RESSTATE .
  END
END

/* IF THE CURRENT STATE IS ACTIVE OR ACTIVE W/SESSION, THEN GET       */
/* INCREMENT AND UPDATE THE COMMON GLOBAL VARIABLE WITH THE NAME      */
/* 'RESOURCE NAME' CONCATENATED WITH '@A'.  NOTE THAT SINCE THE       */
/* GLOBALV COMMAND REQUIRES THE VARIABLE NAME, A VARIABLE HAS         */
/* TO BE SET TO THE VARIABLE NAME, SINCE IT IS DYNAMICALLY            */
/* CONSTRUCTED.  THE REXX INTERPRET INSTRUCTION MUST ALSO BE USED     */
/* TO PERFORM OPERATIONS ON THE DYNAMICALLY CONSTRUCTED VARIABLE      */
IF RESSTATE = 'ACTIV' | RESSTATE = 'ACT/S' THEN DO
  VARNAME = RESNAME||'@A'
  'GLOBALV GETC 'VARNAME
  INTERPRET 'ACT# ='VARNAME'
  IF DATATYPE(ACT#) ¬= 'NUM' THEN ACT# = 1        /* IF NON-NUMERIC   */
  ELSE ACT# = ACT# + 1
  INTERPRET VARNAME'=ACT#'
  'GLOBALV PUTC 'VARNAME
END

/* IF THE CURRENT STATE IS NOT ACTIVE OR ACTIVE W/SESSION, THEN GET   */
/* INCREMENT AND UPDATE THE COMMON GLOBAL VARIABLE WITH THE NAME      */
/* 'RESOURCE NAME' CONCATENATED WITH '@NA'.  NOTE THAT SINCE THE      */
/* GLOBALV COMMAND REQUIRES THE VARIABLE NAME, A VARIABLE HAS         */
/* TO BE SET TO THE VARIABLE NAME, SINCE IT IS DYNAMICALLY            */
/* CONSTRUCTED.  THE REXX INTERPRET INSTRUCTION MUST ALSO BE USED     */
/* TO PEFFORM OPERATIONS ON THE DYNAMICALLY CONSTRUCTED VARIABLE      */
ELSE DO
  VARNAME = RESNAME||'@NA'
  'GLOBALV GETC 'VARNAME
  INTERPRET 'NACT# ='VARNAME
  IF DATATYPE(NACT#) ¬= 'NUM' THEN NACT# = 1        /* IF NON-NUMERIC */
   ELSE NACT# = NACT# + 1
  INTERPRET VARNAME'=NACT#'
  'GLOBALV PUTC 'VARNAME
END
exit 0;
```

Figure 34 (Part 2 of 2). CHKRSTAT Example

DSPRSTAT Example

```
/*********************************************************************/
/*                                                                   */
/*  THE FOLLOWING REXX COMMAND LIST GOES ALONG WITH THE PREVIOUS     */
/*  EXAMPLE (CHKRSTAT), AND SHOWS MANY OF THE SAME TYPE OF FUNCTIONS */
/*  AS THE PREVIOUS EXAMPLE.                                         */
/*                                                                   */
/*  THIS COMMAND LIST COULD BE USED BY ANY OST OPERATOR TO DISPLAY   */
/*  THE RESULTS OF SEVERAL EXECUTIONS OF THE CHKRSTAT COMMAND LIST   */
/*  FOR A SPECIFIC RESOURCE.  IT COULD BE USED AS AN AID IN          */
/*  DETERMINING HOW OFTEN A RESOURCE IS ACTIVE, BASED ON THE INTERVALS*/
/*  IN WHICH IT WAS CHECKED BY THE CHKRSTAT COMMAND LIST             */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*  COMMAND LIST NAME:  DSPRSTAT                                     */
/*                                                                   */
/*     THIS COMMAND LIST CAN BE USED TO DISPLAY HOW OFTEN A RESOURCE */
/*     WAS ACTIVE VS. NOT ACTIVE, AS RECORDED BY THE CHKRSTAT COMMAND */
/*     LIST                                                          */
/*                                                                   */
/*  INPUT PARAMETERS: NONE                                           */
/*                                                                   */
/*  CHANGE CODE  DATE      DESCRIPTION                               */
/*  -----------  --------  ----------------------------------------- */
/*                                                                   */
/*********************************************************************/
PARSE UPPER ARG RESNAME                           /* GET INPUT, IF ANY  */

/* IF NO RESOURCE NAME GIVEN, DISPLAY ERROR MESSAGE AND EXIT         */
IF RESNAME = '' THEN DO;
  SAY 'RESOURCE NAME MUST BE PROVIDED'
  EXIT 99
  END

VARNAMEA = RESNAME||'@A'                        /* SET THE VAR NAME ACT  */
VARNAMENA = RESNAME||'@NA'                      /* SET THE VAR NAME NACT */
'GLOBALV GETC 'VARNAMEA'                        /* GET THE ACTIVE INFO   */
'GLOBALV GETC 'VARNAMENA                        /* GET THE INACTIVE INFO */
INTERPRET 'RACT='VARNAMEA                       /* PUT ACTIVE # IN VAR   */
INTERPRET 'RINACT = 'VARNAMENA                  /* PUT INACTIVE # IN VAR */

/* DISPLAY THE STATISTICS FOF THE RESOURCE SPECIFIED                 */
IF RACT = '' & RINACT = '' THEN
  SAY 'NO STATISTICS HAVE BEEN COLLECTED FOR RESOURCE: 'RESNAME

/* DISPLAY THE STATISTICS FOR THE RESOURCE SPECIFIED                 */
ELSE DO
  IF DATATYPE(RACT) ¬= 'NUM' THEN RACT = 0          /* IF NOT NUMERIC */
  IF DATATYPE(RINACT) ¬= 'NUM' THEN RINACT = 0       /* IF NOT NUMERIC*/
```

Figure 35 (Part 1 of 2). DSPRSTAT Example

```
  SAY 'RESOURCE 'RESNAME' STATISTICS:'
  SAY '    NUMBER OF TIMES RESOURCE WAS ACTIVE  : 'RACT
  SAY '    NUMBER OF TIMES RESOURCE WAS INACTIVE: 'RINACT
  PERCENTACT = RACT/(RACT+RINACT)*100||'%'      /* DETERMINE PERCENT */
  SAY '    PERCENTAGE OF TIMES RESOURCE WAS ACTIVE: 'PERCENTACT
END
EXIT 0;
```

Figure 35 (Part 2 of 2). DSPRSTAT Example

Part Three. Writing Command Lists in the NetView Command List Language

Chapter 6. Simple NetView Command List Language Command Lists

In this chapter, you will learn the basics of writing command lists for the NetView program using the NetView command list language. This chapter also describes how variables, assignment statements, and built-in functions fit together and how to combine them in command lists.

In simple command lists, each statement is interpreted in the order it appears. One or more segments of your application processing are performed in sequence as coded. Simple command lists do not give you the flexibility to skip around some segments, to perform alternate segments, or to repeat processes. These features are included in the NetView command list language, and you will learn about them in Chapter 7, "NetView Command List Language Branching" on page 107.

A comprehensive sample command list, with examples of each type of statement in this chapter, appears in "Sample Command List—Chapter Review" on page 105. Also, see "Examples Comparing REXX and NetView Command List Language" on page 193 for samples of command lists written in the NetView command list language and the equivalent REXX command lists.

What the NetView Command List Language Includes

The NetView Command list language is made up of statements, each having a separate function. NetView uses the following six types of command list statements:

- Command
- Comment
- Control
- Assignment
- Label
- Null.

You can use the following features of the command list language within command list statements:

- Parameter variables
- Control variables
- User variables
- Global variables
- Built-in functions.

All of these, except global variables, are discussed in detail in later sections of this chapter. Global variables and descriptions of passing parameter values are described in Chapter 8, "NetView Command List Language Global Variables."

The NetView command list language also gives you the capability to write application code to perform repetitive or alternate processing (if-then or loop structures). These features are implemented with the following control statements:

- &IF
- &GOTO
- &EXIT
- &WAIT.

These control statements are discussed in Chapter 7, "NetView Command List Language Branching."

Note: Command lists can interrupt the processing of other command lists. This is done using the CMDMDL statement in the DSICMD. For more information, see *Netview Administration Reference.*

Coding Conventions for NetView Command List Language Statements

Like any other language, the NetView command list language requires that you follow syntax rules. The following coding conventions for NetView are divided into sections describing the conventions for general coding, continuation characters, suppression characters, and double-byte character sets.

Note: The syntax diagrams used in this book are formatted according to the coding conventions described under "Coding Conventions Used in This Book" on page xiv.

General Coding Conventions

Use the following coding conventions when writing command lists in the NetView command list language:

- Optionally, code a CLIST statement as the first line of your command list. Code CLIST statements as follows:
  - Optionally, code a label. The label must begin in column one.
  - Code the characters CLIST beginning in column two or later. CLIST must be preceded by at least one blank.

    Note: On VM systems, if the command list is not defined on a CMDMDL definition statement, you must start the command list with a CLIST statement. It is not optional.
- Leave column 72 blank for all statements.
- Do not use columns 73-80. They are reserved for optional sequence numbers.
- Code at least one blank after a label (if there is one) or before a keyword.
- Code at least one blank between a control statement and the first operand.
- Separate operands with one or more blanks, or a single comma with no blanks.
- Code any number of leading or trailing blanks on your statements.
- Use lowercase letters only as comments or part of a message sent to the operator. In all other cases, use uppercase for alphabetical characters A-Z.

- Code statements so that their maximum length is 240 characters after variable substitution. To familiarize yourself with how variable substitution works, see "Variable Substitution Order" on page 76.
- Code comment lines with an asterisk (*) in the first column followed by the comment. Comment records cannot appear on the first line of a command list.
- Code the command list so that it ends by processing the last command list statement, or by reaching an &EXIT statement. An operator entering RESET also ends the command list.

Conventions for Continuation Statements

Use a plus sign (+) or a hyphen (-) as a continuation character to continue a statement that is too long to fit on one line. Code the continuation character as the last non-blank character before column 72 on the line to be continued.

Note: Do not code a comment between the beginning and end of a continued statement.

- The plus sign causes the text of the continuation line to begin where the plus sign was placed without any of the blanks leading up to the first non-blank character on the continued line.

  The plus sign causes these lines:

  ```
  &WRITE THIS STATEMENT IS CODED +
                             AS +
                       THREE LINES
  ```

  to become this single statement:

  ```
  THIS STATEMENT IS CODED AS THREE LINES
  ```

- The hyphen causes NetView to keep all the blanks at the end of the line with the hyphen (up to but not including column 72) and then fill the line to its end with characters from the beginning of the continuation line. The hyphen itself is replaced by a blank. When filling a line with characters from the beginning of the continuation line, NetView presentation services does not split a word across lines of an output screen. The last character used for filling in from the continuation line must be a blank or the last character on the line.

  For example, if you coded the following &WRITE statement to be displayed on an 80-character-wide terminal:

  ```
  &WRITE STATEMENT CONTINUED WITH THE HYPHEN TO KEEP -
    BLANKS
  ```

  all the blanks from the P in KEEP to the B in BLANKS would be kept. The first line would write 64 characters to the output screen (41 characters of text plus 23 blanks from the end of the text to column 72). The output screen has 68 columns to be used for display (80 minus the 12-character prefix), so the hyphen would cause the first four characters of the second line to be placed at the end of the first line. In the example, this would be two blanks and the letters BL. However, since NetView presentation services will not split a word across lines of the output screen, the example would be displayed as:

  ```
  STATEMENT CONTINUED WITH THE HYPHEN TO KEEP
  BLANKS
  ```

Conventions for Double-Byte Character Set Text

A double-byte character set (DBCS) is a character set in which each symbol is represented by a two-byte code rather than one-byte codes. For example, Kanji is a language used in Japan that is too rich in symbols to display all the characters using one-byte codes. Therefore, a double-byte character set is used. Double-byte character set support is available for MVS/XA only.

In the following, DBCS refers to double-byte character set characters. The term "Latin characters" refers to the English character set, a-z and A-Z.

- Use a Latin character set to code NetView commands and command lists used as commands.
- DBCS data input is not supported.
- Enclose all DBCS strings within shift-out (X'0E') and shift-in (X'0F') codes. Be sure there are an even number of characters in each DBCS string (if you are using an editor and terminal that supports double-byte characters this is done automatically).
- Label names and variable values can be coded in DBCS characters. Restrict them to a length of 11 bytes.
- When DBCS labels and variables are displayed on a DBCS terminal, the shift-out and shift-in codes appear as blanks.
- DBCS text can be split across multiple lines, using an EBCDIC plus sign (+) or hyphen (-) as a continuation character.
- When writing DBCS text in a &BEGWRITE, the SUB option is required.
- Comments can contain DBCS strings enclosed by shift-out (X'0E') and shift-in (X'0F') codes.
- &WRITE, &CONCAT, and &SUBSTR are enabled for double-byte character sets.

Conventions for Suppression Characters

You can define a suppression character with the SUPPCHAR operand of the NCCFID definition statement and use it to prevent a command list command or statement from appearing on the operator's screen. See *NetView Administration Reference* for more information about the NCCFID definition statement.

The following rules apply when coding suppression characters:

- Code the suppression character in column 1 of the statement.
- Only the first line of a continued statement can be suppressed when the command list is listed using the LIST CLIST= command.
- When you browse a file, you can see every line, even suppressed lines.

In Figure 36 on page 75, the question mark (?) has been defined as a suppression character. The first and last lines of the command list in the example will be suppressed.

```
?* COMMAND LIST UPDATED 2/5/87 BY OPERATOR IRENE
START DOMAIN=&1
&WRITE ENTER GO WHEN MESSAGE DSI809I ARRIVES FROM &1
&PAUSE
?ROUTE &1,OPER1,123456
```

Figure 36. Suppression Characters

When issuing a command that returns its status in the return code, you can suppress synchronous output from the command by coding the suppression character twice. For example, if you coded:

```
??SET PF24 IMMED RETRIEVE
```

in a command list, no synchronous output from the command list is displayed to the operator.

Use the double suppression character to enhance performance on commands that produce line mode messages synchronously and when sufficient status is provided by the return code. Using the double suppression character does not affect output that is scheduled by a command (for example, D NET,APPLS) nor does it reliably reduce output from a long running command (for example, NLDM).

Labels

Labels identify command list statements for control of flow, or internal documentation, or to indicate the target statement for a transfer of control. You will learn how to use transfer of control in Chapter 7, "NetView Command List Language Branching" on page 107.

Labels can be coded on any command list statement, except a comment statement. This means you can code labels on commands, control statements, assignment statements, and null statements. If NetView cannot find the label, processing stops, and NetView issues an error message.

Code the label as the first non-blank word in the command list line. A label consists of an EBCDIC hyphen (-) followed by one to eleven characters (A-Z, 0-9, #, @, $). Start the command list statement after the label, leaving at least one blank between the label and a keyword. Labels can be used on null lines, so you do not have to code a command list statement after a label.

You can also code other labels. All labels must be unique within a command list. If you have two identical labels in one command list, NetView ends the command list. You can also code labels as internal comments to show where different parts of your command list start. For example, you can use labels to highlight certain processing routines.

Figure 37 shows examples of labeled command list statements.

```
-MYLABEL VARY NET,INACT,ID=LU1234
-$PROC2  &LEN = &LENGTH &1
-SETUP   &USER = 55
-ALLALONE
```

Figure 37. Labels in Command List Statements

Note: Labels are used with &BEGWRITE to show where a message stops. Variables are not allowed in labels, but you can code a variable as the label name with the &BEGWRITE, &GOTO, or &WAIT statements. These statements for transfer of control are described in Chapter 7, "NetView Command List Language Branching" on page 107.

Variables

Variables let you accept from an operator, or define for yourself, different values for the statements within a command list. With variables, you can write a command list that operates correctly in many different situations. There are four types of variables:

- Parameter
- Control
- User
- Global.

In this section you will learn how to use parameter, control, and user variables. This section also describes how to use the NetView PARSEL2R command to parse variables in a command list. See Chapter 8, "NetView Command List Language Global Variables" on page 123 for a description of global variables.

Variable Substitution Order

Variable substitution is performed when NetView scans each statement from right to left and substitutes values for each variable. This is done as follows:

1. Each element is scanned from right to left for an ampersand (&).

   Note: The value of X'50' (ampersand in the English character set) is ignored within double-byte character sets.

   - If found, the ampersand and the rest of the element to the right are substituted with the value of that variable.

   - If no value exists, the variable becomes null.

   - If the first character to the right of the ampersand is a number, the variable is assumed to be a parameter variable. NetView then scans to the right and takes any following numbers as part of the parameter variable. When NetView comes to a blank or a letter, the search stops. If a special character (non-alphanumeric) is found, it delimits the variable name.

     For example, &21A is taken as &21 and is replaced by the value of &21. Therefore, &21A becomes *value*A. For another example, if an element contains &A=&XYZ, NetView first substitutes the value of &XYZ, then NetView replaces &A with the value it has just substituted for &XYZ.

2. The scan resumes at the next character to the left, and the search for an ampersand continues. If found, the ampersand and the entire syntactical element to the right, including the previous substitution, are taken as the name of a variable and are replaced by its value. Note that the value substituted is not scanned for an ampersand.

   If the element is the target of an assignment statement, the scan stops on the second character to preserve the variable name that will be assigned a value. For example, the statements in Figure 38 set the value of user variable &A1 to 2.

```
&B = 1
&A&B = 2
```

Figure 38. Variable Substitution Example

Variable substitution is not done on the following features of the language:

- The &PAUSE statement

  The variables are assigned values when you enter a GO command. For more information, see "&PAUSE Control Statement" on page 97.

- The &THEN clause on an &IF statement

  If the &IF clause is true, the &THEN clause is made into a statement and processed as if it had been coded separately. For more information, see "&IF Control Statement" on page 107.

- Any statements in an &BEGWRITE NOSUB series of messages

  For more information, see "&BEGWRITE Control Statement" on page 95.

- Control keywords

  For more information, see "&CONTROL Control Statement" on page 92.

- Built-in functions

  For more information, see "Built-In Functions" on page 99.

Parameter Variables

A parameter variable is a positional variable that is defined at the time a command list is run. You specify parameter variables by entering them as operands following the name of the command list that you are running. Parameter variables have the following characteristics:

- Identified within the command list by a numbered position, for example, &1
- Entered following the command list name at run time
- Delimited by commas, apostrophes, or blanks.

When you code your command list with parameter variables, use the following guidelines:

- You can use up to 31 parameter variables in a single command list.
- You can use the same parameter variables more than once in a command list.
- The value of a parameter variable can be 238 characters long.
- Parameter variables can contain either numerical or character values.
- A parameter variable can have a numerical value between -2147483647 and 2147483647.

Note: When NetView receives a message coded in an &WAIT statement, it sets the four control variables (&MSGORIGIN, &MSGID, &MSGCNT, and &MSGSTR) and then changes the values of the parameter variables (&1 - &31) to reflect the information in the received message. See "Control Variables" on page 81 for information on these variables. LINKPD sets the same control and parameter variables. See "LINKPD Results" on page 167 for more information on the LINKPD command.

Passing Variable Information to a Command List

When activating a command list that uses parameter variables, the operator enters the command list name followed by a value for each parameter variable in the command list. Figure 39 shows the format for an operator passing up to 31 parameter variables to a command list.

```
cmdlistname  _____,_____,_____,. . .,_____
              &1    &2    &3          &31
```

Figure 39. Format for Passing Parameter Variables to a Command List

The first value after the command list name replaces &1 in the command list, the next value replaces &2, and so on. For example, the second parameter variable in a command list would be coded &2 at the place where you want the value of that parameter.

Assume that you wrote a command list named RESC to start resource LU100 as shown in Figure 40.

```
RESC CLIST
&CONTROL ERR
VARY NET,ACT,ID=LU100
```

Figure 40. RESC Command List to Start LU100

If you want the command list to use parameter variables, you can change it to activate or inactivate any resource. Figure 41 shows how the command list looks with parameter variables.

```
RESC CLIST
&CONTROL ERR
VARY NET,&1,ID=&2
```

Figure 41. RESC Command List with Parameter Values

The operator can then start resource LU100 by entering RESC ACT,LU100.

When the command list runs, &1 and &2 are replaced with the positional parameters: &1 with ACT, and &2 with LU100. The command list takes the values for &1 and &2 from the entered operands in the order in which the operands were entered after the command list name.

Note: The operator who uses the command list must be told how many parameter variables to supply and what values to provide. For more information, see "&BEGWRITE Control Statement" on page 95.

If a command list is activated by a message, each word of the message becomes a separate parameter variable. This is explained in more detail in Chapter 9, "Message Automation" on page 135.

How Parameter Variables Are Used in the Command List

There is no set order for placing the parameter variables in the command list. Figure 42 shows that you can use &2 before &1.

```
V NET,&2,ID=&1
```

Figure 42. Nonsequential Use of Parameter Variables in a Command List

&1 is given the first value the operator enters, and &2 is given the second value.

If there are two or more parameter variables in one command list statement, the rightmost variable is changed first. NetView continues to scan right to left and replaces the next variable. You can use this to change the meaning of some of your parameter variables. If you need to test how many parameters an operator entered or what parameter values were entered, use the control variables &PARMCNT and &PARMSTR. They are described in "Control Variables" on page 81.

Passing Parameter Variables to a Nested Command List

You can code parameter variables on the command list statement that activates the nested command list. These parameter variables follow the same basic rules as other parameter variables. One added function is that you can pass either actual values or other variables as parameter variables. If you pass other variables, make sure these variables are known to the next activated command list. Here are some examples of passing parameters.

Command List CALLER contains a line of code such as:

```
CALLEE LINES,TERMS,CDRMS
```

Figure 43. Example of Passing Parameters

Here is how command list CALLEE picks up these variables:

Variable	Value
&1	LINES
&2	TERMS
&3	CDRMS

Command list MAJOR is activated by entering MAJOR ALPHA,BETA and contains the following statements:

```
&A = 55
MINOR &A,&1,&2
```

Figure 44. Statements in MAJOR Example Command List

Here is how command list MINOR picks up these variables:

Variable	Value
&1	55
&2	ALPHA
&3	BETA

Command list MINOR takes the value of &A (55) as its first parameter, the value of MAJOR's first parameter (ALPHA) as its second parameter, and the value of MAJOR's second parameter (BETA) as its third parameter.

Using Text Strings or Special Characters in Parameters

If you need to use a blank, apostrophe, or comma as part of a value, you must make the value a special character string by using single quotes. If you want a text string to be taken as the value for one parameter, it must also be made a special character string.

A special character string is any text that meets one of the following requirements:

- Text that is preceded by a delimiter and a single quote, and is followed by either a single quote and a delimiter or a single quote that is the rightmost non-blank.
- Text that is preceded by a single quote that is the leftmost non-blank, and is followed by a single quote and a delimiter
- Text that is preceded by a single quote that is the leftmost non-blank, and is followed by a single quote that is the rightmost non-blank.

Suppose you activate a command list named RESC by entering the following:

```
RESC ACT,'LU200,LOGMODE=S3270'
```

The parameter variables in the RESC command list would contain the following values:

```
&1 = ACT
&2 = LU200,LOGMODE=S3270
```

Now suppose you activated the RESC command list by entering:

```
RESC ACT,LU200,LOGMODE=S3270
```

The parameter variables in this case contain the values:

```
&1 = ACT
&2 = LU200
&3 = LOGMODE=S3270
```

Null Parameter Values

Use a comma immediately following another comma (,,) to give a parameter variable a null value when it is followed by other non-null parameters. After the last non-null parameter, all remaining parameter variables up to &31 are automatically given null values. Null parameters are useful when a value is not required. For example, assume you wrote a command list called CONN that contained the following statements:

```
BGNSESS
OPCTL,APPLID=&1,SRCLU=&2,SESSID=&3,LOGMODE=&4
```

Figure 45. Statements in CONN Example Command List

If you do not want to specify all the values, you can enter the following:

```
CONN HCF,,SESSHCF
```

Figure 46. Statement to Activate CONN Example Command List

In this example, HCF is &1, &2 is null, SESSHCF is &3, and &4 is null. The extra comma between HCF and SESSHCF tells the command list that &2 is null and that SESSHCF should be &3. If you only used one comma, the command list takes SESSHCF for &2 and incorrectly uses SESSHCF as the SRCLU.

If you allow null parameter variables, you must test for them in your command list and provide default values. Otherwise, as in the example, the BGNSESS command issued with SRCLU=*null* would get a syntax error.

Control Variables

Control variables are set by NetView based on system information. To use a control variable, place the variable name in the command list at the location where you want the information to be accessed. When the command list runs, NetView gives the correct values to each control variable. Use the LISTVAR command to view the values of your variables (except for &PARMCNT, &PARMSTR, and &RETCODE). For more information on LISTVAR, see *NetView Operation*.

The control variables let you obtain information about the operating environment, test conditions in the command list, and take actions based on the results.

For more information on control variables used with the SPCS commands LINKDATA and LINKTEST, see “LINKDATA and LINKTEST Results” on page 166.

Time and Date

&DATE
becomes the current date in the format *mm/dd/yy*, where *mm* is the month, *dd* is the day, and *yy* is the year.

&TIME
becomes the CPU time in the format *hh:mm*, where *hh* is the hour and *mm* is the minutes. The time is based on a 24-hour clock, so 3 o'clock p.m. is shown as 15:00 (12:00 noon + 3:00 p.m. = 15:00).

Session Information

&APPLID
becomes the application program identifier for the task under which the command list is running. &APPLID is always the NetView domain ID appended with a 3-character alphanumeric value assigned by NetView. For example, if your domain ID is PARIS, &APPLID might be PARIS001.

&NCCFCNT
becomes the number of NetView domains with which the operator can establish a cross-domain session.

&OPSYSTEM
becomes a character string that indicates the operating system for which NetView was compiled. &OPSYSTEM allows the same command list to run under different operating systems by allowing the command list to test for the type of operating system and process accordingly. &OPSYSTEM can contain the following character values:

MVS/XA
VM.

&TASK

becomes the 3-character string PPT (primary POI task), OST (operator station task), or NNT (NetView-NetView task), depending on the task under which the command list is running.

&TASK allows a command list to run under any of these tasks, because the command list can test for the task type and process accordingly. This is required because there are restrictions on command lists running under some tasks. See “Primary POI Task Restrictions” on page 17 for an example of these restrictions.

&VTAM
becomes a character string that indicates the level of the access method used. The variable is returned in one of two formats depending on whether the level of the access method includes a modification number.

If the level of the access method does not include a modification number, the format of the variable is *VTvr*, where:

```
VT - indicates the access method is VTAM
v  - indicates the version number of the access method
r  - indicates the release number of the access method.
```

If the level of the access method includes a modification number, the format of the variable is *Vvrm*, where:

```
V - indicates the access method is VTAM
v - indicates the version number of the access method
r - indicates the release number of the access method
m - indicates the modification number of the access method.
```

For example, for VTAM Version 3 Release 2, the variable returns a value of VT32. For VTAM Version 3 Release 1 Modification 1, the variable returns a value of V311.

Note: The value of &VTAM is null if VTAM is not active.

Terminal Information

&HCOPY
becomes the name of the hard-copy log printer started by the operator. If there is no hard-copy printer for this operator, &HCOPY is null.

&LU
becomes the logical unit name for the operator terminal.

Operator Information

&OPID
becomes this operator's ID.

Command List Information

&COMPNAME
the 16-byte name of the component running when the command list was initiated. For example, if command list HELP is initiated, &COMPNAME defines the active component so that the correct HELP command list is initiated. &COMPNAME can contain the following character values:

```
DSINCCF DSINCCF
DSINPDA
DSINLDM
DSIVIEW 1
DSIVIEW 3
DSIVIEW 4
DSIVIEW 5
DSIVIEW 6
DSIVIEW 7
DSIVIEW 8
DSIVIEW 9
DSISTATMONRESUME
DSILBROWSERESUME
DSIVIEW APPL1
```

&PARMCNT
becomes the number of parameter variables entered when the command list was initiated. For example, if command list RESC is initiated by entering RESC ACT,LU200, then &PARMCNT becomes 2. If there were no parameter variables, &PARMCNT would be 0.

&PARMSTR
becomes the string of parameter values used when the command list was initiated. &PARMSTR does not include the command list name. For example, if command list RESC is initiated by entering RESC ACT,LU200, then &PARMSTR becomes ACT,LU200. If there are no parameter variables, &PARMSTR is null. &PARMSTR must not exceed 255 characters.

&RETCODE
is the return code set by either the most recent command processor or the most recently activated or nested command list.

&RETCODE is initialized to zero. &RETCODE is set by a command processor or nested command list. When you write a command list that is called by another command list, you can set a return code on the &EXIT statement in the nested command list. You can use &RETCODE to test this return code in the calling command list. See "&EXIT Control Statement" on page 109.

On the &EXIT statement, you can set the return code to 0, -1, or a positive integer. NetView can set the return code to 0, -1, -2, or -3. You cannot code -2 or -3 on the &EXIT statement, but you can test for them. All other negative return codes are reserved.

Here are the possible values and meanings of &RETCODE:

Values	**Meaning**
0	No error.
Positive integer	You define the meaning. If &CONTROL ERR is in effect, the command is echoed to the screen.
-1	An error was found. This command list and all nested command lists end. Message DSI197I is issued for this command list.
-2	A command in the command list is not correct. The message DSI209I is displayed with the incorrect command. The command is ignored, and the command list continues.
-3	A command in the command list is not in the operator's scope of commands. The incorrect command list statement is displayed along with message DSI210I. The command is ignored, and the command list continues.
-5	A command list is terminated as the result of a RESET or other failure.

Message Processing Information

&MSGCNT
is the number of elements of text in a message string. &MSGCNT is used with &WAIT and with the LINKPD command.

See "Control and Parameter Variables Used with &WAIT" on page 116 for more information about using control variables with &WAIT.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

&MSGID
is the message identifier of the message most recently received by NetView. The message identifier is the first token of the message (up to 10 characters). If the first token is longer than 10 characters, &MSGID uses only the first 10 characters. If a reply ID is sent with the message, it is not used as the first token. For an MLWTO, &MSGID uses the first token of the first line of the message. &MSGID is used in message automation, with &WAIT, and with the LINKPD command. See Chapter 9, "Message Automation" on page 135 for more information about message automation.

See "Control and Parameter Variables Used with &WAIT" on page 116 for more information about using control variables with &WAIT.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

&MSGORIGIN
is the domain where the message originated. &MSGORIGIN is used for message automation, with &WAIT, and the LINKPD command.

See Chapter 9, "Message Automation" on page 135 for more information about message automation.

See "Control and Parameter Variables Used with &WAIT" on page 116 for more information about using control variables with &WAIT.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

&MSGSTR
is the message text of the message most recently received by NetView. &MSGSTR does not include the message identifier (the token used by the &MSGID control variable). &MSGSTR is used with &WAIT and with the LINKPD command.

See "Control and Parameter Variables Used with &WAIT" on page 116 for more information about using control variables with &WAIT.

See "LINKPD Results" on page 167 for more information about the LINKPD command.

&SESSID
is the TAF session ID where the message originated. &SESSID is used in message automation and with &WAIT. See Chapter 9, "Message Automation" on page 135 for more information about message automation.

Note: If TAF starts a session with an &SESSID equal to the domain ID, &SESSID will not be set correctly, and message automation may not work.

The remainder of this section contains descriptions of control variables that are used only for message automation. More information on these control variables can be found in *MVS Systems Programming Library: Systems Macros and Facilities, Vol. 2*.

Some of the message information control variables are filled with values only after NetView receives a message through the subsystem interface (SSI). The message information control variables that have null values until NetView receives a message through the SSI are:

- &AREAID
- &DESC
- &JOBNAME
- &JOBNUM
- &MCSFLAG
- &MSGTYP
- &ROUTCDE
- &SMSGID
- &SYSCONID
- &SYSID.

&AREAID
provides a one-letter (A-Z) identifier for the area on the console screen that displays the message.

&DESC
provides the system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.

&HDRMTYPE
provides the 1-character NetView message type of the message. NetView message types are described in *NetView Customization: Using Assembler*.

Note: &HDRMTYPE is a NetView-supplied message control variable.

&JOBNAME
provides the 1- to 8-character MVS JOB name identifier. Because the JOBNAME is the name of the job that originated the message, it may not always be the same as the name of the job to which the message is referring. For example, this can occur when MVS issues a message about the NetView job. Also, JOBNAME can contain the name of an initiator (instead of the actual jobname) when a job is started or terminated. If the message is issued during startup or termination, extract the job name from the message text rather than using the JOBNAME variable.

&JOBNUM
provides the 8-character MVS JOB number identifier. Depending on the MVS release, &JOBNUM can be a character string such as 'JOB   4', or simply a number such as '   4'.

Note: The appropriate number of blanks are imbedded within JOBNUM to ensure a total length of 8 characters.

&LINETYPE
provides the multi-line write-to-operator (MLWTO) line type, as follows:

C	The line is a message control line.
L	The line is a message label line.
D	The line is a message data line.
DE	The line is the last message data line.
E	The line is the last message line and contains no data.
blank	The message is a single-line message.

&MCSFLAG
provides the system message flags in a binary series of on (1) and off (0) codes corresponding to the following meanings:

first	Send message conditionally to console &SYSCONID
second	Send message unconditionally to console &SYSCONID
third	RESP
fourth	REPLY
fifth	BRDCST
sixth	HRDCPY only
seventh	NOTIME
eighth	NOCPY.

&MSGTYP
provides the system message type as three consecutive binary characters. An on character (1) in one of the positions corresponds to the following meanings:

first	SESS
second	JOBNAMES
third	STATUS.

&REPLYID
provides a three-character reply identifier for WTOR command replies. See "WTOR" on page 140 for more information about the WTOR command.

&ROUTCDE
provides the system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.

&SMSGID
provides an 8-character value that identifies a particular instance of a message. This control variable is used by the DOM command to identify action messages to be removed from the display. See "DOM" on page 141 for more information about DOM.

&SYSCONID
provides the console number (in decimal) that receives the message.

&SYSID
provides the identifier of the MVS system that sent the message.

&WTOREPLY
is the reply sent by the operator in response to a WTOR command. See "WTOR" on page 140 for more information about the WTOR command.

Panel Information

&VIEWAID
returns the AID key that the operator used to enter panel input.

The possible values and meanings for &VIEWAID are as follows:

Values	**Meaning**
PF 1 - 24	Programmed function (PF) key. PF1, PF2, ...PF24.
PA 1 - 3	Program access (PA) key. PA1, PA2, or PA3.
ENTER	The ENTER key.

&VIEWCURCOL
returns the panel column where the cursor was positioned when the AID key was pressed.

&VIEWCURROW
returns the panel row where the cursor was positioned when the AID key was pressed.

User Variables

User variables are variables you create and set within the command list. User variables are set with an assignment statement or an &PAUSE control statement. Figure 47 shows the syntax of an assignment statement.

```
&uservariable = expression
```

Figure 47. Assignment Statement Syntax

The user variable is set to the value of the expression. For example, &A = 3 sets the user variable &A to 3. You will learn more about assignment statements in "Assignment Statements" on page 90.

The &PAUSE control statement halts the command list, allows the operator to enter data, and picks up the value of the user variable from the operator when the command list continues. &PAUSE is described in "&PAUSE Control Statement" on page 97.

When you create user variables, observe the following rules:

- The first character must be an ampersand (&).
- The first character following the ampersand must be a letter or a symbol, not a number. Otherwise, NetView treats it as a parameter variable.
- The ampersand is followed by 1 to 11 characters. A-Z, 0-9, #, @, and $ are valid characters.
- Double-byte character set (DBCS) variable names are not supported. All variable names must be written using Latin characters. See "Conventions for Double-Byte Character Set Text" on page 74 for more information about DBCS characters in command lists.
- The value of the user variable can be 255 characters long. For DBCS characters, the maximum number of double-byte characters between the shift-out and shift-in is 126.
- You can give user variables a numerical value between -2147483647 and 2147483647. The only characters you can use in a numerical value are 0-9. The numerical value can be immediately preceded by a character indicating whether the value is positive (+) or negative (−).

Note: You can create a user variable that NetView has already defined as a control statement, control variable, or built-in function. However, you cannot use that NetView function in this command list.

Table 2 shows some examples of user variable names.

Table 2. Valid and Invalid User Variable Names

Valid	**Invalid**	**Reason Why Invalid**
&A	&2A	Will be read as &2, a parameter variable
&USERNAME	&INVALIDUSERNAME	Too long
&@23456	&A%	% is an invalid character

Figure 48 shows how to manipulate user variables in assignment statements to set parameters and to communicate with the operator.

```
    &PAUSE VARS &ONE &TWO
    &SUM = &ONE + &TWO
    CLEAR
    &WRITE >>> THE SUM OF &ONE + &TWO IS --->&SUM
```

Figure 48. User Variables in Command List Statements

Comments

You will find it very helpful to code comments in a command list. Command lists with comments are easier to maintain and expand.

You can use comments to show:

- When the command list was created and updated
- Who wrote the command list
- The function of the command list
- What input and output is expected
- Whether this command list depends on other programs or on other command lists.

To write a comment, code an asterisk (*) as the first non-blank character of the command list line. Ensure that you do not use a string of hyphens to separate sections of the command list.

Figure 49 shows how comments are used for a title block and other internal documentation.

```
******************************************************************
***
*** CLIST: SAMPLE
*** DATE: 1-2-87   BY: JANE DOE
*** PURPOSE:
***    SAMPLE CLIST TO SHOW USE OF CONCEPTS OF CHAPTER 2 IN THE
***    CLIST MANUAL
*** INPUT:   'SAMPLE' FOLLOWED BY A MESSAGE YOU WANT DISPLAYED
***    ON THE TERMINAL
******************************************************************
***
*** FIRST ISSUE THE CLEAR COMMAND TO GIVE US A CLEAN SCREEN
CLEAR
*** NOW LETS SHOW THE USE OF &PARMSTR AND POSITIONAL VARIABLES,
*** AND THEN GET SOME USER INPUT...
```

Figure 49. Comment Statements for Internal Documentation

Null Statements

Another type of command list statement is the null statement. A null statement contains all blanks or a label followed by all blanks. A null statement with a label can be the target of flow control (conditional processing) statements or &BEGWRITE statements. See "Labels" on page 75 for details about using labels.

You can use a null statement to help format a message to the operator or to break up a long command list so that it is easier to read and update. If a null statement is part of a message written with an &BEGWRITE statement, it is sent to the operator as a blank line. If a null statement is used to break up the command list, it is ignored by NetView when the command list is run.

Assignment Statements

Assignment statements are used to give values to variables and do arithmetic operations within a command list. Figure 50 shows the syntax of an assignment statement:

```
&variable = expression
```

Figure 50. Assignment Statement Syntax

There must be a blank before and after the equal sign.

When the command list runs, the value of the user variable is set to the value of the expression. For example, if you had the assignment statement &A = 5, the value of &A becomes 5. If you had the assignment statement &B = &1, the value of &B is set to the value of &1, and &1 keeps its value.

An expression is one of the following:

A Constant

A constant consists of alphanumeric characters that are not replaced by other values. The values are fixed. For example, if you code the following assignment statement:

```
&VAR = 5
```

the value 5 is assigned to user variable &VAR.

If you want to use a constant string that contains a blank, comma, apostrophe, or hyphen, make it a special character string by using single quotes. For example:

```
&NAME = 'JOHN B. DOE'
```

is a constant string containing blanks.

The constant cannot be longer than 255 characters. If it is a number, the constant must be between -2147483647 and 2147483647. The only characters you can have in a numerical value are 0-9. The numerical value can be immediately preceded by a character indicating whether the value is positive (+) or negative (−).

A Variable

A variable can be a parameter variable, control variable, user variable, or global variable.

The following assignment statement:

```
&PARMVAR = &4
```

assigns the value of parameter variable &4 to user variable &PARMVAR.

To assign the value of control variable &OPID to user variable &USERVAR, code the following:

```
&USERVAR = &OPID
```

Note: You cannot use a control statement as a variable, even if the control statement is enclosed in single quotes. For example, you could **not** have the following assignment statements:

```
&A = &IF
&A = '&WAIT ERROR'
```

An Arithmetic Operation

The addition and subtraction operations are allowed in an assignment statement. The format is two numbers separated by a plus (+) or minus (-) sign. You can also use a variable that will be set to a number. The only characters you can use in a numerical value are 0-9. The numerical value can be immediately preceded by a character indicating whether the value is positive (+) or negative (−).

The plus or minus sign must be separated from the numbers on each side by at least one blank unless it indicates a positive or negative number (-2, -4). For example, both 4 - 2 and 4 - -2 are correct, but 4 -2 will not work.

The results of the arithmetic operation must be between -2147483647 and 2147483647.

The following assignment statement shows how you can use a control variable in an arithmetic operation:

```
&SUM = 38 — &PARMCNT
```

The value of control variable &PARMCNT is subtracted from 38, and the resulting value is assigned to user variable &SUM.

In arithmetic expressions with leading zeros, the leading zeros are not shown in the result. For example, assume &A is 01 and you code the following:

```
&C = &A + 1
```

The value of &C becomes 2, not 02.

A Built-In Function

A built-in function can be used in an assignment statement. The result of the operation is placed in the user variable. See "Built-In Functions" on page 99 for a detailed description.

The following examples show how to code built-in functions in assignment statements:

```
&STR2 = &SUBSTR &STRING 2 1
&STR1 = &SUBSTR &STRING 1 1
&NEWSTR = &CONCAT &STR5 &STR4
&NEWSTR = &CONCAT &NEWSTR &STR3
```

To review a command list that contains all of the concepts covered in this chapter, see "Sample Command List—Chapter Review" on page 105.

Control Statements

Control statements are unique command list statements that the way NetView acts on other statements in the command list. The control statements in this chapter can be used for either straight-line coding or in conjunction with the statements described in Chapter 7, "NetView Command List Language Branching" on page 107 for structured conditional processing.

The NetView command list language lets you use structured programming techniques in writing your automation applications. You can use control statements to to change the strict sequential order of processing. Command list control statements allow you to do the following:

- Send messages to the operator from the command list
- Control the order in which commands are run
- Ask the operator to enter information needed to continue the command list
- Wait for a solicited message to arrive before continuing the command list.

Each command list control statement begins with the control symbol in the form *&word.* Only one control statement can be coded on a line, except when using &IF.

After reading the descriptions of the control statements, you should have a general idea of the capability of these basic statements. Read the sections that follow for details concerning each control statement.

The control statements are:

&CONTROL
indicates the command list statements shown on the operator's screen while the command list is running.

&WRITE
writes a message to the designated operator.

&BEGWRITE
writes a message or series of messages to the operator. &BEGWRITE is a shortened form of begin writing.

&PAUSE
halts the command list until the operator enters information needed to continue the command list.

&CONTROL Control Statement

&CONTROL lets you indicate which command list statements are displayed at the operator's terminal while the command list is running. The indicated command list statements are displayed after all substitutions have been made and before the command list statements run. The display of the command list statements from &CONTROL ALL or &CONTROL CMD can be used to help debug your command list.

Set &CONTROL at the beginning of the command list. You can change the &CONTROL setting within the command list as many times as necessary. &CONTROL is in effect from that point in the command list until the next &CONTROL statement is reached. For instance, if you just added a new section of code to a command list, you can display the entire new section of code but view only the errors for the existing sections of code. Code this control statement by typing &CONTROL followed by a blank and then by an operand. Figure 51 on page 93 shows the syntax of the &CONTROL control statement.

```
&CONTROL     [ALL|CMD|ERR]
```

Figure 51. &CONTROL Control Statement Syntax

&CONTROL ALL
: displays all command list statements at the operator's terminal. Each statement is displayed just before it is processed. This is a good choice when you first write the command list and want to test it. Once your command list is tested, &CONTROL CMD or &CONTROL ERR is a better choice. When processing for this command list is complete, message DSI013I, COMMAND LIST *clistname* COMPLETE is displayed. If you code &CONTROL without any operands, or if you do not code &CONTROL, &CONTROL ALL is the default.

&CONTROL CMD
: displays all commands at the operator's terminal. Each command is displayed just before it runs. The other command list statements—such as comments, control statements, and other command list language statements—are not displayed unless they contain an error. When processing for this command list is complete, message DSI013I, COMMAND LIST *clistname* COMPLETE is displayed.

&CONTROL ERR
: displays only statements that contain errors and commands that have non-zero return codes. If &CONTROL ERR is in effect at the end of a command list, message DSI013I is not displayed.

The control statement, &CONTROL ERR, is coded in "Sample Command List—Chapter Review" on page 105 for displaying incorrect statements and those commands with non-zero return codes.

Writing to the Operator

&WRITE and &BEGWRITE send messages to the operator terminal. &WRITE only sends a one line message, whereas &BEGWRITE allows multi-line messages to be sent. These statements are used, for example, to tell the operator what the command list is doing.

The messages are sent to the operator regardless of the &CONTROL setting. If you code a command on an &WRITE control statement, the text is sent to the operator as a message, but it is not run as a command list command.

Do not confuse the use of &WRITE and &BEGWRITE with the use of command list comments. Comments are for the person writing the command list and are not sent to the operator unless &CONTROL ALL is set; &WRITE and &BEGWRITE send messages to the operator.

If you are sending more than one message line or displaying a table that takes up the whole screen, you might want to issue the NetView CLEAR command first. The CLEAR command erases the screen and causes whatever you are writing to the operator to begin at the top of the screen. If you do not want commands and control statements that complete correctly to be displayed with what you are writing to the operator, make sure &CONTROL ERR is in effect before issuing the CLEAR command.

&WRITE Control Statement

&WRITE sends one line of text to the operator. NetView performs variable substitution on the message text before sending the message to the operator. If you do not want substitution performed on the message text, use &BEGWRITE. If you do not include message text, NetView sends a blank line to the operator. The syntax for &WRITE statements is shown in Figure 52.

```
&WRITE        message text
```

Figure 52. &WRITE Control Statement Syntax

If you want to include blanks in front of the first character of the line, code a non-blank character after &WRITE.

The period causes this line:

```
&WRITE .      THIS LINE WILL START IN COLUMN 8
```

to print like this:

```
.      THIS LINE WILL START IN COLUMN 8
```

Otherwise, the line will shift left until the first non-blank character is in column 1.

The following line has no period:

```
&WRITE             THIS LINE WILL SHIFT TO COLUMN 1
```

so it prints like this:

```
THIS LINE WILL SHIFT TO COLUMN 1
```

Figure 53 is an example of a command list called PATH that uses the &WRITE control statement and a VTAM command.

```
PATH  CLIST
&CONTROL CMD
* THIS COMMAND LIST DISPLAYS INFO ON VTAM SWITCHED PATHS
&WRITE *** STATUS OF VTAM SWITCHED PATHS FOR &1 ***
D NET,PATHS,ID=&1
```

Figure 53. Example Command List Using &WRITE

Activating this command list by entering PATH HD3790N1 causes the operator to see a display similar to the following.

```
*** STATUS OF VTAM SWITCHED PATHS FOR HD3790N1 ***
D NET,PATHS,ID=HD3790N1
IST097I  DISPLAY ACCEPTED
IST148I  DIAL OUT PATH INFORMATION FOR PHYSICAL UNIT HD3790N1
IST149I  LINE GRP   TELEPHONE NUMBER OR LINE NAME  PID GID CNT
IST168I  EGROUP40                 4094             001 001 005 AVA
AUT
IST168I  EGROUP50                 4094             002 002 001 AVA
MAN
IST314I  END
```

Figure 54. Result of PATH Example Command List

Notice that the &1 in the &WRITE statement is replaced by the value HD3790N1 before it is sent to the operator. Because &CONTROL CMD was coded, the command is also shown. The rest of the display is the response to the VTAM command.

Figure 55 shows several &WRITE statements, which send one-line messages to the operator.

```
CLEAR
&WRITE >>> THE SUM OF &ONE + &TWO IS --->&SUM

&WRITE THE MIRROR IMAGE IS: &NEWSTR

&WRITE TOTAL CHARACTERS ENTERED: &LEN

&WRITE *** END OF SAMPLE CLIST, BYE  ***
```

Figure 55. &WRITE Statements to Send Operator Messages

&BEGWRITE Control Statement

You can use &BEGWRITE to write a series of lines to the operator terminal. You can also control whether variables are replaced before sending the messages. The syntax for &BEGWRITE statements is shown in Figure 56.

&BEGWRITE [**SUB**| **<u>NOSUB</u>**][−*label*]

Figure 56. &BEGWRITE Control Statement Syntax

&BEGWRITE is coded differently than &WRITE. You code &BEGWRITE on a line by itself, one line above the first operator message you wish to send. You can also specify a label on &BEGWRITE. The label tells the command list where the messages end and command list processing continues. See "Labels" on page 75 for more information about labels.

You can indicate that you want variables replaced by their actual values before the messages are sent to the operator. If you do not indicate a choice, variables are *not* replaced.

&BEGWRITE SUB *-label*

causes NetView to carry out substitution on the message text before sending the messages to the operator. See "Variable Substitution Order" on page 76 for information on how NetView carries out variable substitution.

If there are blanks before the first character on a message line, the line is shifted left until the first non-blank character is in column 1. If you want the blanks sent to the operator's screen, code a nonblank character in column 1. If you are using &BEGWRITE to write a message containing double-byte character set (DBCS) characters, you must use the SUB option. These coding rules are the same as those for &WRITE.

&BEGWRITE <u>NOSUB</u> *-label*

writes the messages to the operator exactly as they are typed, with no variable substitution. In other words, &1 is sent as &1, not as the value of &1. Use this operand to write about the command list variables in your messages. NOSUB does not remove blanks. It displays the text exactly as it is entered. If you code &BEGWRITE without an operand, NetView assumes NOSUB.

-label

indicates the line that follows the text to be displayed to the operator. If you code a label in the statement, this label must be on a statement following the end of the message text lines in the command list. The command list lines between &BEGWRITE and the statement with the label are sent to the operator. The command list statement with the label is *not* sent to the operator; it is processed as the next command list statement. If NetView cannot find the label, the rest of the command list statements are sent to the operator as comments and the command list is ended. If there is no label on &BEGWRITE, only the first command list statement after &BEGWRITE is sent to the operator.

You can code a variable for your label on &BEGWRITE, but make sure the variable is replaced by a valid value.

Figure 57 shows an example of &BEGWRITE with variable substitution.

```
     &BEGWRITE SUB -ENDTEXT
.
>>> HELLO &OP.
>>> YOU CAN INITIATE CROSS-DOMAIN SESSIONS WITH &ID.
.
.   .NOW FOR SOME CHARACTER MANIPULATION
.    ENTER 'GO' FOLLOWED BY A FIVE CHARACTER STRING.
.    THE CLIST WILL PRINT OUT THE MIRROR IMAGE TO YOU.
.
-ENDTEXT
```

Figure 57. &BEGWRITE with Variable Substitution

In some cases, you might not want variable substitution. In Figure 58, the &BEGWRITE shows the operator how to use the ENDIT command list.

```
&CONTROL ERR
&BEGWRITE NOSUB -OVER
TO END FULL SCREEN SESSIONS,
TYPE "ENDIT &1,&2,&3"
REPLACE &1,&2,&3 WITH
THE APPLID NAMES OF THE
FLSCN SESSIONS TO BE ENDED
-OVER
```

Figure 58. &BEGWRITE with No Variable Substitution

The ENDIT command list is called by entering ENDIT. The operator sees the following messages:

```
TO END FULL SCREEN SESSIONS,
TYPE "ENDIT &1,&2,&3"
REPLACE &1,&2,&3 WITH
THE APPLID NAMES OF THE
FLSCN SESSIONS TO BE ENDED
```

Figure 59. Result of ENDIT Example Command List

Notice that &1, &2, and &3 are not replaced by their values when the messages are sent to the operator.

&PAUSE Control Statement

Using the &PAUSE control statement along with other commands, you can code command lists that ask the operator questions and pick up the entered responses. Use the &BEGWRITE and &WRITE control statements to send instructions to the operator. For example, you can code the command list to instruct the operator to enter the NetView GO command followed by a value or values for a user variable. Then code the &PAUSE statement to temporarily halt the command list. The command list pauses until the operator enters the GO command to continue processing or the RESET command to end the command list. The &PAUSE command can be coded to allow the command list to pick up the operands following the GO commands and take them as user variables. See "User Variables" on page 87 for more information about this subject.

Note: There are times when you cannot use &PAUSE. Do not use &PAUSE in an automation task command list or a command list that runs under the Primary POI Task (PPT). For more information about automation task command lists, see "NetView Release 3 Automation Task" on page 136. For more information on PPT restrictions, see "Primary POI Task Restrictions" on page 17.

You can code &PAUSE three different ways. Figure 60 shows the syntax for &PAUSE statements.

&PAUSE [<u>**NOINPUT**</u>|**VARS** *variable* [...]|**STRING** *variable*]

Figure 60. &PAUSE Control Statement Syntax

&PAUSE <u>NOINPUT</u>

pauses until the operator enters the GO or RESET command. No operands are allowed with the GO command. If the operator enters operands, an error message is displayed. If you code &PAUSE without any operands, &PAUSE NOINPUT is assumed.

&PAUSE VARS variable [...]

pauses until the operator enters the GO command with or without the correct number of operands, or the RESET command. The operator is told by a previous &WRITE or &BEGWRITE statement to enter operands with the GO command. Each operand is taken as a user variable coded on the &PAUSE VARS statement. These variables can then be used in the command list.

&PAUSE STRING variable
pauses until the operator enters the GO command with or without a string, or the RESET command. The operator is told by a previous &WRITE or &BEGWRITE statement to enter operands with the GO command. The entire string of operands is taken as one user variable. The variable can then be used in the command lists.

When the command list interprets an &PAUSE control statement, NetView puts the letter P on the upper right-hand corner of the terminal screen to show the operator that the command list is in pause state. Pause state means that the command list has halted and is waiting for input from the terminal.

Note: If a command list in pause state was called by an NNT session, the P indicator is not displayed on the OST screen.

Using NetView Commands with &PAUSE

The operator can enter the NetView commands GO, RESET, STACK, and UNSTACK during a pause. See *NetView Operation* for details of these network commands.

STACK and UNSTACK let the operator suspend and then resume command list processing during an &PAUSE. Once the STACK is issued, the operator can enter any network commands.

Note: While an &PAUSE is suspended with the STACK command, the P is removed from the upper right-hand corner of the screen. The P reappears after UNSTACK is issued. After UNSTACK is issued, the operator enters GO either with or without operands to continue the command list, or enters the RESET command to end the command list. RESET also ends any nested command lists.

The operands on the GO command are positional. This means the first operand becomes the first user variable, the second operand becomes the second user variable, and so on. Operands are separated by either a blank or a comma. If you want to include a blank or a comma as part of one variable, use either &PAUSE STRING or put the operand between single quotes.

You should code a user variable for each expected operand. If the operator enters more operands on the GO command than expected by the command list, the extra operands are ignored. If the operator enters fewer operands than expected, the remaining variables are set to null. The operator can also skip over one operand by coding two commas in a row.

You should always precede pauses for operator input with messages telling the operator what to enter. Use the &WRITE or &BEGWRITE statements to send this information.

Note: It is important to remember that the operator can invoke your command list from any NetView component. If you expect the command list to run from components other than the command facility, use the NetView command NCCF in the command lists to present the operator with the command facility screen and command screen input area. (Do this before issuing any messages.) If the command list is running in the command facility, the NCCF command has no effect.

An Example Using &PAUSE

Figure 61 contains a portion of a command list that shows how you can ask for information from an operator.

```
&BEGWRITE SUB -ENDTEXT
.   ENTER 'GO' FOLLOWED BY YOUR LAST NAME,
.   FIRST NAME, AND MIDDLE INITIAL.
-ENDTEXT
* GET THE INPUT FROM THE USER
&PAUSE VARS &LAST &FIRST &MI
```

Figure 61. Example &PAUSE Statement

The example writes a message to the operator asking for the operator's last name, first name, and middle initial. The command list pauses until the operator enters a GO or RESET command. To continue processing the current command list, the operator enters the GO command followed by the string required by the command list.

If the operator enters:

```
GO SMITH JOHN A
```

the value of &LAST becomes SMITH, the value of &FIRST becomes JOHN, and the value of &MI becomes A. These variables can then be used by other statements in the command list.

Built-In Functions

Built-in functions perform predefined operations. They are used as expressions either in an assignment statement or in an &IF control statement. See "&IF Control Statement" on page 107 for information on the &IF control statement. In an assignment statement, the value of the user variable is set to the result of the built-in function's operation.

Be careful not to confuse built-in functions with variables. Although they appear similar, they are not the same. A built-in function looks like a variable because they both start with an ampersand (&). Here is the difference:

- A variable is replaced by its value when the command list runs. The variable is really just a place holder for the value.
- A built-in function is never replaced by a value. It is an action indicator rather than a place holder.

These are the built-in functions you can use:

- &CONCAT
- &LENGTH
- &NCCFID
- &NCCFSTAT
- &SUBSTR.

Figure 62 shows the format to use for coding built-in functions for an assignment statement.

```
&uservariable = Built-In Function {&variable|constant}
```

Figure 62. Syntax for Coding Built-in Functions in an Assignment Statement

The examples in this section use built-in functions in assignment statements. Examples with built-in functions in the &IF control statement are in "&IF Control Statement" on page 107.

In an &IF control statement, the result of the built-in function is used as one or both of the compared expressions. For example, you might use the &LENGTH built-in function to compare the lengths of two variables.

&CONCAT Built-In Function

&CONCAT concatenates the values of two variables, two constants, or a variable and a constant to form a new value. &CONCAT is a shortened form of concatenate. The syntax of the &CONCAT built-in function is shown in Figure 63.

```
&uservariable = &CONCAT {&variable|constant} {&variable|constant}
```

Figure 63. Syntax for &CONCAT in Assignment Statements

Ensure that when the two items are joined, the resulting value does not exceed 255 characters. If the combined value exceeds 255 characters, it is truncated to 255 characters. If the value of both items being joined is null, the value of *&uservariable* is null.

When &CONCAT is used to concatenate two double-byte character set (DBCS) strings it removes adjacent shift-in (SI) and shift-out (SO) characters.

Figure 64 shows how &CONCAT is coded to concatenate the values of five variables into one new variable.

```
    &NEWSTR = &CONCAT &STR5 &STR4
    &NEWSTR = &CONCAT &NEWSTR &STR3
    &NEWSTR = &CONCAT &NEWSTR &STR2
    &NEWSTR = &CONCAT &NEWSTR &STR1
```

Figure 64. &CONCAT Function to Build a Character String

&LENGTH Built-In Function

&LENGTH returns the length of a variable or a constant. Figure 65 shows the syntax of &LENGTH.

```
&uservariable = &LENGTH {&variable|constant }
```

Figure 65. Syntax of &LENGTH in Assignment Statements

The value of *&uservariable* is set to the length of the constant or variable. If the variable on the right of the equal sign is null, the length is 0, and the value of the user variable becomes 0.

Figure 66 is an example of you how to use &LENGTH. Suppose you called command list SAMP by entering SAMP LU2525. Assume the name of the hard-copy printer (&HCOPY) control variable was HC55.

```
SAMP CLIST
&HCLENGTH = &LENGTH &HCOPY
&RESLEN = &LENGTH &1
```

Figure 66. Example Command List Using &LENGTH

After processing, the variable settings are:

Variable	Value
&HCOPY	HC55
&HCLENGTH	4
&1	LU2525
&RESLEN	6

User variable &HCLENGTH is set to the length of the hard-copy device name. The hard-copy device is HC55. HC55 has four characters, so &HCLENGTH becomes 4. &RESLEN becomes the length of the first parameter variable. The first parameter variable is LU2525, so &RESLEN becomes 6.

&NCCFID Built-In Function

&NCCFID indicates the NetView domain identifier of a domain with which you can establish a cross-domain session. The value of &NCCFID is not necessarily the domain identifier of your domain. To use this built-in function, code &NCCFID followed by a number. For more information on defining domains to NetView, see *NetView Installation and Administration Guide* and *NetView Administration Reference*.

The command list can use &NCCFID to automatically start or stop a cross-domain session. Figure 67 shows the syntax of NCCFID.

&uservariable = **&NCCFID** *number*

Figure 67. Syntax for &NCCFID in Assignment Statements

The number is either a constant or a variable. The largest number permitted is the value of &NCCFCNT, the control variable that shows the total number of cross-domain sessions this operator can start.

Figure 68 on page 102 is an example of you how to use &NCCFIC.

```
&DOM1 = &NCCFID 1
&DOM2 = &NCCFID 2
&DOM3 = &NCCFID 3
START DOMAIN=&DOM1
START DOMAIN=&DOM2
START DOMAIN=&DOM3
```

Figure 68. Using &NCCFID Function to Start a Cross-Domain Session

Assume the DOMAINS table has these entries:

1	ALPHA
2	BETA
3	GAMMA

After processing, the user variables are set as follows:

Variable	Value
&DOM1	ALPHA
&DOM2	BETA
&DOM3	GAMMA

The command list uses &NCCFID to index the first three entries of the DOMAINS table. &DOM1 is set to ALPHA, the first domain listed. &DOM2 is set to BETA, the second domain. &DOM3 is set to the third domain, GAMMA. These three domains are then started with the NetView START command.

In this example, the operator must know there are three domains that can be started. You can also use the &IF control statement to test &NCCFCNT to find the number of domains and start them.

&NCCFSTAT Built-In Function

&NCCFSTAT indicates whether you have an active cross-domain session with a domain. Figure 69 shows the syntax of &NCCFSTAT.

```
&uservariable = &NCCFSTAT domain
```

Figure 69. Syntax for &NCCFSTAT in Assignment Statements

In this case, *domain* is either a domain name or a variable that becomes a domain name.

The *&uservariable* variable is replaced by the characters ACT if the operator has an active cross-domain session with the domain. The user variable is replaced by the characters INACT if the operator does not have an active cross-domain session with the domain.

For example, you can write a command list to check the status of a domain and start that domain if it is not active. Assume you activated the STARTEM command list in Figure 70 on page 103 by entering STARTEM NCCFA.

```
STARTEM  CLIST
&CONTROL ERR
&STATUS = &NCCFSTAT &1
&IF &STATUS = INACT &THEN START DOMAIN=&1
&IF &STATUS = ACT &THEN &WRITE DOMAIN &1 IS ALREADY ACTIVE
```

Figure 70. Example Command List Using &NCCFSTAT

After processing, the variables are set as follows:

Variable	Value
&1	NCCFA
&STATUS	ACT|INACT

The parameter variable &1 is set to NCCFA, and the status of domain NCCFA is checked. If you have an active cross-domain session with NCCFA, &STATUS is set to ACT. If not, &STATUS is set to INACT. The &IF statement tests whether &STATUS is set to ACT or INACT (for more information, see "&IF Control Statement" on page 107).

If NCCFA is inactive, the command list starts it. If NCCFA is active, you receive this message:

```
DOMAIN NCCFA IS ALREADY ACTIVE
```

&SUBSTR Built-In Function

&SUBSTR uses part of a variable to form the value of a new user variable. &SUBSTR is a shortened form of substring. Figure 71 shows the syntax of &SUBSTR.

&uservariable = **&SUBSTR** *&variable start* [*length*]

Figure 71. Syntax for &SUBSTR in Assignment Statements

&SUBSTR takes the variable and starts at position *start* for *length* characters. Suppose you had the following statements:

```
&HOLD = ACF/VTAM
&FIRST = &SUBSTR &HOLD 1 3
&SECOND = &SUBSTR &HOLD 5 4
```

Figure 72. Example Command List Using &SUBSTR

After processing, the user variables are set as follows:

Variable	Value
&HOLD	ACF/VTAM
&FIRST	ACF
&SECOND	VTAM

The first line of Figure 72 sets the value of variable &HOLD to ACF/VTAM. In the next line, &SUBSTR starts at the first character of &HOLD (the letter A) and moves three characters to the right (to the character F). The letters ACF become the value of the variable &FIRST. In the last line, &SUBSTR starts at the fifth character of &HOLD (the letter V) and goes for a length of four (to the character M). The letters VTAM are put into variable &SECOND. The starting positions are determined as shown:

1	2	3	4	5	6	7	8
A	C	F	/	V	T	A	M

Note: The first starting position is 1, the second is 2, and so on. *Zero is not a valid position*. Because the largest variable value is 255 characters, it is invalid to have a starting point greater than 255.

You do not have to specify a length. If the length is not specified, the remainder of the string to the right beginning with the starting position becomes the substring. NetView never pads substrings with blanks. If you specify a length that is too long, NetView assumes no length was specified and uses the entire string beginning at the starting position. If the length is 0, or the starting position is beyond the variable length, the result of &SUBSTR is null.

Figure 73 shows how you can use a substring of the &APPLID control variable to determine the name of the domain running the command list.

```
GETDOMID  CLIST
&CONTROL ERR
* DETERMINE THE LENGTH OF THE APPL ID
&LENAPPL = &LENGTH &APPLID
* SUBTRACT 3 TO GET THE LENGTH OF THE DOMAIN ID
&LENAPPL = &LENAPPL - 3
* START AT COLUMN 1 OF NEW LENAPPL FOR LENGTH OF DOMAIN ID
* THE VALUE OF &DOMAIN WILL BE THE DOMAIN ID
&DOMAIN = &SUBSTR &APPLID 1 &LENAPPL
* &DOMAIN NOW CONTAINS THE DOMAIN ID
```

Figure 73. Using &SUBSTR to Find the Domain Name from &APPLID

Using &SUBSTR with DBCS Characters

When using double-byte character set characters along with Latin characters (A-Z; a-z), &SUBSTR will adjust the variable as follows:

Start byte = shift-out character	No adjustment
Start byte = shift-in character	Replace by blank
Start byte = first half of double-byte	Replace by blank and shift-out character
Start byte = second half of double-byte	Replace by shift-out character
Last byte = shift-out character	Replace by blank
Last byte = shift-in character	No adjustment
Last byte = first half of double-byte	Replace by shift-in character
Last byte = second half of double-byte	Replace by shift-in character and blank.

Following is an example of the &SUBSTR statement used on a double-byte character set (DBCS) and Latin character string:

```
&DBCS = 'AB<D1D2D3>EFG'
```

where A, B, E, F, G are Latin characters, < is the shift-out character, > is the shift-in character, and D1, D2, D3 are double-byte characters. Using this value, &SUBSTR works like this:

```
&FIRST= &SUBSTR &DBCS 1 3
       = 'AB<' (interim string)
       = 'AB ' (recovery string)

&SECOND = &SUBSTR &DBCS 3 5
        = '<D1D2' (interim string)
        = '<D1> ' (recovery string)

&THIRD = &SUBSTR &DBCS 4 5
       = 'D1D2D' (interim string)
       = ' <D2D' (interim string)
       = ' <D2>' (recovery string)
```

Note: The DBCS delimiters are 1 byte long; the DBCS codes are 2 bytes long.

Sample Command List—Chapter Review

```
SAMPLE  CLIST
&CONTROL ERR
*******************************************************************
***
*** CLIST: SAMPLE
*** DATE: 1-2-87   BY: JANE DOE
*** PURPOSE:
***    SAMPLE CLIST TO SHOW USE OF CONCEPTS OF CHAPTER 6 IN
***    NETVIEW CUSTOMIZATION: WRITING COMMAND LISTS
*** INPUT:   'SAMPLE' FOLLOWED BY A MESSAGE YOU WANT DISPLAYED
***    ON THE TERMINAL
*******************************************************************
***
*** FIRST ISSUE THE CLEAR COMMAND TO GIVE US A CLEAN SCREEN
CLEAR
*** NOW LETS SHOW THE USE OF &PARMSTR AND POSITIONAL VARIABLES,
*** AND THEN GET SOME USER INPUT...
&BEGWRITE SUB -ENDWRITE
*******************************************************************
***
***          &PARMSTR
***-------------------------------------------------------------***
***  &1 &2 &3 &4 &5
***         &6 &7 &8 &9 &10
***                 &11 &12 &13 &14 &15
*******************************************************************
```

Figure 74 (Part 1 of 2). Review Command List

```
.
.    HI, THIS IS A MANIPULATION CLIST
.        &TIME       &DATE
.
.    ENTER 'GO' FOLLOWED BY TWO NUMBERS
.    AND THIS CLIST WILL RETURN THE SUM
-ENDWRITE
     &PAUSE VARS &ONE &TWO
     &SUM = &ONE + &TWO
     CLEAR
     &WRITE >>> THE SUM OF &ONE + &TWO IS ---> &SUM
*
*** LETS DEMONSTRATE THE USE OF SOME CONTROL VARIABLES
*** THEN ASK FOR MORE USER INPUT
     &ID = &NCCFID 1
     &OP = &OPID
     &BEGWRITE SUB -ENDTEXT
.
>>> HELLO &OP.
>>> YOU CAN INITIATE CROSS-DOMAIN SESSIONS WITH &ID.
.
.   NOW FOR SOME CHARACTER MANIPULATION
.   ENTER 'GO' FOLLOWED BY A FIVE-CHARACTER STRING.
.   THE CLIST WILL PRINT OUT THE MIRROR IMAGE TO YOU.
.
-ENDTEXT
***
*** GET THE INPUT FROM THE USER
    &PAUSE VARS &STRING
*** REVERSE FIVE CHARACTERS BY SEPARATING THE CHARACTERS
*** USING THE &SUBSTR FUNCTION THEN RECOMBINING THEM USING
*** THE &CONCAT FUNCTION
***
    &STR5 = &SUBSTR &STRING 5 1
    &STR4 = &SUBSTR &STRING 4 1
    &STR3 = &SUBSTR &STRING 3 1
    &STR2 = &SUBSTR &STRING 2 1
    &STR1 = &SUBSTR &STRING 1 1
    &NEWSTR = &CONCAT &STR5 &STR4
    &NEWSTR = &CONCAT &NEWSTR &STR3
    &NEWSTR = &CONCAT &NEWSTR &STR2
    &NEWSTR = &CONCAT &NEWSTR &STR1
***
    &WRITE THE MIRROR IMAGE IS: &NEWSTR
***
*** TELL THEM HOW MANY CHARACTERS WERE ENTERED
*** BY USING THE &LENGTH FUNCTION
    &LEN = &LENGTH &STRING
    &WRITE TOTAL CHARACTERS ENTERED: &LEN
***
*** LET THE USER KNOW THIS CLIST IS DONE
    &WRITE *** END OF SAMPLE CLIST, BYE ***
    &EXIT
```

Figure 74 (Part 2 of 2). Review Command List

Chapter 7. NetView Command List Language Branching

This chapter guides you through the NetView command list language features that let you code conditional and unconditional branching logic in command lists.

The &IF statement allows you to perform conditional branching based on logical or arithmetical comparisons. The result of a test or comparison in an &IF statement determines which alternative to perform. Conditional processing statements give you the flexibility to code if-then and loop structures.

The &GOTO statement allows you to perform unconditional branching.

The &EXIT statement lets you code logical exit points within a command list.

The &WAIT statement allows you to wait for expected events before processing continues.

See "Examples Comparing REXX and NetView Command List Language" on page 193 for samples of NetView command list language command lists that show how these control statements can be used. The equivalent REXX command lists are also included.

&IF Control Statement

You can initiate a conditional branch by coding an &IF control statement. The &IF control statement lets you specify processing based on a certain condition. The condition is formed with two expressions and a logical or arithmetical operator.

A logical or arithmetical expression is evaluated while processing the &IF statement. When the condition is true, the &THEN clause is processed. When the condition is false, processing continues at the statement following the &IF control statement. Figure 75 shows the syntax of the &IF control statement:

&IF *comparison* **&THEN** *statement*

Figure 75. &IF Control Statement Syntax

comparison
The comparison clause is in the form: expression1 symbol expression2.

expression1 is any expression that can be used in an assignment statement. It can be a constant, a variable, an arithmetic operation, or a built-in function. For more information on these, see "Assignment Statements" on page 90.

symbol stands for the logical or arithmetical operator in the comparison clause. It is coded with one of the following mathematical symbols:

Symbol		**Meaning**
=	(or EQ)	Equal
¬=	(or NE)	Not equal
<	(or LT)	Less than
>	(or GT)	Greater than

<=	(or LE)	Less than or equal
>=	(or GE)	Greater than or equal
¬>	(or NG)	Not greater than
¬<	(or NL)	Not less than

Note: You can use either the symbol code or the 2-character letter code. Both mean the same thing.

expression2 is the second term of comparison. It follows the same rules as expression1.

&THEN
separates the comparison from the command list statement that is processed if the condition is true. You must code &THEN in every &IF statement.

Note: Be sure to code the ampersand (&) with &THEN. The ampersand identifies the word as part of the control statement.

statement
The command list statement that is processed if the comparison is true. If the comparison is not true, this statement is ignored. The statement can be any NetView command list statement.

Variables coded in the comparison expressions are replaced by their values before the comparison is checked. If the variable has a null value, you get an error. For example, if you code the expression &A = &B, and &B is null, NetView cannot do the comparison. To avoid problems, put a period as the first character of each expression where a null value is possible. For example, the following line shows this suggested solution.

```
&IF .&A = .&B &THEN &GOTO -label
```

Figure 76. Suggested &IF Coding to Avoid Problems with Null Values

If either &A or &B is null, the comparison fails, but you do not get an error. If &A and &B are both 6, NetView reads the statement as .6 = .6 and the comparison is still true. You can use a period to test if a variable is null. For example, the comparison .&1 = . is true when &1 is null.

You cannot use this code suggestion with arithmetical operations. In this case, ensure that the result is not null to avoid receiving an error.

Figure 77 shows some examples of comparisons.

```
.5 = .&A
.&1 = .
2 + 2 NE &ANSWER
&PARMCNT LE 5
7 > 3 + &1
```

Figure 77. Examples of Arithmetical Comparisons

Figure 78 shows four examples that use the &IF control statement.

```
&IF &APPLID = NCCFA001 &THEN &USERVAR = 10

&IF &NCCFID = NCCFA &THEN &GOTO -PROC2

&IF &1 = LU200 &THEN VARY NET,ACT,ID=&1

&IF &SUBSTR &DATE 1 5 = 12/25 &THEN &WRITE HAPPY HOLIDAY
```

Figure 78. Example Statements Using &IF Control Statement

&GOTO Control Statement

&GOTO unconditionally transfers control to another part of the command list. &GOTO lets you rerun statements or jump ahead to a statement of the command list. A statement label identifies the target or destination statement. When you use both &IF and &GOTO, you can test for various conditions and go to different parts of the command list, depending on the results. Figure 79 shows the syntax of the &GOTO control statement.

&GOTO *-label*

Figure 79. &GOTO Control Statement Syntax

-label
identifies the target statement in this command list where processing will continue.

When NetView interprets the &GOTO statement, it searches the command list for a statement starting with this same label. NetView transfers control to that statement and continues the command list processing. The statement identified by the label can be before or after the &GOTO statement.

You can code a variable for your label as long as the variable is replaced by a value before NetView processes the &GOTO statement. See "Labels" on page 75 for further information about labels.

&EXIT Control Statement

When the command list reaches the &EXIT control statement, the command list processing ends.

You can use &EXIT with &IF to check the command list and exit if there is an error. You can use &EXIT with &GOTO to control the flow of the command list. Figure 80 shows the syntax of the &EXIT control statement.

&EXIT *number*

Figure 80. &EXIT Control Statement Syntax

number
is an error return code. It can be equal to -1, 0, or any positive number up to 2147483647. To debug potential problems in nested command lists, code a return code on &EXIT.

The return code you set on the &EXIT control statement is placed in the &RETCODE control variable. The calling command list can test &RETCODE and take action based on the return code. See "Command List Information" on page 83 for more information about &RETCODE.

You can define your own meanings for the positive numbers. If you code a nonzero return code on the &EXIT statement, and if &CONTROL ERR is in effect, the command list command that generated the nonzero return code is echoed on the screen.

When a command list returns a -1, that command list, and all command lists in the nested chain, end. If you do not code a return code on &EXIT, or if the command list ends when the last line is processed and there is no &EXIT statement, a zero return code is set.

Figure 81 is an example command list named STOPTAF that uses the ENDSESS command to stop all terminal access facility sessions. The command list checks for errors. To start the command list enter STOPTAF or STOPTAF ALL. If you forget what the command list does or forget what to enter, you can use STOPTAF ? to get help.

```
STOPTAF CLIST
&CONTROL ERR
*       IF USER ENTERS STOPTAF ?, GO TO HELP SECTION
&IF .&1 EQ .? &THEN &GOTO -HELP
*       IF NO PARAMETERS, GO TO COMMAND
&IF .&1 EQ . &THEN &GOTO -CMD
*       IF PARAMETER IS ALL, GO TO COMMAND.  ELSE PRINT ERROR MSG
&IF .&1 NE .ALL &THEN &GOTO -ERROR
-CMD
ENDSESS OPCTL,ALL
ENDSESS FLSCN,ALL
&EXIT
-ERROR
&WRITE YOU ENTERED:  STOPTAF &PARMSTR WHICH IS NOT CORRECT
-HELP
&BEGWRITE -END
ENTER:  STOPTAF TO STOP ALL TERMINAL ACCESS FACILITY SESSIONS
-END
&EXIT 4
```

Figure 81. STOPTAF Command List Using &IF, &GOTO, and &EXIT

If you enter STOPTAF or STOPTAF ALL, only the results of the two ENDSESS commands are displayed.

If you enter STOPTAF FLSCN, the following message is displayed:

```
YOU ENTERED:  STOPTAF FLSCN WHICH IS NOT CORRECT
ENTER:  STOPTAF TO STOP ALL TERMINAL ACCESS FACILITY SESSIONS
```

If you enter STOPTAF ?, the following message is displayed:

```
ENTER:  STOPTAF TO STOP ALL TERMINAL ACCESS FACILITY SESSIONS
```

&WAIT Control Statement

Sometimes you might want a command list to wait for a specific event or message. You then define what event will cause the command list to resume processing with the &WAIT control statement. The command list can wait for any message with a 1- to 10-character message identifier.

Note: &WAIT cannot be used when operating under the PPT task, or when using service point service (SPSC) commands. See "Primary POI Task Restrictions" on page 17 for more information using &WAIT under the PPT task. For additional information about using &WAIT with SPSC commands, refer to Chapter 10, "Service Point Command Service Commands" on page 165.

If you use &WAIT in an automation task command list, be sure to specify a reasonable time-out value. For instructions on how to code a time-out event, see "The Event=-Label Pair" on page 113.

&WAIT does the following in a command list:

- It causes NetView to monitor the operator station task (OST) for specific messages and takes action if the message arrives. For example, the command list issues a VTAM command to activate a resource. When VTAM sends the message saying the resource is active, the &WAIT initiates a specific action based on the successful activation of the resource.
- It initiates a specific action if a message does not arrive in a specified period of time. For example, for your installation, you want to display resources if the activation message does not arrive within 5 minutes.

Therefore, you can use &WAIT in the following applications:

- The command list starts a session with an application program, such as IMS/VS, or another NetView domain. The &WAIT causes NetView to monitor the operator station task for messages indicating the session is started; this satisfies the &WAIT condition. When the &WAIT condition is fulfilled, the command list resumes processing and sends the logon and other information.
- The command list issues requests for status information from VTAM, and then processes or reformats this information before sending it to the NetView operator.

&WAIT and &PAUSE work differently. With &PAUSE, the command list does not continue until the operator enters the GO command. Operands on the GO command are used in the command list. However, because &WAIT causes the command list to wait for a specific event or events, GO is only used to resume the command list if the event never occurs. When a command list is in a wait state, NetView ignores operands on the GO command. RESET, STACK, and UNSTACK work the same way for &WAIT and &PAUSE.

Coding an &WAIT Control Statement

There are several ways to code an &WAIT statement. This section discusses the basic format. "Customizing the &WAIT Statement" on page 118 discusses ways to customize &WAIT.

When the command list gets to an &WAIT control statement, NetView displays the letter W on the upper right-hand corner of the terminal screen if the screen is refreshed as the result of a message being received or the ENTER key being pressed. This notifies the operator that a command list process is in a wait state. Wait state means the command list has halted its processing and is waiting for a specific message or group of messages. When the specific message arrives, the control variables and the parameter variables are set to their current values. Figure 82 shows the syntax of the &WAIT control statement.

```
&WAIT   ['command']
        event=-label [,...]
```

Figure 82. &WAIT Control Statement Syntax

'command'
: is any command or command list that you can issue from NetView. This command is optional. It is usually the command from which the command list is waiting for messages. For example, if you want the command list to wait for a successful session startup, the entire BGNSESS command is coded between single quotes. Be sure to code command list continuation characters before the event=-label pairs. The command is run as soon as it is reached in the command list.

  Note: The W signifying a wait state, if present, remains in the upper right corner of the screen while this initial &WAIT command is processed. The W tells the operator that NetView is still waiting for messages. If the operator enters GO before this command or command list completes processing, the GO is rejected with message DSI016I NOT IN PAUSE OR WAIT STATUS. When the command or command list is complete, the GO is accepted. RESET ends a command list that is in a wait state. If you enter the STACK command, the W, if present, does not remain in the upper right-hand corner of the screen.

  You can code one of the NetView timer commands, AT, EVERY, or AFTER, in the &WAIT statement. If the scheduled command is a command list, it cannot run until either the current command list is complete or the STACK command is entered.

event=-label
: is an event=-label pair. You can code as many of these pairs as you want on an &WAIT statement, up to the limit of 255 characters. The event is usually a message for which the command list is waiting. The event can be a trigger that ends the wait state before the message arrives. The &WAIT statement causes NetView to scan all messages sent to the operator. If a message matches one of the events coded, the command list goes to the line with the specified label and continues processing from the labeled statement. For more information on the types of events that can satisfy an &WAIT, see "The Event=-Label Pair" on page 113.

Note: While you can code several event=-label pairs, the first message, or other condition, that matches one of the events stops the command list from waiting for more messages. You can change this if you want to process several messages with one &WAIT statement. See "Customizing the &WAIT Statement" on page 118.

When NetView receives the message it is waiting for, the message is displayed on the operator terminal, as are all NetView messages. However, in this case, the message type is W unless the message satisfying the &WAIT originated from a command list, in which case the message type remains C. If you do not want the operator to see this message see "Customizing the &WAIT Statement" on page 118.

NetView only checks messages that are intended for the operator screen. If you have coded exit routine DSIEX02A (output to the operator), the &WAIT control statement may not be able to match the message. For instance, if DSIEX02A deletes the message, &WAIT does not match it. Since the operator does not receive the message, neither does the waiting command list. Therefore, you should only wait for messages that are displayed on the NetView console.

The Event=-Label Pair

The *event=-label* pair on the &WAIT statement lets you pass control to a statement with a label when one of four types of events occurs. The label is a standard label as described in "Labels" on page 75. The label coded on the &WAIT statement can be a variable, but parameter variables should not be used.

You can pass control to the label on an &WAIT statement by specifying an *event=-label* pair. The events you can use are:

- *token*
- *ERROR
- **nn*
- *ENDWAIT.

token

The event occurs when NetView receives a message matching *token*. *token* is 1 to 10 characters that identify the first token of the message or messages for which the command list is waiting. Optionally, you can identify the domain of a message for which the command list is waiting. If a domain identifier is specified, it precedes the token and is separated from the token by a period (*domainid.token*). You can also use an asterisk (*) to indicate you are specifying a partial domain identifier or token. If you do not specify a domain identifier, the message for which the command list is waiting can be from any domain.

Following are examples of some of the ways you can specify the messages for which you want the command list to wait:

domainid.token	The event occurs when NetView receives any message whose domain identifier matches the 1- to 5-character *domainid* and whose first token matches *token*.
dom.token*	The event occurs when NetView receives any message whose domain identifier matches the partial domain identifier specified by *dom** and whose first token matches *token*. For example, NCCF*.DSI463I means the event occurs when a DSI463I message is received from any domain with an identifier that starts with NCCF (such as NCCFA or NCCFB).

**.token*	The event occurs when NetView receives any message whose first token matches *token*. The message can be from any domain.
token	The event occurs when NetView receives any message whose first token matches *token*. The message can be from any domain.
*tok**	The event occurs when NetView receives any message whose first token matches the partial token specified by *tok**. For example, DSI* means the event occurs when NetView receives any message whose first token begins with DSI (such as DSI463I or DSI386I).
*	The event occurs when NetView receives any message.

Multi-line messages such as multi-line write-to-operator (MLWTO) are treated as one message. Therefore, only the message identifier of the first message in a multi-line message is available to the &WAIT, and the &WAIT can be satisfied only by that message identifier. Use GETMSIZE, GETMTYPE, and GETMLINE to access the other messages in a multi-line message. See "Working with Multi-Line Messages" on page 151, for more information on multi-line messages and an example of using &WAIT with multi-line messages.

Note: When using a *token* event, messages not related to the command issued by the &WAIT can be matched to the event and, depending on the options on the &WAIT statement, can be suppressed. If the command list is suspended and the SUPPRESS option is in effect on the &WAIT statement, any messages received by the task are suppressed before the command list is resumed.

***ERROR**

This event occurs when the command specified on the &WAIT statement returns a nonzero return code. If you do not code *ERROR, NetView continues to wait for the messages associated with this command even if the command ends with an error. If NetView is waiting for a message that says the command was successful, the operators running this command list will be delayed until someone issues GO or RESET. If *ERROR is satisfied, the message control variables are set as follows:

Control Variable	Value
&MSGID	*ERROR
&MSGORIGIN	null
&MSGSTR	null
&MSGCNT	0

NetView issues the messages, so do not issue &MSGID (*ERROR) or &MSGSTR (NULL) at the designated label.

Note: Messages associated with the command can be received before the command returns a non-zero return code. If such a message is coded on an *event*=-*label* pair, control is passed to the first statement whose event has occurred. For instance, if you code the name of the &WAIT command on a MSGID=-*label* pair, and you also code an *ERROR=-*label* pair, NetView honors the MSGID=-*label* pair first because that event occurs first.

**nn*

This event occurs after *nn* seconds. If no other event occurs, the &WAIT ends and control passes to the labeled statement. You can code a value between 1 and 32767 seconds (9 hours, 6 minutes, 7 seconds). If you do not code **nn* and none of the events of the &WAIT are satisfied, &WAIT continues until the operator enters a GO or RESET command.

If a nested command list contains an &WAIT statement with **nn* event, the **nn* of the original command list is ignored.

***ENDWAIT**

This event occurs when the operator or a command list issues a GO command. If you do not code *ENDWAIT=*-label*, the GO command continues processing with the statement following the &WAIT command.

Error Conditions

If an error condition occurs, NetView should be able to go to another part of the command list and take appropriate action. Consider the types of errors you can have and plan to handle them by coding *ERROR, **nn*, and *ENDWAIT events.

Coding Message = -Label Pairs

The order in which you code MSGID=*-label* pairs is important. NetView scans the pairs in the order you code them, from left to right.

For example, assume you code the statement in Figure 83.

```
&WAIT IST*=-ALL,IST123I=-SPECIAL
```

Figure 83. Example &WAIT Command Using MSGID=-Label Pairs

When NetView receives IST123I, it goes to the label -ALL, not -SPECIAL. You should code IST123I before IST*.

You can code as many events as required on one &WAIT control statement up to 255 characters. Remember to use continuation characters if the event pairs take up more than one line. Code the message and domain identifiers in the order that you want them processed. NetView scans the list from left to right until a match is found.

Ending an &WAIT

An &WAIT can be ended in one of the following ways:

- By the operator entering the GO command. Processing continues with the next statement unless *ENDWAIT is specified on the &WAIT statement. If *ENDWAIT is specified on the &WAIT statement, processing continues with the statement marked by the label.
- By the operator entering the RESET command. The command list (and all of its nested command lists) ends.
- By coding *ERROR on the &WAIT statement. If the command specified on the &WAIT statement ends with an error, the command list continues processing at the statement marked with the label. If you do not code *ERROR in this situation, the &WAIT does not end until the operator enters GO or RESET.

- By coding **nn* on the &WAIT statement. The command list continues processing at the statement specified by the label if another event does not occur within *nn* seconds.
- Upon receipt of a message matching an *event=-label* pair. The command list continues processing with the statement marked with the label.

Using NetView Commands with &WAIT

When a command list written in the NetView command list language is in a pause or wait state, operator commands that are entered can be deferred. Whether the commands are deferred is based on the NetView DEFAULTS, OVERRIDE, and CMD commands. See *NetView Operation* for information on these commands.

The GO, STACK, UNSTACK, and RESET commands affect the processing of command lists in a wait state as follows:

- GO ends the wait. If *ENDWAIT is coded, processing continues with the labeled statement.
- STACK suspends command list processing and causes any commands that have been deferred to be processed. You can enter any command or command list for normal processing while a command list is suspended. The &WAIT is not suspended, and events are still matched as they occur. The W, if present, does not remain in the upper right corner of the NetView screen. The GO command is rejected until the command list resumes processing.
- UNSTACK resumes command list processing. The &WAIT resumes processing events that were matched while the command list was suspended. The &WAIT does not resume after expiration of a specified time if, while the command list was suspended, you ran another command list that used &WAIT or WAIT with a time specified.
- RESET ends a command list that is in a wait state, as well as all command lists related to it by nesting.

For more information on the GO, STACK, UNSTACK, and RESET commands, see *NetView Operation.*

When processing MLWTO messages received in response to an &WAIT control statement, use the GETMLINE, GETMSIZE, and GETMTYPE commands. For more information about these commands, see “Working with Multi-Line Messages” on page 151.

Control and Parameter Variables Used with &WAIT

NetView sets the values of the &MSGCNT, &MSGID, &MSGORIGIN, &MSGSTR, and &MSGTYP control variables and the &1 - &31 parameter variables based on the receipt of a message coded on an &WAIT control statement.

&MSGCNT
becomes the number of elements of the text of &MSGSTR.

&MSGID
becomes the message identifier of the message received. The message identifier is the first token of the message (up to 10 characters). If the first token is longer than 10 characters, &MSGID uses only the first 10 characters.

&MSGORIGIN
becomes the name of the domain where the message originated.

&MSGSTR
becomes the message text exactly as it is received by NetView. &MSGSTR does not include the message identifier (the token used by the &MSGID control variable).

&MSGTYP
becomes the system message type of the message received.

&1 - &31
NetView changes the values of the &1 - &31 parameter variables to reflect the text of the message. Each parameter variable is set to a token of the message. Tokens are delimited by commas, apostrophes, or blanks. &1 is set to the first token following the message identifier (the token used by the &MSGID control variable). &2 is set to the next token to the right of &1, and so on up to a maximum of 31 variables.

Following is an example of how the variables are set when the following message from domain DOM01 is intercepted by an &WAIT:

```
DSI008I SPAN1  NOT ACTIVE
```

Variable	Value
&MSGORIGIN	DOM01
&MSGID	DSI008I
&MSGSTR	SPAN1 NOT ACTIVE
&MSGCNT	3
&1	SPAN1
&2	NOT
&3	ACTIVE
&4 - &31	NULL

Notes:

1. If NetView receives a multi-line message, the control variables and parameter variables are set according to the first line of the message. See "Working with Multi-Line Messages" on page 151 for information concerning working with multi-line messages.
2. If &1 - &31 are given values when the command list runs, save the parameter variables in user variables before invoking the &WAIT control statement. This lets you use the original values after &WAIT changes them.
3. After issuing an &WAIT control statement, save the values of the control variables in user variables before issuing another &WAIT control statement. This lets you use the values after another &WAIT changes them.
4. If you are using &WAIT CONTWAIT, be careful when using the control variable &MSGID before the &WAIT has ended. If an &WRITE or &BEGWRITE is used to display &WAIT as the first character in the text, the output can be suppressed or cause the command list to loop. If the &WAIT SUPPRESS option is in effect, an &WRITE or &BEGWRITE, with &MSGID as the first character string of the text, matches the MSGID=-*label* operand of the active &WAIT. Therefore, the text of the &WRITE or &BEGWRITE is not sent to the operator's screen. If an &WAIT CONTINUE statement is encountered after a MSGID=-*label* is matched, and there is no other statement to end the command list or the &WAIT, the command list will loop.

Using &WAIT in Nested Command Lists

The command in the &WAIT statement can be a command list. The nested command list can contain an &WAIT statement too. You should be aware of the following considerations when using &WAIT with nested command lists:

- Messages that arrive for the waiting command lists are queued until the nested command list is finished processing.
- If you specify the same message number on &WAIT statements in both the waiting and nested command lists, the message satisfies the &WAIT in the nested command list.
- If you specify timer events using *nn* on &WAIT statements in both the waiting and nested command lists, the timer event of the waiting command list is canceled.

Customizing the &WAIT Statement

The previous sections described the simplest form of the &WAIT command, where the first message received that satisfies the wait is displayed on the operator's terminal and causes the command list to continue processing.

This section describes how to customize the &WAIT statement for even more flexibility.

To customize your &WAIT statements use the following syntax.

```
&WAIT    [DISPLAY|SUPPRESS]
         [ENDWAIT|CONTWAIT ]
         [CONTINUE]
```

Figure 84. Syntax for Customizing an &WAIT Statement

DISPLAY|SUPPRESS
determines whether a message that matches a wait event is displayed at the operator's terminal. The DISPLAY and SUPPRESS options can be changed at any point in a command list. Once messages have been suppressed, you must code another &WAIT statement with the DISPLAY operand to begin displaying messages again.

DISPLAY indicates that the message the command list is waiting for is to be displayed at the operator's terminal upon arrival to NetView. This is the default value.

SUPPRESS indicates that any messages that have satisfied an &WAIT are not displayed.

Note: When SUPPRESS is in effect, you do not know whether messages have been received. Therefore, it is possible that all of the messages will not be processed when an operator issues a GO or RESET command to end an &WAIT.

ENDWAIT|CONTWAIT
indicates whether the command list should continue to wait for additional events or should end the wait after the first event that satisfies the &WAIT. The ENDWAIT and CONTWAIT options can be changed at any point in a command list. Once CONTWAIT has started, you must code another &WAIT statement with the ENDWAIT operand to return to the default value.

ENDWAIT sets up processing for the next *event=-label* pair to be processed. This is the default value. ENDWAIT indicates that the current, or the next, *event=-label* pair ends after the first event that satisfies the &WAIT. Although ENDWAIT does not end a wait already in process, operators can still use the GO command to end the wait. The RESET command, which ends a wait, also ends the command list.

CONTWAIT indicates that the next &WAIT *event=-label* statement encountered waits for additional events until the wait is ended. This enables one &WAIT statement to process more than one event. This is useful when you want to retrieve more than one message from a single command, such as a LIST command.

CONTINUE
directs the command list to continue waiting for the next event that satisfies the original &WAIT statement. CONTINUE is used only when &WAIT CONTWAIT is specified prior to the &WAIT *event=-label*. If you want the wait to continue after event processing is finished, code &WAIT CONTINUE. This directs the command list to continue waiting for the next event that satisfies the original &WAIT statement.

The operands of this format are optional and can be coded in any order. However, they cannot be coded on the &WAIT *event=-label* statement. The &WAIT statement does not put the command list into a wait state. Instead, it indicates how the command list processes the next &WAIT *event=-label* control statement.

If you update this statement using SUPPRESS, CONTWAIT, or CONTINUE, the new settings remain in effect for the rest of the &WAIT statements in the command list, including an &WAIT currently in process. To reinstate the initial settings, you must code another &WAIT statement with the appropriate operands. If you activate a nested command list, the default settings are in effect for that command list unless an &WAIT statement is coded for the nested command list.

Ending &WAIT if CONTWAIT is in Effect

"Ending an &WAIT" on page 115 described ways to end a wait when a command list is waiting for only one event. When the command list is waiting to match more than one event, you can end the wait in one of the following ways:

- By entering the GO command at the terminal.

  If an &WAIT CONTINUE was the last &WAIT statement encountered, processing continues with the next command list statement following the &WAIT CONTINUE statement. If the *ENDWAIT event is coded, processing continues at the label statement. If no *event=-label* match has occurred, processing continues with the line following the &WAIT statement.

- By coding the GO command in the command list statement that follows an &WAIT ENDWAIT statement.

  If the *ENDWAIT event is coded, processing continues at the label statement. If no *event=-label* match occurred, processing continues with the line following the GO command.

- By coding *ERROR as the event on the &WAIT statement.

  If the command specified on the &WAIT statement ends with an error, the command list continues processing at the statement specified with a label.

The &WAIT does not end unless an error occurs. However, if there is an error in the command list and you do not have *ERROR coded, the wait may never end.

- By coding **nn* on the &WAIT statement.

  The command list continues processing at the statement specified with a label if the event does not occur within *nn* seconds.

- By coding *ENDWAIT on the &WAIT statement.

  The command list continues processing at the statement specified with the label when the operator enters the GO command.

- By coding &EXIT following a label.

  This causes the command list to end normally.

- By entering the RESET command.

  The command list, including the command list that initiated it, ends.

Note: Because &WAIT CONTWAIT queues NetView messages, you should also code &WAIT CONTINUE to receive these queued messages. If you code &WAIT CONTWAIT with SUPPRESS and end the wait, you could lose some messages.

Suggestions for Coding &WAIT

It is best to use the &WAIT [ENDWAIT|CONTWAIT] options in the following way:

1. Set up options for the &WAIT *event=-label* statement by coding &WAIT with CONTWAIT, SUPPRESS, or their defaults.
2. Enter an &WAIT state by using an &WAIT *event=-label* statement.
   - If &WAIT ENDWAIT is specified before the &WAIT *event=-label* statement, or is in effect by default, the first matched event ends the wait, and command list processing continues. See "Ending an &WAIT" on page 115.
   - If &WAIT CONTWAIT is specified, the receipt of the first event does not end the &WAIT unless this event is specified in "Ending &WAIT if CONTWAIT is in Effect" on page 119. The command list goes to the label specified for the event and continues processing.

     To complete this section of the command list, do one of the following:

     – Continue the wait by coding &WAIT CONTINUE.

     – Specify that the next event is the last of this wait by coding &WAIT ENDWAIT and then &WAIT CONTINUE.

     – End the wait by coding the &WAIT ENDWAIT statement and GO command in the command list.

     – End the command list by coding &EXIT.
3. Continue the command list according to the results of step 2.

Sample Using &WAIT

This section contains an example of the &WAIT statement in a command list.

Figure 85 on page 121 is an example illustrating the use of &WAIT to wait for one message. The command list is named ACTONE, and it issues a VTAM command to activate a logical unit. The command list traps the messages responding to the activate command, then reformats the messages and writes them to the operator's screen. This command list is activated by entering ACTONE NODE1.

```
&CONTROL ERR
* ACTONE COMMAND LIST
* THIS COMMAND LIST ISSUES A VTAM "V NET,ACT" COMMAND, TRAPS ITS
*   MESSAGES AND REFORMATS THEM.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
* IF THERE IS NO INPUT PARAMETER, ASK FOR ONE
&IF .&1 = . &THEN &GOTO -BADIN
* SAVE THE INPUT PARAMETER
&LU = &1
* END THE WAIT WITH THE FIRST MESSAGE AND DO NOT DISPLAY THE
*   INPUT MESSAGE ON THE SCREEN
&WAIT ENDWAIT SUPPRESS
* ISSUE WAIT WITH THE COMMAND
&WAIT 'V NET,ACT,ID=&LU',IST093I=-REFORM,*ERROR=-FAIL,+
   IST105I=-FAIL,*ENDWAIT=-GOIN
-REFORM
* REFORMAT MESSAGE IST093I (SUCCESSFUL) AND WRITE TO THE SCREEN
* &1 IN THE FOLLOWING LINE IS NOT THE ORIGINAL &1
&ACTIV = &1
&WRITE VTAM MESSAGE IST093I WAS RECEIVED
&WRITE &ACTIV IS NOW ACTIVE
&GOTO -ENDALL
-FAIL
* REFORMAT MESSAGE IST105I (UNSUCCESSFUL) AND WRITE TO THE SCREEN
&WRITE &LU COULD NOT BE ACTIVATED
&GOTO -ENDALL
-GOIN
* IF "GO" ISSUED, INDICATE THAT MESSAGES HAVE NOT BEEN RECEIVED
&WRITE "GO" INPUT COMMAND LIST ACTONE -- &LU IS NOT ACTIVE NOW
&GOTO -ENDALL
-BADIN
&WRITE RE-CALL COMMAND LIST ACTONE WITH PARAMETER OF LU TO BE ACTIVATED
-ENDALL
&WRITE COMMAND LIST ACTONE COMPLETE
&EXIT
```

Figure 85. Command List Issuing &WAIT for One Message

The ACTONE command list waits for one of the following messages:

```
IST093I   modename ACTIVE
IST105I   modename NODE NOW INACTIVE
```

The command list is activated by entering ACTONE and operand NODE1. The operand is the name of the logical unit to be activated. This operand supplies the value for parameter variable &1. Receipt of a message indicating success (IST093I) or failure (IST105I) caused the wait to end because ENDWAIT was specified. Processing continues at the specified label (-REFORM for IST093I, -FAIL for IST105I). The awaited messages are not displayed because SUPPRESS was specified, but any other messages are displayed.

Upon successful activation of NODE1, the following message text is displayed on the operator's terminal:

```
ACTONE NODE1
IST097I  VARY      ACCEPTED
VTAM MESSAGE IST093I WAS RECEIVED
NODE1 IS NOW ACTIVE
COMMAND LIST ACTONE COMPLETE
```

Chapter 8. NetView Command List Language Global Variables

This chapter describes the syntax and use of global variables that are used in command lists written in the NetView command list language. You will learn how to assign values to global variables used in command lists running under the same task and how to assign values to global variables that can be passed between command lists that are running under different tasks.

Global variables allow values to be defined, referenced, and updated by different operators. Values are passed to a command list for updates, and the updated values are passed back to the first command list. For example, command list CLISTA can assign a value to a task global variable, &VAR1, and then activate its nested command list, CLISTB. The nested command list, CLISTB, can check the value assigned to &VAR1 by CLISTA, update the value, and return control to CLISTA. The original command list, CLISTA, now has access to the value assigned to &VAR1 by CLISTB.

There are two types of global variables; task and common. Task global variables let you define, reference, and update any number of global variables to a particular task. The common global variable allows definition of user variables that can be referenced by command lists running under any task, as opposed to task global variables, which can only be referenced by a single task.

When you create global variables, follow these rules:

- The global variable can be 1 to 11 characters. A-Z, 0-9, #, @, and $ are valid characters.

  Note: If you want global variables you create in a REXX command list to also be accessible to command lists written in the NetView command list language, make sure the global variable names are from 1 to 11 characters in length and do not contain a period, _, ¢, !, or ?.

- If more than one global variable is specified on the GLOBALV.

- On the definition statement, an ampersand (&) should not be coded with the global variable name except where you want variable substitution performed. Substitution occurs for any variable with an ampersand. Whenever you use the global variables (except when defining them), you must append an ampersand to the variable name, just as you would for user variables.

- You need two ampersands when referencing a global variable indirectly. See "How Parameter Variables Are Used in the Command List" on page 79 and "Variable Substitution Order" on page 76 for more information on indirect referencing of variables.

- The value of the global variable may be 255 characters long. For Kanji the maximum number of double-byte characters between the shift-out and shift-in is 126.

- You can give global variables a numerical value between -2147483647 and 2147483647.

Task Global Variables

A task global variable can only be referenced by command lists that run under the same task.

Use the following control statement as a model to define any variable as a task global variable. Figure 86 shows the syntax of the &TGLOBAL control statement.

```
&TGLOBAL variable [,...]
```

Figure 86. &TGLOBAL Control Statement Syntax

This statement defines the listed variables as task global variables. This means that, from this statement in the command list, &*variable*1 refers to a task global variable. The value of any variable defined by this statement is whatever was most recently assigned to it by another command list running under the same task. If no value was defined, the value is null. If the &TGLOBAL statement is not used in each command list before a variable is referenced, that variable defaults to a local user variable.

Here is an example using the &TGLOBAL control statement.

```
&NAME = JOHN
&TGLOBAL ABC,&NAME
```

Figure 87. Example &TGLOBAL Control Statement

The first line consists of a local user variable set to the value JOHN. The second line defines two task global variables as follows:

- ABC becomes task global variable &ABC. The value of &ABC is null because a value was not defined.
- &NAME becomes task global variable &JOHN. The value of &JOHN is null because a value has not been defined.

See "Scope of Variables in Command Lists" on page 127 for information on the interaction of task global variables with user variables and common global variables.

If more than one variable name is specified on the &TGLOBAL statement, the variable names must be delimited by commas or blanks.

The following are suggestions for using task global variables:

- The PROFILE IC can set task global variables to indicate a message suppression level or message compression that is different for different types of operators. Command lists driven by various messages can test these variables to determine what information a particular operator needs.
- Any command list can set up and initialize any number of parameters for another command list running under the same operator task. This provides improved nested command list communication because task global variables can return information from a nested command list.

- Task global variables can maintain accurate information about the network regardless of operators logging on and off. Task global variables can keep cumulative information from unsolicited access method messages. For example, notification of a failing resource can be used to recover the resource. With a global variable, a count of the number of retries can be maintained to prevent a loop.

Updating Task Global Variables

Figure 88 and Figure 89 are examples of command lists. The first command list is named CLIST1, and it contains the nested command list UPDATE1. These command lists show how to define, reference, and update a task global variable.

```
* THIS STATEMENT DEFINES TOM AS A TASK GLOBAL VARIABLE.
  &TGLOBAL TOM
* THIS ASSIGNMENT STATEMENT GIVES THE TASK GLOBAL
*   VARIABLE, "TOM", A VALUE OF 5.
  &TOM = 5
* THIS STATEMENT CALLS A NESTED COMMAND LIST NAMED UPDATE1.
*   TOM IS A PARAMETER THAT IS PASSED TO COMMAND LIST UPDATE1.
  UPDATE1 TOM
* THIS STATEMENT WILL WRITE VALUE OF TOM.
  &WRITE TOM = &TOM
  &EXIT
```

Figure 88. CLIST1 Command List to Define, Update, and Reference Task Global Variables

```
* THIS STATEMENT DEFINES &1 AS A TASK GLOBAL VARIABLE.
*   &1 IS SET TO THE VALUE OF THE POSITIONAL PARAMETER
*   TOM, WHICH ON THE FIRST PASS IN THIS CASE IS 5.
  &TGLOBAL &1
* THIS STATEMENT TESTS FOR A NULL VALUE AND INITIALIZES
*   THE TASK GLOBAL VARIABLE PASSED AS &1 TO A VALUE OF
*   0 IF THE VALUE WAS NULL.
*   THE TASK GLOBAL VARIABLE PASSED AS &1 IS REFERENCED
*   AS &&1.  THE VALUE OF &&1 IS EQUAL TO THE VALUE OF TOM,
*   WHICH WAS PASSED TO COMMAND LIST UPDATE FROM CLIST1.
  &IF .&&1 EQ . &THEN &&1 = 0
* THIS STATEMENT UPDATES THE TASK GLOBAL VARIABLE, &&1,
*   BY AN INCREMENT OF 1.
*   THIS UPDATED VALUE OF &&1 PASSED BACK TO CLIST1
*   AS TASK VARIABLE &TOM.
  &&1 = &&1 + 1
  &EXIT
```

Figure 89. UPDATE1 Command List to Update Task Global Variables

CLIST1 defines a task global variable, TOM. The value of the task global variable TOM is null until a value is assigned using the assignment statement, &TOM = 5. CLIST1 activates a nested command list named UPDATE1.

UPDATE1 defines a task global variable, &1. Task global variable &1 receives the value passed from CLIST1 through the positional parameter TOM. The NetView program scans variables from right to left, so the *&1* part of &&1 is evaluated first, and the value of &1 is equal to the value of TOM. The value of task global variable &1 is referenced as &&1. The initial value of &&1 is 5, and then &&1 is incremented by 1 using the &&1 = &&1 + 1 statement.

The updated value is returned to task global variable &TOM in CLIST1. The &WRITE TOM = &TOM statement displays the updated value of the &TOM task global variable.

Common Global Variables

Use the &CGLOBAL control statement to define any variable as a common global variable. Figure 90 shows the syntax of the &CGLOBAL control statement.

```
&CGLOBAL variable [,...]
```

Figure 90. &CGLOBAL Control Statement Syntax

This statement defines the listed variables as common global variables. The value of any variable defined by this statement is whatever was most recently assigned to it by any other command list. If no value has been defined, the value is null. If the &CGLOBAL statement is not used in each command list before a variable is referenced, that variable defaults to a local user variable.

Following is an example using the &CGLOBAL control statement.

```
&NAME = JOHN
&CGLOBAL ABC,&NAME
```

Figure 91. Example &CGLOBAL Control Statement

The first line consists of a local user variable set to the value JOHN. The second line defines two common global variables as follows:

- ABC becomes common global variable &ABC. The value of &ABC is null because a value has not been defined.
- &NAME becomes common global variable &JOHN. The value of &JOHN is null because a value has not been defined.

Note: Be careful if you have more than one command list running under different tasks and accessing the same global variable. The last value that the variable is set to is the value that is retrieved by any command list accessing the variable. For example, a command list accesses a common global variable and then before that command list updates the variable, another command list running under a different task accesses the variable. If both command lists update the variable, the variable assumes the value given to it by the command list that updates it last. To avoid having a common global variable being used by different command lists at the same time, you can have all command lists that use the variable run under the same task.

See "Scope of Variables in Command Lists" on page 127 for information on the interaction of common global variables with user variables and task global variables.

If more than one variable name is specified on the &CGLOBAL statement, the variable names must be delimited by commas or blanks.

You can use the NetView-supplied command lists UPDCGLOB and SETCGLOB to update and set common global variables under the PPT. See *NetView Operation* for information on using UPDCGLOB and SETCGLOB.

Scope of Variables in Command Lists

If a global variable is defined with the same name as a local variable, the value of the local variable is lost. The global variable does not receive the value of the local variable. The value of the global variable is null until a value is assigned.

If a common global variable is defined after the task global variable has been defined and has the same name as a task global variable, the value of the task global variable remains unchanged. However, the value of the task global variable can no longer be accessed by this command list unless the variable is redefined using &TGLOBAL.

If a task global variable is defined after the common global variable has been defined and has the same name as a common global variable, the value of the common global variable remains unchanged. The value of the common global variable, however, can no longer be accessed by this command list unless the variable is redefined using &CGLOBAL.

GLOBVAR1 and GLOBVAR2 (Figure 92 on page 128 and Figure 93 on page 130) illustrate the scope of user variables, task global variables, and common global variables within individual command lists and command lists running under different tasks. These two command lists give you examples of the following variable manipulations:

- Assigning values to user variables
- Defining task global variables
- Defining common global variables
- Setting values for common global variables
- Changing common global to task global variables.

In the examples, the values of the different variables are shown in parentheses. Notice how the values start out as nulls before values are assigned.

The examples assume that the command lists are run under different tasks and that GLOBVAR1 is run before GLOBVAR2. Because the command lists run under different tasks, they do not access the same task global variables. Because GLOBVAR1 runs before GLOBVAR2, GLOBVAR2 accesses the values that GLOBVAR1 sets for common global variables.

```
&CONTROL ERR
*** MEMBER NAME   GLOBVAR1
*** ASSIGN VALUES TO SEVERAL USER VARIABLES AND PRINT THEIR VALUES
*******************************************************************
     &NAME = JON
     &ADDR = BAHAMA
     &PROF = REALTOR
     CLEAR
     &BEGWRITE SUB -ENDLOCAL
        FROM GLOBVAR1: AFTER LOCAL VARIABLES ASSIGNED
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       =========
                LOCAL          NAME           &NAME (JON)
                LOCAL          ADDR           &ADDR (BAHAMA)
                LOCAL          PROF           &PROF (REALTOR)
     -ENDLOCAL
***DEFINE TASK GLOBAL VARIABLES
******************************
     &TGLOBAL NAME ADDR CALLS
     &BEGWRITE SUB -ENDTG1
        FROM GLOBVAR1: AFTER TGLOBAL VARIABLES DEFINED
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          PROF           &PROF   (REALTOR)
                TASK           NAME           &NAME   (NULL)
                TASK           ADDR           &ADDR   (NULL)
                TASK           CALLS          &CALLS  (NULL)
..
            NOTE THAT THE VALUES ASSIGNED TO NAME AND ADDR
            HAVE BEEN LOST AS THESE ARE NO LONGER LOCAL
            VARIABLES AND THE TASK GLOBAL VARIABLES HAVE NOT
            BEEN ASSIGNED A VALUE YET.
     -ENDTG1
* ASSIGN VALUES TO THE TASK GLOBAL VARIABLES
********************************************
     &NAME  = DOUGLAS
     &ADDR  = CARY
     &CALLS = 5
     &BEGWRITE SUB -ENDTG2
        FROM GLOBVAR1:  AFTER VALUES ASSIGNED TO TGLOBAL VARIABLES
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          PROF           &PROF  (REALTOR)
                TASK           NAME           &NAME  (DOUGLAS)
                TASK           ADDR           &ADDR  (CARY)
                TASK           CALLS          &CALLS (5)
     -ENDTG2
```

Figure 92 (Part 1 of 3). GLOBVAR1 Example Showing Scope of Global Variables

```
***DEFINE COMMON GLOBAL VARIABLES
*********************************
     &CGLOBAL NAME ADDR NUMBER
     &BEGWRITE SUB -ENDTG3
        FROM GLOBVAR1: AFTER CGLOBAL VARIABLES DEFINED
              VARIABLE      VARIABLE      VARIABLE
                TYPE          NAME          VALUE
              ========      ========      ========
                LOCAL         PROF          &PROF    (REALTOR)
                TASK          CALLS         &CALLS   (5)
                COMMON        NAME          &NAME    (NULL)
                COMMON        ADDR          &ADDR    (NULL)
                COMMON        NUMBER        &NUMBER  (NULL)
..
             NOTE THAT THE VALUES ASSIGNED TO TASK GLOBAL
             VARIABLES NAME AND ADDR HAVE BEEN REPLACED BY
             COMMON GLOBAL VARIABLES NAME AND ADDR.  THESE
             ARE NULL AS NO VALUE HAS BEEN ASSIGNED TO THEM YET.
     -ENDTG3
***ASSIGN VALUES TO COMMON GLOBAL VARIABLES
*******************************************
     &NAME   = WILLIAM
     &ADDR   = PHOENIX
     &NUMBER = 10
     &BEGWRITE SUB -ENDTG4
        FROM GLOBVAR1: AFTER CGLOBAL VARIABLES ASSIGNED
              VARIABLE      VARIABLE      VARIABLE
                TYPE          NAME          VALUE
              ========      ========      ========
                LOCAL         PROF          &PROF    (REALTOR)
                TASK          CALLS         &CALLS   (5)
                COMMON        NAME          &NAME    (WILLIAM)
                COMMON        ADDR          &ADDR    (PHOENIX)
                COMMON        NUMBER        &NUMBER  (10)
     -ENDTG4
*
* CHANGE ONE COMMON GLOBAL VARIABLE BACK TO A TASK GLOBAL VARIABLE
******************************************************************

     &TGLOBAL NAME
     &BEGWRITE SUB -ENDTG5
        FROM GLOBVAR1: AFTER FINAL TGLOBAL STATEMENT
              VARIABLE      VARIABLE      VARIABLE
                TYPE          NAME          VALUE
              ========      ========      ========
                LOCAL         PROF          &PROF    (REALTOR)
                TASK          NAME          &NAME    (DOUGLAS)
                TASK          CALLS         &CALLS   (5)
                COMMON        ADDR          &ADDR    (PHOENIX)
                COMMON        NUMBER        &NUMBER  (10)
```

Figure 92 (Part 2 of 3). GLOBVAR1 Example Showing Scope of Global Variables

```
          NOTE THAT NAME NOW HAS THE VALUE OF THE TASK
          GLOBAL VARIABLE NAME AGAIN AS THE MOST RECENT
          DECLARATION STATEMENT DEFINED IT AS TASK GLOBAL.
     -ENDTG5
     &WRITE END OF CLIST: GLOBVAR1
 EXIT
```

Figure 92 (Part 3 of 3). GLOBVAR1 Example Showing Scope of Global Variables

```
 &CONTROL ERR
 *** MEMBER NAME   GLOBVAR2
 *** ASSIGN VALUES TO SEVERAL USER VARIABLES AND PRINT THEIR VALUES
 *******************************************************************
     &NAME = RICHARD
     &ADDR = CLAYTON
     &PROF = BARBER
     CLEAR
     &BEGWRITE SUB -ENDLOCAL
        FROM GLOBVAR2: AFTER LOCAL VARIABLES ASSIGNED
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          NAME           &NAME (RICHARD)
                LOCAL          ADDR           &ADDR (CLAYTON)
                LOCAL          PROF           &PROF (BARBER)
     -ENDLOCAL
 ***DEFINE TASK GLOBAL VARIABLES
 *******************************
     &TGLOBAL NAME ADDR CALLS
     &WRITE ENTER 'GO' TO CONTINUE
     &PAUSE
     CLEAR
     &BEGWRITE SUB -ENDTG1
        FROM GLOBVAR2: AFTER TGLOBAL VARIABLES DEFINED
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          PROF           &PROF  (BARBER)
                TASK           NAME           &NAME  (NULL)
                TASK           ADDR           &ADDR  (NULL)
                TASK           CALLS          &CALLS (NULL)
          NOTE THAT THE VALUES ASSIGNED TO NAME AND ADDR
          HAVE BEEN LOST AS THESE ARE NO LONGER LOCAL
          VARIABLES AND THE TASK GLOBAL VARIABLES HAVE NOT
          BEEN ASSIGNED A VALUE YET.  ALSO, SINCE THIS
          COMMAND LIST RUNS UNDER A DIFFERENT TASK THAN
          GLOBVAR1, THE VALUES ASSIGNED TO THE TASK
          GLOBAL VARIABLES BY GLOBVAR1 ARE NOT REFERENCED
          HERE.
     -ENDTG1
```

Figure 93 (Part 1 of 3). GLOBVAR2 Example Showing Scope of Global Variables

```
* ASSIGN VALUES TO THE TASK GLOBAL VARIABLES
*********************************************
     &NAME  = DAVID
     &ADDR  = RALEIGH
     &CALLS = 1
     &WRITE ENTER 'GO' TO CONTINUE:
     &PAUSE
     CLEAR
     &BEGWRITE SUB -ENDTG2
        FROM GLOBVAR2:  AFTER VALUES ASSIGNED TO TGLOBAL VARIABLES
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          PROF           &PROF  (BARBER)
                TASK           NAME           &NAME  (DAVID)
                TASK           ADDR           &ADDR  (RALEIGH)
                TASK           CALLS          &CALLS (1)
     -ENDTG2
***DEFINE COMMON GLOBAL VARIABLES
*********************************
     &CGLOBAL NAME ADDR NUMBER
     &WRITE ENTER 'GO' TO CONTINUE
     &PAUSE
     CLEAR
     &BEGWRITE SUB -ENDTG3
        FROM GLOBVAR2: AFTER CGLOBAL VARIABLES DEFINED
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          PROF           &PROF   (BARBER)
                TASK           CALLS          &CALLS  (1)
                COMMON         NAME           &NAME   (WILLIAM)
                COMMON         ADDR           &ADDR   (PHOENIX)
                COMMON         NUMBER         &NUMBER (10)

            NOTE THAT THE VALUES ASSIGNED TO TASK GLOBAL
            VARIABLES NAME AND ADDR HAVE BEEN REPLACED BY
            THE VALUES ASSIGNED TO COMMON GLOBAL VARIABLES
            NAME AND ADDR.  IF GLOBVAR1 HAD NOT BEEN RUN
            FIRST, NAME AND ADDR WOULD BE NULL.
     -ENDTG3
```

Figure 93 (Part 2 of 3). GLOBVAR2 Example Showing Scope of Global Variables

```
*
* CHANGE ONE COMMON GLOBAL VARIABLE BACK TO A TASK GLOBAL
VARIABLE
*******************************************************************

     &TGLOBAL NAME
     &WRITE ENTER 'GO' TO CONTINUE
     &PAUSE
     CLEAR
     &BEGWRITE SUB -ENDTG4
        FROM GLOBVAR2: AFTER FINAL TGLOBAL STATEMENT
              VARIABLE       VARIABLE       VARIABLE
                TYPE           NAME           VALUE
              ========       ========       ========
                LOCAL          PROF          &PROF   (BARBER)
                TASK           NAME          &NAME   (DAVID)
                TASK           CALLS         &CALLS  (1)
                COMMON         ADDR          &ADDR   (PHOENIX)
                COMMON         NUMBER        &NUMBER (10)

            NOTE THAT NAME NOW HAS THE VALUE OF THE TASK
            GLOBAL VARIABLE NAME AGAIN AS THE MOST RECENT
            DECLARATION STATEMENT DEFINED IT AS TASK GLOBAL.
     -ENDTG4
     &WRITE END OF CLIST: GLOBVAR2
&EXIT
```

Figure 93 (Part 3 of 3). GLOBVAR2 Example Showing Scope of Global Variables

Part Four. Advanced Command List Topics

Chapter 9. Message Automation

This chapter provides the following information about message automation for the NetView program:

- A definition of message automation
- How message automation for NetView Release 3 differs from NCCF and NetView Release 1
- How to define command lists for message automation
- How to send messages from a message automation command list to the MVS operator console
- How to route messages from message driven command lists
- How to parse variables using the PARSEL2R command
- How to process multi-line messages
- How to use the SDOMAIN command with the QUIET option to return messages for automation processing
- How to migrate to NetView Release 3 message automation.

This chapter is intended to help the customer perform message automation using command lists. It primarily contains guidance in implementing message automation. Unless specifically stated otherwise, the information in this chapter must not be used for programming purposes. However, this chapter also provides general use programming interfaces, which are explicitly identified when they occur. These interfaces are provided to allow the customer to write programs that use the services of the NetView program.

What Is NetView Message Automation

NetView message automation is a process that allows you to automate system response to messages. This allows you to have command lists and commands that are issued automatically when specific messages occur during operations. To learn more about setting up message automation for your system, see *NetView Installation and Administration Guide*.

How NetView Release 3 Message Automation is Different

NetView Release 3 message automation differs from NCCF and NetView Release 1. However, NetView Release 3 and NetView Release 2 have the same message automation. The following sections describe the differences in NetView message automation between Release 3 and NCCF and Release 1.

A utility program, DSICNVRT, is provided with NetView Release 3 to help you in migrating NCCF or NetView Release 1 to Release 3 message automation. See "How to Set Up for Migration" on page 164 for information about migrating to NetView Release 3 message automation. More detailed information about migrating to NetView Release 3 is available in the *NetView Installation and Administration Guide*.

Note: You do not need to migrate your message automation from Release 2 to Release 3.

How NetView Release 3 Differs From NCCF

NCCF allows VTAM messages to be automated by treating the message as a command instead of a message. The message ID is checked to see if it is defined as a command verb. If it is, the rest of the message is used as the operands of the command. To provide a command list to handle a message under NCCF, you simply use the message ID as your command list name, and create a CMDMDL statement for the command list name in DSICMD.

NetView message automation in Release 3 allows the command list to be driven by the same message as it was under NCCF, by using an IF-THEN automation statement in the message automation member of DSIPARM. If you are using MVS/OCCF for message automation, read "Conversion Considerations for MVS/OCCF" on page 164.

How NetView Release 3 Differs from NetView Release 1

NetView Release 3 message automation differs from NetView Release 1 in the way message automation is defined and in the new AUTOTASK command, which allows you to set up an automation task. If you are using MVS/OCCF for message automation, read "Conversion Considerations for MVS/OCCF" on page 164.

NetView Release 3 Message Automation Definition

In NetView Release 1, you use a MSGCMD statement in the message automation member instead of an IF-THEN automation statement. The MSGCMD statement associated a message containing a given text string with the name of a command list. If NetView received a message defined by MSGCMD, it would run the command list with the entire message as its parameter string (ARG(1) or &PARMSTR). In NetView Release 3, you can use the IF-THEN automation statement to parse the message into variables for the command list to use.

NetView Release 3 Automation Task

NetView Release 3 also provides the AUTOTASK command for creating automation subtasks. These function like Operator Station Tasks (OSTs) except that they are not logged on to a terminal and do not depend on an active VTAM session. Therefore, messages can be processed under automation tasks to bring up VTAM, JES2, JES3, and other system components automatically. New commands have been added to support the automation task.

How to Define NetView Message-Driven Command Lists

To define a message-driven command list, code an IF-THEN automation statement in a message automation member of DSIPARM.

After you code the IF-THEN automation statement in the message automation member, issue the AUTOMSG command using the name of that specific message automation member. You can enter the AUTOMSG command at a terminal, from a command list, or in an initialization command list at system startup. For the syntax of the AUTOMSG command, see *NetView Operation*.

There is no need to pass the message text as a parameter string to your command list, as was done in NCCF and NetView Release 1. Important variable information in the text of a message can be parsed into variables in the IF portion of an IF-THEN automation statement. You can use the variables as parameters of the command list you call as an action in the THEN portion of the statement. This allows you to ignore certain characters of the message text (such as commas and apostrophes) instead of treating them as command syntax elements. For a complete definition of the syntax of the IF-THEN automation statement, see *NetView Administration Reference*.

Sending Messages to the MVS Operator Console

This section describes general-use programming interfaces, which allow the customer to write programs that use the services of the NetView program. This section ends where "Routing Messages from Command Lists" on page 142 begins.

There are three NetView commands that can be used in message automation command lists to send and remove messages to and from the MVS system console. These commands run in an MVS environment only. The three commands are:

WTO sends a message to the MVS operator console.

WTOR sends a message to the MVS operator console and waits for a reply.

DOM cancels a WTO.

When a command list is driven from a message automation table, certain REXX message functions or NetView command list language message control variables are assigned values when the command list starts. These values are based on the message that drives the command list. These message functions or control variables are also assigned values when a MSGREAD instruction is issued or an &WAIT control statement is satisfied within the command list. The values are replaced after each MSGREAD or &WAIT. The values assigned by message automation, MSGREAD, or &WAIT are called the system values.

The WTO and WTOR commands use the values of the message functions or control variables as input when the commands are processed. However, before you issue a WTO or WTOR command, you have the option of changing the system values to your own user-assigned values.

You cannot assign a value to a REXX function. Therefore, with REXX you must assign a value to a variable with the same name as the function (but without the parentheses at the end). For example, a MSGREAD instruction reads a message and that message assigns MCSFLAG() a value of 10000100. If you want to change the value before issuing a WTO command, you could use the following assignment statement to give a new value to the MCSFLAG variable:

```
MCSFLAG = '01000100'
```

Before processing the WTO or WTOR command, a REXX command list checks to see if any variables with the same names as the message functions have been set. If so, the command list uses the user-assigned values as input to the command. If not, the command list uses the current system values contained in the functions.

The MSGREAD instruction does not change the user-assigned variable values. If you want to go back to using the system values that MSGREAD assigns to the functions, use the REXX DROP instruction to drop the variables before issuing the WTO or WTOR command (for example, DROP MCSFLAG). See *REXX Reference* for information on the DROP instruction.

With the NetView command list language, you can use assignment statements to directly change the values of the message control variables. For example, to change the value of &MCSFLAG, you could use the following assignment statement:

```
&MCSFLAG = 01000100
```

When processing a WTO or WTOR command, a NetView command list language command list uses the current values of the message control variables regardless of whether the value is a system value or user-assigned value.

WTO

WTO is a NetView command that allows you to send a message to the MVS operator console. In an automation task command list written to process an MVS WTO message, you can use the NetView WTO command as an alternative to automatic processing. For example, use a WTO command for instances that require operator intervention, such as adding paper to a printer or choosing among several processing alternatives.

Figure 94 shows the syntax of the WTO command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding WTO in command lists written in the NetView command list language, do not include the quotes.

'WTO *messagetext'*

Figure 94. WTO Command Syntax

messagetext
is the message you want to send to the system console. You can send a character string or use a variable name set to the value of the message you want to send.

For REXX command lists, character strings should be enclosed within quotes with the command. If you use a variable, put a blank after the command, close the quotes, then put the name of the variable outside the quotes. For example, if the message is contained in a variable named MSG1, you would code:

```
'WTO 'MSG1
```

Incorrect usage of the WTO command to display multi-line messages can cause the MVS operator's console to hang.

The WTO command does not provide error checking to enforce proper usage of the REXX variables or the NetView command list language control variables that are used as input to the command.

The WTO command uses the values of the following REXX variables or NetView command list language control variables as input:

- AREAID, &AREAID

  Note: If the WTO command is not issued for a MLWTO message, (the LINETYPE variable or the &LINETYPE control variable is blank), then AREAID or &AREAID is not used or checked for a valid value.

- DESC, &DESC
- LINETYPE, &LINETYPE

- MCSFLAG, &MCSFLAG
- MSGTYP, &MSGTYP
- ROUTECDE, &ROUTCDE
- SMSGID, &SMSGID
- SYSCONID, &SYSCONID.

The values of these variables determine how the WTO command is processed. The variables provide the same input as the keywords on an MVS WTO macro. If a command list does not set the variables before issuing the WTO command, their values default to the current system values. For REXX command lists, the current system values are contained in the functions that correspond to the variable names. For example, the current system value for the REXX SYSCONID variable is contained in the SYSCONID() function.

For more information on the REXX variables used as input to WTO, see "Message Processing Information" on page 53. For more information on the NetView command list language control variables used as input to WTO, see "Message Processing Information" on page 84. For more information on the MVS WTO macro, see *MVS System Programming Library: System Macros and Facilities, Vol. 2.*

The WTO command returns values in the following REXX variables or NetView command list language control variables:

- RC, &RETCODE
- SMSGID, &SMSGID.

The return code, RC or &RETCODE, indicates the processing results as follows:

Code	**Meaning**
0	Processing successful
8	No storage available to continue processing
100	Invalid AREAID or &AREAID
104	Invalid SMSGID or &SMSGID length
108	Invalid SMSGID or &SMSGID value
112	Invalid SYSCONID or &SYSCONID length
116	Invalid SYSCONID or &SYSCONID value
120	Internal decimal convert failure
124	Command List dictionary update failure
128	Null text without end specified
132	Command is not allowed under PPT
136	Invalid LINETYPE or &LINETYPE.

A return code with a value greater than 200 indicates that the return code was passed from the MVS WTO macro. Subtract 200 from the value of the return code. The new value corresponds to the return code that was passed from the MVS WTO macro. Look up the meaning of the MVS WTO macro return code in *MVS System Programming Library: System Macros and Facilities, Vol. 2.* For example, if you receive a return code of 208, look in the MVS documentation for the meaning of return code 8 from the MVS WTO macro.

WTOR

WTOR is a NetView command that allows you to send a message to the MVS operator console and request a reply. Command lists that use the WTOR command will not complete until an operator replies. Therefore, use WTOR with care. If the command list is written in REXX, the operator reply is stored in the WTOREPLY variable, and the ID of the system console that replied is stored in the SYSCONID variable. If the command list is written in the NetView command list language, the operator reply is stored in the control variable &WTOREPLY, and the ID of the system console that replied is stored in &SYSCONID.

Figure 95 shows the syntax of the WTOR command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding WTOR in command lists written in the NetView command list language, do not include the quotes.

'WTOR *messagetext'*

Figure 95. WTOR Command Syntax

messagetext
: is the message you want to send to the system console. You can send a character string or use a variable name set to the value of the message you want to send.

  For REXX command lists, character strings should be enclosed within quotes with the command. If you use a variable, put a blank after the command, close the quotes, then put the name of the variable outside the quotes. For example, if the message is contained in a variable named MSG1, you would code:

```
'WTOR 'MSG1
```

The WTOR command does not provide error checking to enforce proper usage of the REXX variables or the NetView command list language control variables that are used as input to the command.

The WTOR command uses the values of the following REXX variables or NetView command list language control variables as input:

- DESC, &DESC
- MCSFLAG, &MCSFLAG
- MSGTYP, &MSGTYP
- ROUTECDE, &ROUTCDE
- SYSCONID, &SYSCONID.

The values of these variables determine how the WTOR command is processed. The variables provide the same input as the keywords on an MVS WTOR macro. If a command list does not set the variables before issuing the WTOR command, their values default to the current system values. For REXX command lists, the current system values are contained in the functions that correspond to the variable names. For example, the current system value for the REXX SYSCONID variable is contained in the SYSCONID() function.

If the command list is invoked from a message automation table, the current system values are set according to the message that activated the command list. Also, the current system values are reset according to messages that are processed within the command list by MSGREAD for REXX command lists or &WAIT for command lists written in the NetView command list language.

For more information on the REXX variables used as input to WTOR, see "Message Processing Information" on page 53. For more information on the NetView command list language control variables used as input to WTOR, see "Message Processing Information" on page 84. For more information on the MVS WTOR macro, see *MVS System Programming Library: System Macros and Facilities, Vol. 2.*

The WTOR command returns values in the following REXX variables or NetView command list language control variables:

- RC, &RETCODE
- SYSCONID, &SYSCONID
- WTOREPLY, &WTOREPLY.

The return code, RC or &RETCODE, indicates the processing results as follows:

Code	**Meaning**
0	Processing successful
100	Null message text or running under PPT
104	SYSCONID or &SYSCONID more than 10 digits
108	SYSCONID or &SYSCONID not numeric
112	Task posted to terminate
116	WTOREPLY or &WTOREPLY command list dictionary update failure.

DOM

DOM is used in an automation task command list to remove a WTO message from the operator console. You can use DOM to remove action messages after checking to see that the action was taken. DOM uses the SMSGID variable in REXX command lists or the &SMSGID control variable in NetView command list language command lists to determine which message to remove. If you do not assign a value to SMSGID or &SMSGID, the current system value is used. The WTO command resets the value of SMSGID or &SMSGID each time the command issues a message. For REXX, if the SMGSID variable is not set, WTO uses the value contained in the SMSGID() function. See "Message Processing Information" on page 53 for more information about SMSGID or see "Message Processing Information" on page 84 for more information about &SMSGID.

Figure 96 shows the syntax of the DOM command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding DOM in command lists written in the NetView command list language, do not include the quotes.

```
'DOM'
```

Figure 96. DOM Command Syntax

The DOM command does not provide error checking to enforce proper usage of the REXX variables or the NetView command list language control variables that are used as input to the command.

The return code, RC or &RETCODE, indicates the processing results as follows:

Code	Meaning
0	Processing successful
4	Syntax error
8	No storage available to continue processing
100	SMSGID or &SMSGID had too many numerics
104	SMSGID or &SMSGID was not numeric
108	Not invoked from a command list.

Routing Messages from Command Lists

This section describes general-use programming interfaces, which allow the customer to write programs that use the services of the NetView program. This section ends where "Parsing Variables with PARSEL2R" on page 144 begins.

In message driven command lists, use the MSGROUTE command to route the message driving the command list to operators or groups of operators. Use MSGROUTE when the decision where to route the message cannot be made in the message automation table. For example, use MSGROUTE if you need to check the value of global variables or the message text of a line other than the first line in a multi-line write-to-operator message, before you decide where to route the message.

When MSGROUTE routes a message, message automation does not process the message a second time.

Figure 97 shows the syntax of the MSGROUTE command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. If you want to use variables for any parts of the command, leave the variable name outside of the quotes. When coding MSGROUTE in command lists written in the NetView command list language, do not include the quotes.

```
'MSGROUTE oper [,...]
          [action-name(Y|N)]'
```

Figure 97. MSGROUTE Command Syntax

oper[,...]
: the operator identifier of the operators to whom the message is routed. The operator identifier must be defined to NetView with an OPERATOR definition statement. See *NetView Administration Reference* for information on the OPERATOR definition statement. The maximum length of an operator identifier is 8 characters. You can code as many operator identifiers as needed.

  You can also specify group identifiers for the groups of operators to whom the message is routed. The group identifier must be defined to NetView with the ASSIGN command. See *NetView Operation* for information about the ASSIGN command. The maximum length of a group identifier is 8 characters, and it must begin with a plus (+) sign.

action-name(Y|N)
the actions NetView should take when routing the message. Any or all of the following *action-names* can be specified:

BEEP
determines whether an audible alarm is sounded when the message is displayed. If BEEP is not specified, the default is BEEP(N).

DISPLAY
determines whether the message is displayed. If DISPLAY is not specified, the default is DISPLAY(Y).

HCYLOG
determines whether the message is placed in the hard-copy log. If HCYLOG is not specified, the default is HCYLOG(Y).

HOLD
determines whether the message is held on the operator's screen after it is displayed. If HOLD is not specified, the default is HOLD(N).

NETLOG(N)|(Y *[indicator-number]* **[*]** *[oper[,...]]* *[+grp[,...]]***)**
determines whether the message is placed in the NetView log and whether the message activates a status monitor important message indicator for specified operators or groups of operators. If NETLOG is not specified, the default is NETLOG(Y).

indicator-number	identifies the status monitor important message indicator.
*	means the message is logged as important for the operator task that the message is routed to, or the current operator task (the task where the message is intercepted for automation checking).
oper[,...]	the operator identifier of the operators for whom the message is logged as important. The operator identifier must be defined to NetView with an OPERATOR definition statement (see *NetView Administration Reference*). The maximum length of an operator identifier is 8 characters. You can code as many operator identifiers as needed.
+grp[,...]	the group identifier of the groups of operators for whom the message is logged as important. The maximum length of a group identifier is 8 characters, and it must begin with a plus (+) sign. Define group identifiers with the ASSIGN command. See *NetView Operation* for more information about the ASSIGN command.

If the operator is not in status monitor or log browse but is logged on, message CNM039I is displayed:

```
CNM039I AN IMPORTANT MESSAGE HAS BEEN LOGGED -
PLEASE BROWSE THE NETVIEW LOG.
```

If only an *indicator-number* is specified, the message is logged as important for the authorized receiver. The following example shows how NETLOG is coded with only an indicator-number:

```
MSGROUTE OPER1 NETLOG(Y 2)
```

The message is routed to OPER1. The message is also placed in the NetView log and is logged as an important message with a status monitor important message indicator number of 2.

If an *indicator-number* and a list of operators or groups of operators are specified, the message is logged as important for the operators and groups of operators listed. The following example shows how a message is logged when an *indicator-number* and a list of operators and groups of operators are specified:

```
MSGROUTE OPER4 NETLOG(Y 2 * OPER1 +GRP5 OPER6)
```

The message is routed to OPER4. The message is also placed in the NetView log and is defined as an important message with a status monitor important message indicator number of 2. The message activates a status monitor important message indicator for OPER1, OPER6, all of the operators assigned to group +GRP5, and the current operator. If operators OPER1 and OPER6 are also assigned to group +GRP5, each operator receives only one copy of message CNM039I (if they are not in STATMON).

SYSLOG
determines whether the message is placed in the system log. If SYSLOG is not specified, the default value is SYSLOG(N).

The return code, RC or &RETCODE, indicates the processing results, as follows:

Code	**Meaning**
8	Operator or group identifier not specified or greater than 8 characters
12	Invalid value for message action
16	MSGROUTE not entered from a REXX or NetView command list language command list
20	MSGROUTE not issued from a message driven REXX or NetView command list language command list, or from the message automation table
24	Operator or group identifier or message action not in operator's scope
28	Storage request failed
32	DSIMQS failed to route message.

Parsing Variables with PARSEL2R

This section describes general-use programming interfaces, which allow the customer to write programs that use the services of the NetView program. This section ends where "Working with Multi-Line Messages" on page 151 begins.

The PARSEL2R command allows you to extract data from the character-string value of a variable and assign the extracted data to one or more variables using a set of rules called a "parsing template". To parse variables with the NetView command list language, you must use the PARSEL2R command. However, in REXX you can use either PARSEL2R or the REXX PARSE instruction. See *REXX Reference* for more information on the PARSE instruction.

Figure 98 on page 145 shows the syntax of the PARSEL2R command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding PARSEL2R in command lists written in the NetView command list language, do not include the quotes.

```
'PARSEL2R sourcevariable parsingtemplate'
```

Figure 98. PARSEL2R Command Syntax

sourcevariable
identifies a command list variable. In REXX command lists, *sourcevariable* must be coded within the quotes. In command lists written in the NetView command list language, *sourcevariable* must be coded without an ampersand. PARSEL2R extracts data from the value of the variable you named as the *sourcevariable*.

parsingtemplate
is a list of symbols, patterns, or character selectors, or a combination of any of these, separated by blanks. PARSEL2R uses this list as a template when parsing the source variable.

Symbols are command list variable names. In REXX command lists, code command list variable names inside the single quotes. In command lists written in the NetView command list language, code command list variable names without an ampersand. For more information on using symbols, see "Using Symbols in a Parsing Template" on page 146.

Patterns are coded using slashes (/) as delimiters. A pattern is the part of the source variable that you want to match. For more information on using patterns, see "Using Patterns in a Parsing Template" on page 147.

A character selector is coded using an asterisk (*) for each single character you want to extract from the source variable. For more information on using character selectors, see "Using Character Selectors in a Parsing Template" on page 150.

PARSEL2R sets the return code (RC or &RETCODE) to indicate the processing results as follows:

Code	**Meaning**
0	Processing successful
8	No storage available to continue processing
100	Not enough parameters
104	Blank input buffer
108	Command list dictionary lookup of source variable failure
112	Invalid hexadecimal data in template
116	Command list dictionary update failure
120	Trailing slash (/) missing.

For examples of how PARSEL2R can be used with multi-line write-to-operator (MLWTO) messages, see "Examples of Command Lists Processing MLWTO Messages" on page 156.

The following sections describe how to use symbols, patterns, and character selectors in a parsing template.

Using Symbols in a Parsing Template

The symbols in the parsing template identify command list variables. In REXX command lists, code command list variables within the quotes that enclose PARSEL2R. In command lists written in the NetView command list language, code command list variables without the ampersand. If only symbols appear in the parsing template, the source variable data is assigned token-by-token from left to right. Tokens are defined as a string of non-blank characters. Tokens in the source variable are separated from each other by one or more blanks. The tokens are assigned to the command list variables you identified with symbols in the parsing template.

Figure 99 shows three lines from a REXX message automation command list that uses a parsing template containing only symbols. Figure 100 shows the NetView command list language equivalent.

```
TITLE = 'DON''T TREAD(ROUGHLY) ON ME, PLEASE'
'PARSEL2R TITLE A1 A2 A3 A4 A5 A6 A7'
'PARSEL2R TITLE B1 B2 B3'
```

Figure 99. REXX PARSEL2R Example Using Symbols

```
&TITLE = 'DON'T TREAD(ROUGHLY) ON ME, PLEASE'
PARSEL2R TITLE A1 A2 A3 A4 A5 A6 A7
PARSEL2R TITLE B1 B2 B3
```

Figure 100. NetView Command List Language PARSEL2R Example Using Symbols

The resulting values of the variables show how the token-by-token assignment from left to right works. The following table shows the resulting values for the REXX and NetView command list language variables:

REXX Variable	NetView Variable	Value
A1	&A1	DON'T
A2	&A2	TREAD(ROUGHLY)
A3	&A3	ON
A4	&A4	ME,
A5	&A5	PLEASE
A6	&A6	null
A7	&A7	null
B1	&B1	DON'T
B2	&B2	TREAD(ROUGHLY)
B3	&B3	ON ME, PLEASE

Note: The value of B3 or &B3 is not a single token, but the remainder of the source variable after the PARSEL2R parsed it into the first two symbols.

Except for the last variable, leading blanks and trailing blanks are removed from each token in the string before it is assigned to a variable. Variable B3 or &B3 would have leading or trailing blanks, if TITLE contained extra blanks before ON or after PLEASE.

Using Patterns in a Parsing Template

A pattern is a character or string of characters expected to appear within the source variable. It is coded in the PARSEL2R parsing template using slashes (/) as delimiters. Patterns are used within a parsing template to divide the source variable into segments.

When a pattern that you coded in the parsing template occurs in the source variable, the preceding portion of the source variable is treated as a segment. The symbols you defined in the parsing template preceding the pattern are used to parse the tokens in the corresponding segment of the source variable.

Note: If you want to use a slash as a part of a pattern, code two consecutive slashes within the delimiter slashes. PARSEL2R reads this as one slash to be matched in the source variable. Two consecutive slashes by themselves (outside of delimiters) means to end the parse.

Figure 101 shows how a parsing template containing patterns and symbols can be used in a REXX message automation command list. Figure 102 shows the NetView command list language equivalent.

```
PARSIT = 'DON''T TREAD(ROUGHLY) ON ME, PLEASE'
'PARSEL2R PARSIT A1 A2 A3 /(/ B1 B2 B3 /)/ C1 C2 C3 /,/ D1 D2 D3'
```

Figure 101. REXX PARSEL2R Example Using Patterns and Symbols

```
&PARSIT = 'DON'T TREAD(ROUGHLY) ON ME, PLEASE'
PARSEL2R PARSIT A1 A2 A3 /(/ B1 B2 B3 /)/ C1 C2 C3 /,/ D1 D2 D3
```

Figure 102. NetView Command List Language PARSEL2R Example Using Patterns and Symbols

The following table shows the resulting values for the REXX and NetView command list language variables:

REXX Variable	NetView Variable	Value
A1	&A1	DON'T
A2	&A2	TREAD
A3	&A3	null
B1	&B1	ROUGHLY
B2	&B2	null
B3	&B3	null
C1	&C1	ON
C2	&C2	ME
C3	&C3	null
D1	&D1	PLEASE
D2	&D2	null
D3	&D3	null

Figure 103 on page 148 is another example of a parsing template containing patterns and symbols. Figure 104 on page 148 is the NetView command list language equivalent.

```
PARSIT = 'DON''T TREAD    ROUGHLY, ON ME'
'PARSEL2R PARSIT A1 A2 A3 /,/ A4'
```

Figure 103. REXX PARSEL2R Example Using Leading Blanks

```
&PARSIT = 'DON'T TREAD    ROUGHLY, ON ME'
PARSEL2R PARSIT A1 A2 A3 /,/ A4
```

Figure 104. NetView Command List Language PARSEL2R Example Using Leading Blanks

The following table shows the resulting values for the REXX and NetView command list language variables:

REXX Variable	**NetView Variable**	**Value**
A1	&A1	DON'T
A2	&A2	TREAD
A3	&A3	ROUGHLY
A4	&A4	ON ME

Note that because the variable right before a pattern (A3 or &A3 in the previous examples) is treated as the last variable in a segment, the leading and trailing blanks are not removed.

If the parsing template were specified as:

```
PARSEL2R PARSIT A1 A2 A3 AX /,/ A4
```

then the blanks would be removed from the A3 or &A3 variable, and the AX or &AX variable would be null.

You can use variables as part of a pattern (between the slashes). When a variable is part of a pattern, it needs to be coded outside of the quotes in a REXX command list or with an ampersand in command lists written in the NetView command list language.

Figure 105 shows three lines from a REXX command list that uses a parsing template with a pattern containing a variable. Figure 106 on page 149 shows the NetView command list language equivalent.

```
A0 = ROUGHLY
PARSIT = 'DON''T TREAD (ROUGHLY) ON ME'
'PARSEL2R PARSIT A1 A2 /('A0')/ B1'
```

Figure 105. REXX PARSEL2R Example Using a Pattern that Contains a Variable

```
&A0 = ROUGHLY
&PARSIT = DON'T TREAD (ROUGHLY) ON ME
PARSEL2R PARSIT A1 A2 /(&A0)/ B1
```

Figure 106. NetView Command List Language PARSEL2R Example Using a Pattern that Contains a Variable

The following table shows the resulting values for the REXX and NetView command list language variables:

REXX Variable	NetView Variable	Value
A1	&A1	DON'T
A2	&A2	TREAD
B1	&B1	ON ME

In the REXX example, because A0 is outside the the quotes, its value is used as part of the pattern. The pattern becomes (ROUGHLY). Likewise in the NetView command list language example, because &A0 is outside the quotes, its value is used as part of the pattern. The pattern becomes (ROUGHLY).

You can also use hexadecimal codes in the parsing template pattern. Code a hexadecimal pattern using an X before the slashes. PARSEL2R matches the hexadecimal code in the template with the character in the source variable that corresponds to your system.

Figure 107 shows how to code a parsing template, containing a hexadecimal pattern, in a REXX command list. Figure 108 shows the NetView command list language equivalent.

```
'PARSEL2R PARSIT A1 A2 X/5B/ A3'
```

Figure 107. REXX PARSEL2R Example Using a Hexadecimal Pattern

```
PARSEL2R PARSIT A1 A2 X/5B/ A3
```

Figure 108. NetView Command List Language PARSEL2R Example Using a Hexadecimal Pattern

Using patterns and symbols in a parsing template gives you a powerful tool to use when coding your command lists. For example, you can use PARSEL2R to code a source variable in your command list that contains a small table. You can also use the combination of symbols and patterns to search the source variable and assign a token to a variable, based on the matching pattern. This table can contain a string of alternating variables and labels. You can then use PARSEL2R to match a variable with a label to define the flow of logic within your command list.

Using Character Selectors in a Parsing Template

Character selectors in a PARSEL2R are coded as one or more asterisks (*), indicating that the preceding symbol must be assigned one or more characters from the source variable. If a character selector does not follow a symbol in the template (that is, it is coded at the beginning of the template or following a pattern), PARSEL2R skips that number of characters.

For example, if the source variable is:

```
DSI039I MSG FROM CNM01PPT: COMMON GLOBAL VARIABLES HAVE BEEN SET
```

and the parsing template is:

```
/FROM / DOMNAM ***** TASK /:/
```

then the following values are assigned:

```
DOMNAM or &DOMNAM = CNM01
TASK or &TASK = PPT
```

Character selectors are usually used to break up a single token into multiple variables.

Figure 109 and Figure 110 show a parsing template using character selectors to change the value of a REXX message variable or a NetView command list language control variable. When the command list is entered, MCSFLAG or &MCSFLAG is set to 00000110. The command list statements set bit 6 to 0.

```
'PARSEL2R MCSFLAG BITS1_5 ***** BIT6 * BITS7_8 **'
BIT6 = 0
MCSFLAG = BITS1_5 || BIT6
MCSFLAG = MCSFLAG || BITS7_8
```

Figure 109. REXX PARSEL2R Example Using Character Selectors

```
PARSEL2R MCSFLAG BITS1_5 ***** BIT6 * BITS7_8 **
&BIT6 = 0
&MCSFLAG = &CONCAT &BITS1_5 &BIT6
&MCSFLAG = &CONCAT &MCSFLAG &BITS7_8
```

Figure 110. NetView Command List Language PARSEL2R Example Using Character Selectors

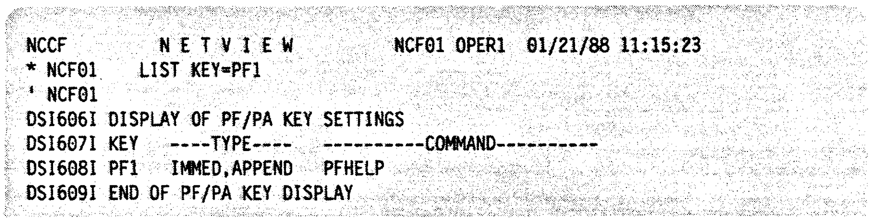
After executing the command list statements in the previous examples, the value of MCSFLAG or &MCSFLAG is 00000010.

Working with Multi-Line Messages

This section describes general-use programming interfaces, which allow the customer to write programs that use the services of the NetView program. This section ends where "Using the SDOMAIN Command with the QUIET Option" on page 158 begins.

Some commands return a reply that appears to be a sequence of separate messages, when in fact the reply is a single multi-line write-to-operator (MLWTO) message. NetView treats a MLWTO message as a single message. Only the first message identifier that appears as part of a MLWTO message is made available to satisfy a REXX TRAP instruction, or a NetView command list language &WAIT command. The first line of a multi-line message is also the only line used for comparisons in message automation tables.

Figure 111 shows a MLWTO message as it would appear on a NetView operator's screen. The message is in response to a LIST KEY=PF1 command. The MLWTO message appears to be a sequence of several separate messages, but the single quote that appears as the message type identifier, identifies the message as a single MLWTO message from NetView. A double quote identifies a multi-line message from an IBM product other than NetView. An equal sign identifies a user-written multi-line message.



```
NCCF         N E T V I E W         NCF01 OPER1  01/21/88 11:15:23
* NCF01    LIST KEY=PF1
' NCF01
DSI606I DISPLAY OF PF/PA KEY SETTINGS
DSI607I KEY   ----TYPE----   -----------COMMAND----------
DSI608I PF1   IMMED,APPEND   PFHELP
DSI609I END OF PF/PA KEY DISPLAY
```

Figure 111. Example Multi-Line Message

TRAP, &WAIT, and message automation table processing use only the first line of a multi-line message. However, NetView provides three commands that allow you to work with multi-line messages in a command list. These commands allow you to work with information in each individual line of a multi-line message. The three commands are:

GETMSIZE determines the number of lines of a multi-line message.

GETMTYPE determines the line type of a specific line in a multi-line message.

GETMLINE assigns the text of a specific line of a multi-line message to a specified variable.

GETMSIZE

GETMSIZE is a command used in command lists to determine the number of lines in a multi-line message. Use this command in a command list that was driven by message automation or that has processed a message using MSGREAD (REXX) or &WAIT (NetView command list language).

Figure 112 shows the syntax of the GETMSIZE command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding GETMSIZE in command lists written in the NetView command list language, do not include the quotes.

```
'GETMSIZE variablename'
```

Figure 112. GETMSIZE Command Syntax

variablename
identifies a command list variable coded in this command. If the command list is written in the NetView command list language, the ampersand should be removed from the variable name. GETMSIZE sets the value of the variable to the number of lines in the multi-line message. If the message is a single-line instead of a multi-line message, the variable value is set to 1.

GETMSIZE sets the return code (RC or &RETCODE) to indicate the processing results, as follows:

Code	**Meaning**
0	Successful completion, variable has been set
8	No storage available to continue processing
100	No variable was specified
104	Error in multi-line message
108	An invalid variable name was given.

For example, assume the following statement is coded in an automation command list:

```
GETMSIZE  NUMLINES
```

If the command list containing this command is triggered by the message in Figure 113, the variable &NUMLINES is set to the value of 7.

```
IEE104I 13.02.15 87.189 ACTIVITY 607                                    C
 JOBS      M/S      TS USERS      SYSAS      INITS      ACTIVE/MAX VTAM
00000     00007      00000        00008      00001        00000/00300
 LLA       LLA       LLA      NSW  S   JES2JES2     IEFPROC  NSW  S
 SYSLOG    621       IEFPROC  OWT  S   BASENET  BASENET  VTAM      NSW  S
 TSO       TSO       TCAS     OWT  S   NETVREL2 NETVREL2 NETVIEW   NSW  S
 NETV2     NETV2     NETVIEW  NWS  S
```

Figure 113. IEE104I Message to Trigger an Automation Task Command List

See "Examples of Command Lists Processing MLWTO Messages" on page 156 for examples of command lists that show how GETMSIZE is used with MLWTO messages.

GETMTYPE

GETMTYPE is a command used in command lists to determine the line type of an individual line in a multi-line message. Use this command in a command list that was driven by message automation or that has processed a message using MSGREAD (REXX) or &WAIT (NetView command list language).

Figure 114 shows the syntax of the GETMTYPE command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding GETMTYPE in command lists written in the NetView command list language, do not include the quotes.

```
'GETMTYPE variablename number'
```

Figure 114. GETMTYPE Command Syntax

variablename
identifies a command list variable coded in this command. If the command list is written in the NetView command list language, the ampersand should be removed from the variable name. GETMTYPE sets the value of this variable as one of the following line types:

blank	The message is a single-line message.
C	The line is a control line.
L	The line is a label line.
D	The line is a data line.
DE	The line is a data end line.
E	The line is an end line without data.

number
specifies the number of the line in the multi-line message for which you want line type information. For single-line messages, the value of *number* must be 1.

When coding an actual number for *number* in REXX command lists, the number should be inside the quotes that enclose GETMTYPE. For example:

```
'GETMTYPE TYPE1 3'
```

When coding a variable for *number* in REXX command lists, leave a blank after *variablename* and close the quotes. Code the name of the variable that contains the value for *number* outside the quotes. For example, if the value of *number* is contained in a variable named NUM1, you would code:

```
'GETMTYPE TYPE1 'NUM1
```

GETMTYPE also sets the return code (RC or &RETCODE) to indicate the processing results, as follows:

Code	Meaning
0	Processing successful
8	No storage available to continue processing
100	Either *variablename* or *number* was omitted
104	*number* has too many digits
108	*number* is not a numeric value
112	*number* does not equal 1 for a single-line message
116	Requested line number does not exist
120	Internal command list processing error

124 Internal command list processing error
128 *variablename* invalid.

For example, you can use GETMTYPE along with GETMSIZE in a command list to test each line of a multi-line message for the type. If you use a variable name for the *number* field in the GETMTYPE command, you can test each line. Using GETMSIZE, you can code your command list to test the correct number of lines for each message it receives. If the command list received the message illustrated in Figure 113 on page 152:

- the line type of the first line would equal C
- the line type of the second and third lines would equal L
- the line type of the fourth through the sixth line would equal D
- the line type of the seventh line would equal DE.

GETMLINE

GETMLINE is used within a command list to assign the text of an individual line of a multi-line message to a specified variable. Use this command in a command list that was driven by message automation or that has processed a message using MSGREAD (REXX) or &WAIT (NetView command list language).

Figure 115 shows the syntax of the GETMLINE command.

Note: The command is enclosed in single quotes to avoid substitution by REXX. When coding GETMTYPE in command lists written in the NetView command list language, do not include the quotes.

'**GETMLINE** *variablename number*'

Figure 115. GETMLINE Command Syntax

variablename
identifies a command list variable coded in this command. If the command list is written in the NetView command list language, the ampersand should be removed from the variable name. GETMLINE assigns the value of the line you specify to this variable name.

number
identifies the number of the line in the multi-line message from which you want to obtain the value.

When coding an actual number for *number* in REXX command lists, the number should be inside the quotes that enclose GETMLINE. For example:

```
'GETMLINE LINE1 2'
```

When coding a variable for *number* in REXX command lists, leave a blank after *variablename* and close the quotes. Code the name of the variable that contains the value for *number* outside the quotes. For example, if the value of *number* is contained in a variable named NUM1, you would code:

```
'GETMLINE LINE1 'NUM1
```

GETMLINE also sets the return code (RC or &RETCODE) to indicate the processing results, as follows:

Code	**Meaning**
0	Processing successful
8	No storage available to continue processing
100	Either *variablename* or *number* was omitted
104	*number* has too many digits
108	*number* is not a numeric value
112	*number* does not equal 1 for a single-line message
116	Requested line number does not exist
120	Internal command list processing error
124	Internal command list processing error
128	*variablename* invalid.

For example, an automation task command list written in the NetView command list language contains the following two lines:

```
&NUM = 2
GETMLINE  SECOND  &NUM
```

These two lines in a command list place the text of the second line of a multi-line message into the command list variable &SECOND.

See "Examples of Command Lists Processing MLWTO Messages" on page 156 for example command lists that show how GETMLINE can be used with MLWTO messages.

Examples of Command Lists Processing MLWTO Messages

The command lists in Figure 116 and Figure 117 on page 157 are examples of how you can code your command list to process based on the information in the individual lines of a multi-line message. Figure 116 is written in REXX. Figure 117 is written in the NetView command list language.

```
/***********************************************************************/
/* SESSCNT  COMMAND LIST                                               */
/* ---------------------                                               */
/* FUNCTION:    This command list counts the number of OPCTCL          */
/*              sessions and FLSCN sessions that are active.           */
/*                                                                     */
/* INPUT PARMS: None                                                   */
/*                                                                     */
/* OUTPUT:      Two informational messages that display the number of */
/*              OPCTCL and FLSCN sessions to the operator.             */
/***********************************************************************/
OPCTLCNT = 0                                   /* init OPCTL counter  */
FLSCNCNT = 0                                   /* init FLSCN counter  */
'TRAP AND SUPPRESS MESSAGES *'                 /* TRAP the MLWTO msg  */
'LISTSESS'                                     /* issue the command   */
'WAIT 5 SECONDS FOR MESSAGES'                  /* WAIT for the msg    */
SELECT                                         /* SELECT an EVENT     */
  WHEN EVENT() = 'M' THEN                      /* message received    */
    DO                                         /* process the message */
      'MSGREAD'                                /* read the message in */
      'GETMSIZE ' NUMLINES                     /* get number of lines */
      DO CNTR = 4 TO NUMLINES                  /* loop thru MLWTO buf */
        'GETMLINE LINE' CNTR                   /* get a line in buf   */
                                               /* parse the line out  */
        'PARSEL2R LINE APPLNM SRCLUNM SESSNM TYPE RESTLINE'
        IF TYPE = 'OPCTL' THEN                 /* OPCTL session ?     */
          OPCTLCNT = OPCTLCNT + 1              /* yes,increment count */
        ELSE                                   /* must be FLSCN sess  */
          FLSCNCNT = FLSCNCNT + 1              /* increment count     */
        'TRAP NO MESSAGES'                     /* end message trapping*/
        'FLUSHQ MESSAGES'                      /* flush message queue */
      END                                      /* loop thru MLWTO buf */
    END                                        /* process the message */
  OTHERWISE                                    /* event not a message */
    SAY 'ERROR PROCESSING LISTSESS COMMAND'    /* issue an error msg  */
END                                            /* SELECT an EVENT     */
/*   issue messages to operator with count of FLSCN & OPCTL sessions  */
SAY 'YOU HAVE ' OPCTLCNT ' OPCTL SESSIONS ACTIVE AND'
SAY '          ' FLSCNCNT ' FLSCN SESSIONS ACTIVE'
```

Figure 116. Command List Using Multi-Line Messages - REXX Example

```
&CONTROL ERR
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
* COMMAND LIST AUTHTOTE
*
* THIS COMMAND LIST DISPLAYS A COUNT OF HOW MANY MESSAGES (OR GROUPS
*
*   OF MESSAGES, USING THE * SUFFIX) HAVE BEEN ASSIGNED TO BE
*
*   ROUTED TO SPECIFIC AUTHORIZED RECEIVERS.
*
* IT IS CALLED AS FOLLOWS:   AUTHTOTE
*
*                            AUTHTOTE DISPLAY
*
*                            AUTHTOTE SUPPRESS
*
* THE FIRST TWO FORMS WILL DISPLAY THE MESSAGES RECEIVED ("DISPLAY"
*   IS THE DEFAULT) AND A COUNT OF THE TOTAL NUMBER OF ASSIGNMENTS
*   MADE, AND THE LAST FORM WILL SIMPLY DISPLAY THE TOTAL NUMBER
*   OF ASSIGNMENTS.
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* *
* SAVE THE INPUT PARAMETER
&OPT = &1
* INITIALIZE THE COUNTER FOR THE NUMBER OF MESSAGES ASSIGNED TO
* AUTHORIZED RECEIVERS.
&COUNT = 0
* SET THE COUNTER FOR THE NUMBER OF THE MESSAGE LINE TO BE
* PROCESSED TO 1.  THE WAIT STATEMENT PROCESSES THE FIRST MESSAGE LINE.
&MSZI = 0
* ISSUE THE COMMAND AND ENTER WAIT STATE
&WAIT ENDWAIT &OPT
&WAIT 'LIST MSG=AUTH',DSI636I=-COUNTER,DSI643I=-NOEXIT, +
   *60=-TIMEOUT,*ENDWAIT=-GOIN
-NXTLINE
* ADD 1 TO THE COUNTER FOR NUMBER OF LINES PROCESSED
&MSZI = &MSZI + 1
* IF THE NUMBER OF THE LINE TO BE PROCESSED IS GREATER THAN THE
* NUMBER IF LINES IN THE MESSAGE, THEN DISPLAY THE RESULTS
&IF &MSZI = &NUMLINES &THEN &GOTO -ENDIT
* ASSIGN THE TEXT OF THE NEXT LINE OF THE MESSAGE TO MSGTXT
GETMLINE MSGTXT &MSZI
* PARSE MSGTXT SO THAT THE MESSAGE ID OF THE LINE IS PLACED IN
* THE MSGID VARIABLE.
PARSEL2R MSGTXT MSGID MSGSTR
&IF &MSGID = DSI642I &THEN &GOTO -ENDIT
&IF &MSGID = DSI643I &THEN &GOTO -NOEXIT
&IF &MSGID = DSI636I &THEN &GOTO -COUNTER
&GOTO -NXTLINE
```

Figure 117 (Part 1 of 2). Command List Using Multi-Line Messages - NetView Command List Language Example

```
-NOEXIT
*   THIS INSURES THAT THE OPERATOR IS INFORMED OF THE RESULTS OF
*   THE COMMAND WHEN NO MESSAGES HAVE BEEN ASSIGNED, BUT THE
*   MESSAGE INFORMING THE OPERATOR HAS BEEN SUPPRESSED
&IF .&OPT = .SUPPRESS &THEN &GOTO -ENDIT
&EXIT
-COUNTER
* IF THE FIRST MESSAGE LINE IS BEING PROCESSED, DETERMINE THE NUMBER
* OF LINES FOR THE MESSAGE
IF &MSZI = 1 &THEN &GOTO -LINES
-COUNTER1
* TOTAL THE NUMBER OF MESSAGES ASSIGNED TO PRIMARY RECEIVERS
&COUNT = &COUNT + 1
&GOTO -NXTLINE
-LINES
* ASSIGN THE NUMBER OF LINES IN THE MESSAGE TO THE VARIABLE NUMLINES
GETMSIZE NUMLINES
&GOTO -COUNTER1
-GOIN
&WRITE COMMAND LIST AUTHTOTE ENDED BY "GO" COMMAND -- COUNT INCOMPLETE
&EXIT
-TIMEOUT
&WRITE NO RESPONSE WITHIN 60 SECONDS. COMMAND LIST TERMINATING
&EXIT
-ENDIT
* DISPLAY THE COUNT OF MESSAGES ASSIGNED TO AUTH RECEIVERS
&WRITE &COUNT MESSAGE(S) ASSIGNED TO SPECIFIC AUTH RECEIVERS
&EXIT
```

Figure 117 (Part 2 of 2). Command List Using Multi-Line Messages - NetView Command List Language Example

Using the SDOMAIN Command with the QUIET Option

This section describes general-use programming interfaces, which allow the customer to write programs that use the services of the NetView program. This section ends where "Hints for Implementing Message Automation" on page 161 begins.

The hardware monitor SDOMAIN command can be issued with the QUIET option from a command list, to set the domain and return a message for automation. To trap the message in a REXX command list, issue the SDOMAIN command after issuing a TRAP instruction but before issuing a WAIT command. To trap the message in a command list written in the NetView command list language, issue the SDOMAIN command from the &WAIT statement.

NetView supplies a sample command list written in the NetView command list language that issues this command (see Figure 118 on page 159). In this command list, whenever an SDOMAIN message occurs that is not tested on the &WAIT statement, the message is written to the command facility panel and the command list stops execution.

The most common messages produced by the SDOMAIN command are:

BNJ911I	Current domain now XXXX, was YYYY
BNJ912I	Attempting a cross domain session to an incompatible level of NetView
BNJ924I	Attempting a cross domain session with an undefined domain
BNJ926I	SD/SDOMAIN command failed, current domain is unchanged.

Figure 118 is an example of how to invoke the SDOMAIN command with the QUIET option from a command list.

```
    CLIST
&CONTROL ERR
************************************************************************
*  (C) COPYRIGHT IBM CORP. 1988                                        *
*  LAST CHANGE:                                                        *
*  DESCRIPTION: THIS COMMAND LIST ISSUES THE SDOMAIN QUIET COMMAND     *
*               WHICH INITIATES A CROSS DOMAIN SESSION WITHOUT         *
*               DISPLAYING THE NPDA MAIN MENU.  IF THE SDOMAIN QUIET   *
*               COMMAND IS SUCCESSFUL, THE ALERTSD COMMAND IS ISSUED.  *
*  CNME0044 CHANGED ACTIVITY:                                          *
*  CHANGE CODE  DATE      DESCRIPTION                                  *
*  ------ ----- --------- -------------------------------------------*
************************************************************************
* THE FIRST (AND ONLY) PARAMETER EXPECTED BY THIS COMMAND LIST IS THE  *
* DOMAIN NAME FOR WHICH THE ALERTSD INFORMATION IS DESIRED.            *
************************************************************************
&DOMAINID = &1
************************************************************************
* IF A DOMAIN NAME IS NOT PASSED TO THE COMMAND LIST, THEN SET THE     *
* DOMAIN NAME TO THE DOMAIN THE USER IS LOGGED ONTO.                   *
************************************************************************
&IF .&DOMAINID NE . &THEN &GOTO -XDOMAIN
&DOMPART = &LENGTH &APPLID
&DOMPART = &DOMPART - 3
&DOMAINID = &SUBSTR &APPLID 1 &DOMPART
************************************************************************
* INVOKE THE SDOMAIN COMMAND WITHIN THE &WAIT STATEMENT TO TRAP THE    *
* MESSAGES PUT OUT BY HARDWARE MONITOR.                                *
************************************************************************
-XDOMAIN
&WAIT CONTWAIT SUPPRESS
&WAIT 'NPDA SDOMAIN &DOMAINID QUIET' +
            BNJ911I=-NPDACM          +
            BNJ912I=-INCOMPAT        +
            BNJ924I=-BADXDOM         +
            BNJ926I=-SDFAIL          +
            *ERROR=-ERROR            +
            *10=-TIMEOUT
&GOTO -ERROR
```

Figure 118 (Part 1 of 2). NetView Command List Language Command List Issuing SDOMAIN with QUIET option

```
************************************************************************
*                     DISPLAY APPROPRIATE MESSAGE                      *
************************************************************************
**
** CROSS DOMAIN SESSION TO AN INCOMPATIBLE LVL OF NETVIEW WAS ATTEMPTED
**
-INCOMPAT
GO
&WRITE IN CNME0044:  &MSGID &MSGSTR
&GOTO -END
**
** CROSS DOMAIN SESSION TO AN UNDEFINED DOMAIN WAS ATTEMPTED
**
-BADXDOM
&WRITE IN CNME0044:  &MSGID &MSGSTR
&WAIT CONTINUE
**
** THE SDOMAIN COMMAND FAILED
**
-SDFAIL
GO
&WRITE IN CNME0044:  &MSGID &MSGSTR
&GOTO -END
**
** AN UNEXPECTED ERROR OCCURRED
**
-ERROR
&WRITE IN CNME0044:  UNDETERMINED ERROR OCCURRED
&GOTO -END
**
** &WAIT TIMED OUT BEFORE ANY MESSAGES IT WAS TESTING FOR OCCURRED
**
-TIMEOUT
&WRITE IN CNME0044:  TIME OUT ON SDOMAIN COMMAND
&GOTO -END
************************************************************************
* SDOMAIN COMMAND WORKED CORRECTLY, ISSUE THE ALERTS DYNAMIC COMMAND   *
************************************************************************
-NPDACM
NPDA ALERTSD
&GOTO -END
************************************************************************
*                         END OF COMMAND LIST                          *
************************************************************************
-END
&EXIT
```

Figure 118 (Part 2 of 2). NetView Command List Language Command List Issuing SDOMAIN with QUIET option

Hints for Implementing Message Automation

This section provides suggestions to help you effectively implement message automation.

Suppressing Messages

You can suppress some messages so that the operator never receives them. To suppress messages with NetView message automation, make an entry in the message automation member. Assume, for instance, that you do not want the message IST400I TERMINATION IN PROGRESS FOR APPLID *applnm* to be displayed. Figure 119 shows what the message automation statement looks like.

```
IF MSGID='IST400I' THEN DISPLAY(N);
```

Figure 119. Message Automation Statement to Suppress Message

Determining What Task Controls a Command List

If you are not sure what type of task a command list will run under, have the command list check the TASK() function or &TASK control variable in the beginning of the command list. You can then use conditional processing to make the command list flexible enough to run differently under different tasks. See *REXX User's Guide* and *REXX Reference* or Chapter 7, "NetView Command List Language Branching" on page 107 for more information about conditional processing.

Testing Automation Command Lists

You can test command lists invoked from the message automation table by using SAY in REXX command lists or &WRITE or &BEGWRITE in command lists written in the NetView command list language. This method works if the message automation table does not check for the message type (HDRMTYPE() or &HDRMTYPE) or the optional system information. For example, you can send a message to an automation task to trigger the message-driven command list under that automation task. This allows you to test your automation command lists by enabling you to send messages that you expect the command list to handle during regular processing.

Figure 120 shows an example of a REXX command list using the SAY instruction that you could use to generate test messages.

```
/* COMMAND LIST TO TEST AUTOMATION COMMAND LISTS */

     TRACE ERRORS
     PARSE ARG PARMS
     SAY PARMS
     EXIT
```

Figure 120. REXX Command List to Test Automation Command Lists

Figure 121 on page 162 shows an example of a command list written in the NetView command list language using &WRITE that you could use to generate test messages.

```
TESTMSG CLIST
        &CONTROL ERR
        &WRITE &PARMSTR
        &EXIT 0
```

Figure 121. NetView Command List Language Command List to Test Automation Command Lists

You can then test the text of any single line message by typing the following as a NetView command, or within another command list:

```
TESTMSG your message text here
```

Figure 122 is a REXX command list that shows how to use the SAY instruction to generate a multiple-line message:

```
/* COMMAND LIST TO TEST MULTIPLE LINE MESSAGES */
   TRACE ERRORS
   SAY ,
'IEE000I THIS IS A TEST MULTIPLE LINE MESSAGE                         ',
'       THIS IS LINE 2                                                ',
'       THIS IS LINE 3                                                ',
'       THIS IS LINE 4                                                ',
'       THIS IS THE LAST LINE FOR IEE000I'
    EXIT
```

Figure 122. REXX Command List to Generate a Multiple-Line Message

Note: There must be 67 characters or blanks enclosed by the single quotes on each line of the SAY instruction, except the last line.

Figure 123 is a command list written in the NetView command list language that shows how to use &BEGWRITE to generate a multiple-line message:

```
TESTMLN CLIST
        &CONTROL ERR
        &BEGWRITE -ENDLINE
IEE000I THIS IS A TEST MULTIPLE LINE MESSAGE
        THIS IS LINE 2
        THIS IS LINE 3
        THIS IS LINE 4
        THIS IS THE LAST LINE FOR IEE000I
-ENDLINE
        &EXIT 0
```

Figure 123. NetView Command List Language Command List to Generate a Multiple-Line Message

In this example, it is necessary to type the message text into the command list before running, so the multiple-line message format can be produced. You can run the command list by typing TESTMLN as a NetView command.

Recovering From Looping Command Lists

It is possible to write command lists that will loop. For example, if you write a command list that is driven by a message issued by a command in the same command list, looping will occur. If a looping message-driven REXX command list has a WAIT, or a looping message-driven NetView command list language command list has an &WAIT or &PAUSE, issue the STACK command from the operator's console to recover. Then turn off message automation with the command AUTOMSG OFF. If there is no WAIT in a REXX command list or &WAIT or &PAUSE in a command list written in the NetView command list language, you can issue the AUTOMSG OFF command from your terminal. Once the looping has stopped, you can revise the command list.

Considering Operator Interaction

Command lists used for automation of unsolicited messages should not ask the operator for data. For example, a REXX command list using a WAIT or a NetView command list language command list using either &PAUSE VARS or &WAIT, requiring a GO command, is inappropriate. Consider how messages from a command list affect operator requests, and try to make automation command lists interfere as little as possible because automation runs at the same time that operators enter requests.

Other Common Automation Problems

Message automation is invoked after the command facility exit routines (for example, DSIEX02A, DSIEX06, DSIEX11) have been called, so changes made to messages in these routines affect message automation. For example, if a message is deleted by DSIEX02A, NetView does not invoke automation for that message. If a message is assigned to SYSOP or LOG as the primary receiver, NetView does not invoke automation for that message. Since message automation does not occur in the preceding instances, the DISPLAY keyword in the message automation member does not have any effect.

If the message processing facility is used to suppress a message with AUTO=YES coded and this message is used to drive a command list, when the command list is driven and a WTO is issued, the WTO is also suppressed. You must change the setting of the MCSFLAG() function or the &MCSFLAG control variable for the WTO to be displayed. See Figure 109 on page 150 for an example of how to change a function or control variable.

If a multi-line write-to-operator (MLWTO) message is used to drive a command list and a WTO is issued from the command list, the WTO may or may not be displayed, depending on the setting of LINETYPE() or &LINETYPE. You should check the setting of LINETYPE() or &LINETYPE, and if your WTO is a single line message, change the setting of LINETYPE() or &LINETYPE to a blank.

How to Set Up for Migration

NetView comes with a utility program called DSICNVRT that helps you in converting NCCF, and NetView Release 1 message automation to NetView Release 3 message automation. You will find the JCL statements or VM EXEC to run the program in the NetView SAMPLIB. See the *NetView Installation and Administration Guide* for details on how to run the DSICNVRT utility.

You must perform the following tasks when migrating to NetView Release 3 message automation.

For Migrating from NCCF to NetView Release 3:

If you have any CMDMDL statements in DSICMD to support the NCCF method of VTAM-only message automation, do one of the following:

- For MVS systems, set up the INPUT DD statement in your JCL to reference your DSICMD member.
- For VM systems, set up the INPUT FILEDEF in the EXEC file to reference your DSICMD file.

The DSICNVRT program produces a message automation statement in the file specified as output for every CMDMDL that defines a command list starting with IST.

For Migrating from NetView Release 1 to Release 3:

If you have been using NetView Release 1 message automation, do one of the following:

- For MVS systems, set up the INPUT and OUTPUT DD statements in your JCL for each message automation member and then run DSICNVRT to convert each of them.
- For VM systems, set up the FILEDEFS in your EXEC file for each message automation member and then run DSICNVRT to convert each of them.

For Both Types of Migration:

Review the output of DSICNVRT to ensure that the command lists that it calls are consistent with your previously existing command lists. Make the necessary adjustments to take advantage of the extensions to message automation in Release 3.

Conversion Considerations for MVS/OCCF

If you are automating MVS messages using MVS/OCCF, pay extra attention to the conversion process. In MVS the job entry subsystems (JES2 and JES3) offset the actual text of a message by inserting a job identifier. If you coded your message automation for MVS system messages using an offset to read the message ID in the first column, instead of the MVS job ID, you have to convert manually instead of using DSICNVRT. In NetView Release 3 the program offsets the job identifier in the message for you. When the MVS system message crosses the NetView interface, the message ID starts in the first column. The job identifier is still available in the form of the function JOBNUM() in a REXX command list or the control variable &JOBNUM in a command list written in the NetView command list language. For more information about &JOBNUM, see "Message Processing Information" on page 84. Refer to "Message Processing Information" on page 53 for more information about JOBNUM().

Chapter 10. Service Point Command Service Commands

This chapter describes how to use the service point command service (SPCS) commands in command lists.

Service Point Command Service

The service point command service is a set of commands that supports and enhances the NetView program's control of service points, for example, the NetView/PC™. A service point application manages non-SNA devices, such as front-end line switches and multiplexers. You can send commands to the service point application to do problem determination for these devices.

There are four NetView SPCS commands that are used with NetView/PC for problem determination:

- LINKTEST — requests that the service point test a given link or link segment.
- LINKDATA — requests that the service point return device data for a given link or link segment.
- LINKPD — requests problem determination analysis from the service point on a given link or link segment.
- RUNCMD — sends service point application commands to the service point applications from NetView.

See *NetView Operation* for the syntax of the LINKTEST, LINKDATA, LINKPD, and RUNCMD commands.

The SPCS commands are long running commands that suspend the command list when they are executed. The command list resumes when the SPCS command is complete.

Note: The REXX WAIT instruction and the NetView command list language &WAIT control statement should not be used with the SPCS commands. Use message-driven command lists to trap messages generated from the SPCS commands, with the exception of:

- LINKPD messages DSI533I, DSI534I, DSI535I, DSI536I, and DSI582I. These five messages are set to values you can use in the form of control and parameter variables. See "LINKPD Results" on page 167 for more information on LINKPD results.
- Responses to RUNCMD with the CLISTVAR keyword. CLISTVAR causes the responses to be stored in variables. See "RUNCMD Results" on page 168 for more information on RUNCMD results.

™ NetView/PC is a trademark of International Business Machines Corporation.

Service Point Command Service Return Codes

After the command is completed, RC for command lists written in REXX or &RETCODE for command lists written in the NetView command list language contains one of the following values:

Code	Meaning
0	The command succeeded.
4	The command failed or CLISTVAR was specified with RUNCMD and no response was returned.
16	The command was cancelled by the CANCMD.
24	Some command list data was truncated.
28	The service point application returned more than 132 responses for the RUNCMD with the CLISTVAR keyword.

LINKDATA and LINKTEST Results

LINKDATA and LINKTEST can be used in command lists to manage service points, for example NetView/PC. The formats of these commands can be found in *NetView Operation*.

You can use the following LINKDATA and LINKTEST variable names in your command lists. Use the variable name without the ampersand for REXX command lists. Use the variable name with the ampersand for NetView command list language command lists. The italicized letters in the variable names will be replaced with the following values:

- *pp* — path number (01)
- *rr* — resource number (01-99)
- *ee* — entry number (01-99).

Note: Path number is always equal to '01' for LINKTEST and LINKDATA.

DSIPATHCNT or &DSIPATHCNT
Number of paths returned. It is always equal to '01' for LINK commands. The path count is the origin of the value of *pp* in the following variable names.

DSI*pp*RC or &DSI*pp*RC
Number of resources for path *pp*. The resource count is the origin of the value of *rr* in the following variable names.

DSI*pprr*EC or &DSI*pprr*EC
Number of entries for resource *rr* on path *pp*. The entry count is the origin of the value of *ee* in the following variable names.

DSI*pprr*RN or &DSI*pprr*RN
Name of resource *rr* on path *pp*

DSI*pprr*RT or &DSI*pprr*RT
Type of resource *rr* on path *pp*

DSI*pprree*DN or &DSI*pprree*DN
Name of data item *ee* for resource *rr* on path *pp*

DSI*pprree*DT or &DSI*pprree*DT
Type of data item *ee* for resource *rr* on path *pp*. Possible values are:

- BIT STRING
- CHARACTER

- DECIMAL
- HEXADECIMAL.

DSI*pprree*DV or &DSI*pprree*DV
Value of data item *ee* for resource *rr* on path *pp*.

LINKTEST Additional Variables

In addition, LINKTEST uses the following variables:

DSIREQUEST or &DSIREQUEST
Number of tests requested

DSIACTUAL or &DSIACTUAL
Actual number of tests executed

DSITESTTYPE or &DSITESTTYPE
Indication of the type of test data reported. Possible values are:

- BACKGROUND
- REQUESTED.

DSIRESULT or &DSIRESULT
Indication of the overall results of the test execution. Possible values are:

- PASSED
- FAILED
- INDETERMINATE.

LINKPD Results

Results from the LINKPD command are returned in messages that can be used in a command list to automate the recovery of resources controlled by a service point, for example, NetView/PC. LINKPD results can be used as the REXX functions MSGCNT(), MSGID(), MSGORIGN(), MSGSTR(), MSGTYP(), and MSGVAR(1) - MSGVAR(31), or the NetView command list language control variables &MSGCNT, &MSGID, &MSGORIGN, &MSGSTR, &MSGTYP, and the parameter variables &1 - &31. For more information about the REXX functions MSGORIGN(), MSGID(), MSGCNT(), MSGSTR(), and MSGTYP(), see "Message Processing Information" on page 53. For more information about MSGVAR(1) - MSGVAR(31), see "Functions Set by MSGREAD" on page 40. For more information about the NetView command list language control variables &MSGCNT, &MSGID, &MSGORIGN, &MSGSTR, and &MSGTYP see "Message Processing Information" on page 84. For more information about parameter variables used in command lists written in the NetView command list language, see "Parameter Variables" on page 77.

RUNCMD Results

If you use RUNCMD without the CLISTVAR keyword, responses from the service point application that performed the RUNCMD are sent to the network operator's terminal, and a return code is set (RC or &RETCODE). See "Service Point Command Service Return Codes" on page 166 for a description of the different return codes.

If you use RUNCMD with the CLISTVAR keyword, the command results in the following:

- A return code is set (RC or &RETCODE). See "Service Point Command Service Return Codes" on page 166 for a description of the different return codes.
- If the command completes with a return code of 0, 24, or 28, the following variables are set. Use the variable name without the ampersand for REXX command lists. Use the variable name with the ampersand for NetView command list language command lists.

  DSIRUNCNT or &DSIRUNCNT
  Contains the number of responses returned from the service point application. The variable has a value from 001 to 132.

  DSIRUN*xxx* or &DSIRUN*xxx*
  Contains the different responses from the service point application. The responses are numbered from 001 to 132.

Note: The responses from the service point must be character data and cannot be longer 255 characters.

Appendixes

Appendix A. REXX Command List Reference Summary

This appendix contains separate summary charts of the REXX instructions and functions provided by the NetView program. In each chart, the entries are listed in alphabetical order.

In the instruction summary chart, the instruction is followed by its operands, a brief description, and where to find more information in this book.

In the function table, the function is followed by its description and where to find more information in this book.

A complete list of all REXX instructions and functions can be found in *REXX Reference* or *REXX User's Guide.*

Notes:

1. In this book *REXX Reference* refers to *TSO/E REXX Reference* for MVS users or *VM/SP System Product Interpreter Reference* for VM users.
2. In this book *REXX User's Guide* refers to *TSO/E REXX User's Guide* for MVS users or *VM/SP System Product Interpreter User's Guide* for VM users.

Table 3. REXX Instruction Summary

Instruction	Operands	Description	Location
FLUSHQ	[**MESSAGES**]	Removes all trapped messages from the message queue.	See "REXX FLUSHQ Instruction" on page 42.
GLOBALV	**PUTT** *variable*[,...] **GETT** *variable*[,...]	Sets and retrieves the variables specified as task global variables.	See "REXX GLOBALV Instruction" on page 44.
GLOBALV	**PUTC** *variable*[,...] **GETC** *variable*[,...]	Sets and retrieves the variables specified as common global variables.	See "REXX GLOBALV Instruction" on page 44.
MSGREAD	None	Reads a trapped message from the message queue.	See "REXX MSGREAD Instruction" on page 40.
TRAP	[[**AND**] **SUPPRESS**|**DISPLAY**]] [**MORE**|**ONLY**] **MESSAGES** *token* [,...]	Defines the messages to be trapped.	See "REXX TRAP Instruction" on page 34.
TRAP	**NO MESSAGES**	Indicates that the list of messages to be trapped that was specified on a previous TRAP instruction should be removed.	See "REXX TRAP Instruction" on page 34.
WAIT	[*n* [**SECONDS**|**MINUTES**]] [**FOR** [**MESSAGES**]]	Causes a command list to temporarily suspend processing until a specific event occurs. You must code a time interval, MESSAGES, or both.	See "REXX WAIT Instruction" on page 36.
WAIT CONTINUE	None	Causes a command list to continue waiting before resuming processing. The options specified on the previous TRAP and WAIT instructions remain in effect for WAIT CONTINUE.	See "Continuing to Wait for Additional Messages" on page 38.

Table 4 (Page 1 of 3). REXX Function Summary

Function	Value	Location
APPLID()	The application program identifier for the task under which the command list is running (NetView domain ID appended with a 3-character alphanumeric value assigned by NetView).	See "Session Information" on page 51.
AREAID()	A one-letter (A-Z) identifier for the area on the console screen that displays the message.	See "Message Processing Information" on page 53.
COMPNAME()	The 16-byte name of the component that was running when the command list was initiated.	See "Command List Information" on page 53
DESC()	The system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.	See "Message Processing Information" on page 53.
EVENT()	The NetView event that satisfied the WAIT instruction.	See "Checking the Result of a WAIT Instruction" on page 38.
HCOPY()	The name of the hardcopy log printer started by the operator.	See "Terminal Information" on page 52.
HDRMTYPE()	The 1-character NetView message type of the message.	See "Message Processing Information" on page 53.
JOBNAME()	The 1- to 8-character MVS JOB name identifier.	See "Message Processing Information" on page 53.
JOBNUM()	The 8-character MVS JOB number identifier.	See "Message Processing Information" on page 53.
LINETYPE()	The multi-line write-to-operator (MLWTO) line type.	See "Message Processing Information" on page 53.
LU()	The logical unit name for this operator terminal.	See "Terminal Information" on page 52.
MCSFLAG()	The system message flags in a binary series of on (1) and off (0) codes.	See "Message Processing Information" on page 53.
MSGCNT()	The number of elements of text in the message string of the last message read by MSGREAD.	See "Message Processing Information" on page 53.
MSGID()	The message ID of the last message read by MSGREAD.	See "Message Processing Information" on page 53.

Table 4 (Page 2 of 3). REXX Function Summary

Function	**Value**	**Location**
MSGORIGN()	The domain where the last message read by MSGREAD originated.	See "Message Processing Information" on page 53.
MSGSTR()	The message text of the last message read by MSGREAD.	See "Message Processing Information" on page 53.
MSGTYP()	The system message type presented as three consecutive binary characters.	See "Message Processing Information" on page 53.
MSGVAR(*n*)	The value of each element of message text of the last message read by MSGREAD. The MSGVAR(*n*) functions can also be given values when a command list is invoked in the same way the &1 - &31 NetView command list language parameter variables can.	See "Functions Set by MSGREAD" on page 40.
NVCNT()	The number of NetView domains with which you can establish a cross-domain session.	See "Session Information" on page 51.
NVID(*n*)	The NetView domain identifier of a domain with which you can establish a cross-domain session.	See "Domain Information" on page 56.
NVSTAT(*name*)	Indicates whether you have an active session with a domain.	See "Domain Information" on page 56.
OPID()	The operator's ID.	See "Operator Information" on page 52.
OPSYSTEM()	A character string that indicates the operating system under which a command list is running.	See "Session Information" on page 51
PARMCNT()	The number of parameter variables that were entered when a command list was initiated.	See "Command List Information" on page 53.
REPLYID()	A three-character reply identifier for WTOR command replies.	See "Message Processing Information" on page 53.
ROUTCDE()	The system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.	See "Message Processing Information" on page 53.

Table 4 (Page 3 of 3). REXX Function Summary

Function	Value	Location
SESSID()	The ID of the TAF session that sent the message.	See "Message Processing Information" on page 53.
SMSGID()	The 8-character value that identifies a particular instance of a message.	See "Message Processing Information" on page 53.
SYSCONID()	The console number (in decimal) that is to receive a message.	See "Message Processing Information" on page 53.
SYSID()	The identifier of the MVS system that sent the message.	See "Message Processing Information" on page 53.
TASK()	The 3-character string PPT (primary POI task), OST (operator station task), or NNT (NetView-NetView task), depending on the task under which the command list is running. TASK() allows the same command list to run under any of these tasks, because the command list can test for the task type and process accordingly.	See "Session Information" on page 51.
VTAM()	A character string indicating the level of the access method used.	See "Session Information" on page 51.
WTOREPLY()	The reply sent by the operator in response to a WTOR command.	See "Message Processing Information" on page 53.

Appendix B. NetView Command List Language Reference Summary

This appendix contains separate summary charts of all control statements, control variables, and built-in functions used in the NetView command list language. In each chart, the entries are listed in alphabetical order.

In the built-in function and control statement charts, the function or statement is followed by its operands, a brief description, and where to find more information in this book.

In the control variable table, the variable is followed by its values and where to find more information in this book.

Table 5. Built-in Function Summary

Function	Operands	Description	Location
&CONCAT	variable variable constant constant variable constant constant variable	Joins the values of two variables or constants to form a new value.	See "&CONCAT Built-In Function" on page 100.
&LENGTH	variable|constant	Provides the number of characters in a variable or a constant.	See "&LENGTH Built-In Function" on page 100.
&NCCFID	number	Provides the identifier of a domain with which you can establish a cross-domain session.	See "&NCCFID Built-In Function" on page 101.
&NCCFSTAT	domain	Indicates whether you have an active session with a domain.	See "&NCCFSTAT Built-In Function" on page 102.
&SUBSTR	variable start [length]	Puts part of a variable into another variable.	See "&SUBSTR Built-In Function" on page 103.

Table 6 (Page 1 of 2). Control Statement Summary

Control Statement	**Operands**	**Description**	**Location**
&BEGWRITE	[**SUB**|**NOSUB**] [*-label*]	Writes a series of messages to the operator.	See "&BEGWRITE Control Statement" on page 95.
&CGLOBAL	*variable* [,...]	Defines the variables listed as common global variables.	See "Common Global Variables" on page 126.
&CONTROL	[**ALL**|**CMD**|**ERR**]	Indicates which command list statements are displayed to the operator.	See "&CONTROL Control Statement" on page 92.
&EXIT	[*number*]	Ends the command list.	See "&EXIT Control Statement" on page 109.
&GOTO	*-label*	Transfers control to another part of the command list.	See "&GOTO Control Statement" on page 109.
&IF	*comparison* &THEN *statement*	Tests a logical comparison and takes action based on the results. If the comparison is true, the &THEN clause is processed.	See "&IF Control Statement" on page 107.
&PAUSE	**NOINPUT**| **VARS** *variable* [...]| **STRING** *variable*	Halts the command list until the operator types in GO or RESET. The operator can enter variables in the command list by coding them as operands on the GO command.	See "&PAUSE Control Statement" on page 97.
&TGLOBAL	*variable* [,...]	Defines the variables listed as task global variables.	See "Task Global Variables" on page 124.
&WAIT	['*command*'] *event*=*-label* [,...]	Halts the command list until a specific message or group of messages is received.	See "Coding an &WAIT Control Statement" on page 112.

Table 6 (Page 2 of 2). Control Statement Summary

Control Statement	**Operands**	**Description**	**Location**
&WAIT	[**DISPLAY**|**SUPPRESS**]	Determines whether messages for the next &WAIT should be displayed to the operator.	See "Customizing the &WAIT Statement" on page 118.
&WAIT	[**ENDWAIT**|**CONTWAIT**]	Establishes whether the next &WAIT can wait for more than one event.	See "Customizing the &WAIT Statement" on page 118.
&WAIT	[**CONTINUE**]	Establishes whether the next &WAIT continues waiting after the event has been satisfied.	See "Customizing the &WAIT Statement" on page 118
&WRITE	*message text*	Sends a one-line message to the operator.	See "&WRITE Control Statement" on page 94.

Table 7 (Page 1 of 3). Control Variable Summary

Variable	Value	Location
&APPLID	The application program identifier for the task under which the command list is running (NetView domain ID appended with a 3-character alphanumeric value assigned by NetView).	See "Session Information" on page 82.
&AREAID	A one-letter (A-Z) identifier for the area on the console screen that displays the message.	See "Message Processing Information" on page 84.
&COMPNAME	The 16-byte name of the component that was running when the command list was initiated.	See "Command List Information" on page 83.
&DATE	The current date in the format *MM/DD/YY*.	See "Time and Date" on page 81.
&DESC	The system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.	See "Message Processing Information" on page 84.
&HCOPY	The name of the hard-copy log printer started by the operator.	See "Terminal Information" on page 83.
&HDRMTYPE	The 1-character NetView message type of the message.	See "Message Processing Information" on page 84.
&JOBNAME	The 1- to 8-character MVS JOB name identifier.	See "Message Processing Information" on page 84.
&JOBNUM	The 8-character MVS JOB number identifier.	See "Message Processing Information" on page 84.
&LINETYPE	The multi-line write-to-operator (MLWTO) line type.	See "Message Processing Information" on page 84.
&LU	The logical unit name for the operator terminal.	See "Terminal Information" on page 83.
&MCSFLAG	The system message flags in a binary series of on (1) and off (0) codes.	See "Message Processing Information" on page 84.
&MSGCNT	The number of elements of text in a message string.	See "Control and Parameter Variables Used with &WAIT" on page 116.

Table 7 (Page 2 of 3). Control Variable Summary

Variable	Value	Location
&MSGID	The message ID of the message most recently received by NetView.	See "Control and Parameter Variables Used with &WAIT" on page 116.
&MSGORIGIN	The domain from which the message was sent.	See "Control and Parameter Variables Used with &WAIT" on page 116.
&MSGSTR	The message text of the message most recently received by NetView.	See "Control and Parameter Variables Used with &WAIT" on page 116.
&MSGTYP	The system message type presented as three consecutive binary characters.	See "Message Processing Information" on page 84.
&NCCFCNT	The number of NetView domains with which the operator can establish a cross-domain session.	See "Session Information" on page 82.
&OPID	The operator's ID.	See "Operator Information" on page 83.
&OPSYSTEM	A character string that indicates the operating system under which a command list is running.	See "Session Information" on page 82.
&PARMCNT	The number of parameter variables that were entered when a command list was initiated.	See "Command List Information" on page 83.
&PARMSTR	The string of parameter variables used when the command list was called.	See "Command List Information" on page 83.
&REPLYID	A three-character reply identifier for WTOR command replies.	See "Message Processing Information" on page 84
&RETCODE	The return code set by either the most recent command processor or most recently activated or nested command list. The user can set &RETCODE with the &EXIT control statement to any positive value or to -1. &RETCODE can be tested to determine command list processing.	See "Command List Information" on page 83.
&ROUTCDE	The system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.	See "Message Processing Information" on page 84

Table 7 (Page 3 of 3). Control Variable Summary

Variable	Value	Location
&SESSID	The ID of the TAF session that sent the message.	See Chapter 9, "Message Automation" on page 135
&SMSGID	The 8-character value that identifies a particular instance of a message.	See "Message Processing Information" on page 84.
&SYSCONID	The console number (in decimal) that will receive a message.	See "Message Processing Information" on page 84.
&SYSID	The identifier of the MVS system that sent the message.	See "Message Processing Information" on page 84.
&TASK	The 3-character string PPT (primary POI task), OST (operator station task), or NNT (NetView-NetView task), depending on the task under which the command list is running. &TASK allows the same command list to run under any of these tasks, because the command list can test for the task type and process accordingly.	See "Session Information" on page 82.
&TIME	The CPU time in the format *hh:mm*.	See "Time and Date" on page 81.
&VIEWAID	The AID key that the operator used to enter panel input.	See "Panel Information" on page 87.
&VIEWCURCOL	The panel column where the cursor was positioned when the AID key was pressed.	See "Panel Information" on page 87.
&VIEWCURROW	The panel row where the cursor was positioned when the AID key was pressed.	See "Panel Information" on page 87.
&VTAM	A character string indicating the level of the access method used.	See "Session Information" on page 82.
&WTOREPLY	The reply sent by the operator in response to a WTOR command.	See "Message Processing Information" on page 84.
&1 - &31	The value of each element of message text of the last message received by NetView. &1-&31 can also be given values when a command list is invoked.	See "Control and Parameter Variables Used with &WAIT" on page 116.

Appendix C. Comparison of REXX and NetView Command List Language

This appendix provides a quick comparison between the features of REXX and the NetView command list language.

Comparison of REXX Instructions and NetView Command List Language Control Statements

Table 8 on page 186 shows the task performed by each control statement used in the NetView command list language and provides the equivalent REXX instruction. The table is in alphabetical sequence based on the name of the NetView command list language control statement. The last column of the table indicates whether the corresponding REXX instruction is a standard instruction provided by REXX or whether it is an instruction provided by the NetView program. If the instruction is a standard REXX instruction, an R appears in this column. If the instruction is provided by the NetView program, an N appears in this column. Instructions provided by the NetView program can only be used in conjunction with NetView. These instructions are not supported by the REXX interpreter and cannot be used in REXX EXECs executed in a non-NetView environment.

Table 8 (Page 1 of 2). Comparison of NetView Command List Language Control Statements and REXX Instructions

NetView Control Statement	Task	REXX Instruction	Task	Provided By
&BEGWRITE	Writes a series of messages to the operator.	None	Use the REXX SAY instruction or the NetView VIEW command.	R/N
&CGLOBAL	Defines the variables listed as common global variables.	GLOBALV	Sets and retrieves global variables.	N
&CONTROL	Indicates which command list statements are displayed to the operator.	TRACE	Indicates whether the result of each expression is displayed to the operator.	R
&EXIT	Ends the command list.	EXIT	Ends the command list.	R
&GOTO	Transfers control to another part of the command list.	SIGNAL	Transfers control to another part of the command list.	R
&IF	Tests a logical comparison and takes action based on the results. If the comparison is true, the &THEN clause is processed.	IF	Tests a logical comparison and takes action based on the results. If the comparison is true, the THEN clause is processed.	R
&PAUSE	Halts the command list until the operator types in GO or RESET.	PARSE EXTERNAL	Reads the next string from the terminal input buffer (system external event queue).	R
		PARSE PULL	Reads the next string from the program stack (system-provided data queue).	R
&TGLOBAL	Defines the variables listed as task global variables.	GLOBALV	Sets and retrieves global variables.	N

Table 8 (Page 2 of 2). Comparison of NetView Command List Language Control Statements and REXX Instructions

NetView Control Statement	Task	REXX Instruction	Task	Provided By
&WAIT	Halts the command list until a specific message or group of messages is received.	TRAP	Defines the messages to be trapped. Trapped messages can later be used in conjunction with the MSGREAD and WAIT instructions to control command processing.	N
		WAIT	Halts the command list until a specific event occurs. The event can be one or more messages, a specific period of time, or both. The EVENT() function can be used to determine the result of the WAIT command.	N
		WAIT CONTINUE	Causes the command list to wait for additional messages before resuming processing with the statement after the WAIT CONTINUE.	N
		MSGREAD	Causes trapped messages to be read.	N
		FLUSHQ	Removes all trapped messages from the message queue.	N
&WRITE	Sends a one-line message to the operator.	SAY	Sends a one-line or multiple-line message to the operator. The REXX SAY instruction can have a character string of any length; however, NetView will output only 32,728 characters at a time.	R

Comparison of REXX Functions and NetView Command List Language Control Variables

Table 9 shows the tasks performed by the various control variables used in the NetView command list language and the equivalent REXX functions. The last column of the table indicates if the REXX function is a standard function provided by REXX or if it is a function provided by the NetView program. If the function is provided by the NetView program, it can only be used in conjunction with NetView and is not supported by REXX.

Table 9 (Page 1 of 5). Comparison of REXX Functions and NetView Command List Language Control Variables

NetView Control Variable	**Task**	**REXX Function**	**Task**	**Provided By**
&APPLID	The application program identifier for the task under which the command list is running.	APPLID()	The application program identifier for the task under which the command list is running.	N
&AREAID	A one-letter (A-Z) identifier for the area on the console screen that displays the message.	AREAID()	A one-letter (A-Z) identifier for the area on the console screen that displays the message.	N
&COMPNAME	The 16-byte name of the component that was running when the command list was initiated.	COMPNAME()	The 16-byte name of the component that was running when the command list was initiated.	N
&CONCAT	Joins the values of two variables or constants to form a new value.	||	Joins the values of two variables or constants to form a new value.	R
&DATE	The current date in the format *mm/dd/yy*.	DATE()	The local date in various formats.	R
&DESC	The system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.	DESC()	The system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.	N
&HCOPY	The name of the hard-copy log printer started by the operator.	HCOPY()	The name of the hard-copy log printer started by the operator.	N

Table 9 (Page 2 of 5). Comparison of REXX Functions and NetView Command List Language Control Variables

NetView Control Variable	Task	REXX Function	Task	Provided By
&HDRMTYPE	The 1-character NetView message type of the message.	HDRMTYPE()	The 1-character NetView message type of the message.	N
&JOBNAME	The 1- to 8-character MVS job name identifier.	JOBNAME()	The 1- to 8-character MVS job name identifier.	N
&JOBNUM	The 8-character MVS job number identifier.	JOBNUM()	The 8-character MVS job number identifier.	N
&LENGTH	Determines how many characters are in a variable or a constant.	LENGTH	Determines how many characters are in a variable or a constant.	R
&LINETYPE	The multi-line write-to-operator (MLWTO) line type.	LINETYPE()	The multi-line write-to-operator (MLWTO) line type.	N
&LU	The logical unit name for this operator terminal.	LU()	The logical unit name for this operator terminal.	N
&MCSFLAG	The system message flags in a binary series of on (1) and off (0) codes.	MCSFLAG()	The system message flags in a binary series of on (1) and off (0) codes.	N
&MSGCNT	The number of elements of text in a message string.	MSGCNT()	The number of elements of text in the message string of the last message read by MSGREAD.	N
&MSGID	The message ID of the message most recently received by NetView.	MSGID()	The message ID of the last message read by MSGREAD.	N
&MSGORIGIN	The domain from which the message was sent.	MSGORIGN()	The domain where the last message read by MSGREAD originated.	N
&MSGSTR	The message text of the message most recently received by NetView.	MSGSTR()	The message text of the last message read by MSGREAD.	N

Table 9 (Page 3 of 5). Comparison of REXX Functions and NetView Command List Language Control Variables

NetView Control Variable	Task	REXX Function	Task	Provided By
&MSGTYP	The system message type presented as three consecutive binary characters.	MSGTYP()	The system message type presented as three consecutive binary characters.	N
&NCCFCNT	The number of NetView domains with which you can establish a cross-domain session.	NVCNT()	The number of NetView domains with which you can establish a cross-domain session.	N
&NCCFID	The NetView domain identifier of a domain with which you can establish a cross-domain session.	NVID(*n*)	The NetView domain identifier of a domain with which you can establish a cross-domain session.	N
&NCCFSTAT	Indicates whether you have an active session with a domain.	NVSTAT(*name*)	Indicates whether you have an active session with a domain.	N
&OPID	The operator's ID.	OPID()	The operator's ID.	N
&OPSYSTEM	A character string that indicates the operating system under which a command list is running.	OPSYSTEM()	A character string that indicates the operating system under which a command list is running.	N
&PARMCNT	The number of parameter variables that were entered when a command list was initiated.	PARMCNT()	The number of parameter variables that were entered when a command list was initiated.	N
&PARMSTR	The string of parameter values used when the command list was initiated.	ARG(1)	The string of arguments used when the command list was initiated.	R
&REPLYID	A three-character reply identifier for WTOR command replies.	REPLYID()	A three-character reply identifier for WTOR command replies.	N

Table 9 (Page 4 of 5). Comparison of REXX Functions and NetView Command List Language Control Variables

NetView Control Variable	Task	REXX Function	Task	Provided By
&RETCODE	The return code set by either the most recent command processor or most recently activated or nested command list.	RC	The return code set by the most recently executed host command or subcommand or the most recently activated or nested command list.	R
&ROUTCDE	The system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.	ROUTCDE()	The system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.	N
&SESSID	The ID of the TAF session that sent the message.	SESSID()	The ID of the TAF session that sent the message.	N
&SMSGID	The 8-character value that identifies a particular instance of a message.	SMSGID()	The 8-character value that identifies a particular instance of a message.	N
&SUBSTR	Puts part of a variable into another variable.	SUBSTR	Puts part of a variable into another variable.	R
&SYSCONID	The console number (in decimal) that will receive a message.	SYSCONID()	The console number (in decimal) that will receive a message.	N
&SYSID	The identifier of the MVS system that sent the message.	SYSID()	The identifier of the MVS system that sent the message.	N
&TASK	The 3-character string PPT (primary POI task), OST (operator station task), or NNT (NetView-NetView task), depending on the task under which the command list is running.	TASK()	The 3-character string PPT (primary POI task), OST (operator station task), or NNT (NetView-NetView task), depending on the task under which the command list is running.	N

Table 9 (Page 5 of 5). Comparison of REXX Functions and NetView Command List Language Control Variables

NetView Control Variable	Task	REXX Function	Task	Provided By
&TIME	The CPU time in the format *hh:mm*.	TIME()	The CPU time in various formats.	R
&VTAM	A character string indicating the level of the access method used.	VTAM()	A character string indicating the level of the access method used.	N
&WTOREPLY	The reply sent by the operator in response to a WTOR command.	WTOREPLY()	The reply sent by the operator in response to a WTOR command.	N
&1 - &31	The value of each element of message text of the last message received by NetView. These variables can also be given values when the command list is invoked.	MSGVAR(*n*)	The value of each element of message text of the last message read by MSGREAD. These variables can also be given values when the command list is invoked.	N

Commands Used in Command Lists

Following is a list of NetView commands that are for use in command lists. These commands can be used in command lists written either in REXX or the NetView command list language. When using the commands in a REXX command list, you should enclose the parts of the command that you do not want variable substitution to take place on in single quotes. The list shows where in the book the command is described:

DOM	See "DOM" on page 141.
GETMLINE	See "GETMLINE" on page 154.
GETMSIZE	See "GETMSIZE" on page 152.
GETMTYPE	See "GETMTYPE" on page 153.
MSGROUTE	See "Routing Messages from Command Lists" on page 142.
PARSEL2R	See "Parsing Variables with PARSEL2R" on page 144.
SDOMAIN (with QUIET option)	See "Using the SDOMAIN Command with the QUIET Option" on page 158.
WTO	See "WTO" on page 138.
WTOR	See "WTOR" on page 140.

Examples Comparing REXX and NetView Command List Language

This section contains examples of command lists. Each example is first shown written in the NetView command list language. Following the NetView command list language example is an example of a REXX command list that performs the same functions.

GREETING Example—NetView Command List Language

```
 CLIST
&CONTROL ERR
***********************************************************************
*                                                                     *
*  GREETING  - SHOW SIMPLE EXAMPLE OF WAITING AND TRAPPING            *
*              USING THE DATE COMMAND                                 *
*                                                                     *
*  NOTE: WHEN DATE IS ENTERED, THE FOLLOWING IS RETURNED:             *
*                                                                     *
*  CNM359I DATE : TIME = HH:MM         DATE = MM/DD/YY                *
***********************************************************************
&WAIT SUPPRESS
&WAIT 'DATE' CNM359I=-DATELAB,+
             *10=-TIMEOUTLAB, +
             *ERROR=-ERRORLAB,+
             *ENDWAIT=-GOLAB
-DATELAB
&HOUR = &SUBSTR  &5  1  2
&IF &HOUR >= 12 &THEN &GOTO -AFTER12
&WRITE GOOD MORNING
&GOTO -EXIT
-AFTER12
&IF &HOUR >= 18 &THEN &GOTO -AFTER18
&WRITE GOOD AFTERNOON
&GOTO -EXIT
-AFTER18
&WRITE GOOD EVENING
&GOTO -EXIT
-ERRORLAB
&WRITE 'ERROR OCCURED WAITING FOR DATE COMMAND RESPONSE'
&GOTO -EXIT
-TIMEOUTLAB
&WRITE 'NO MESSAGE RETURNED FROM DATE COMMAND'
&GOTO -EXIT
-GOLAB
-EXIT
&EXIT
```

Figure 124. GREETING Example—NetView Command List Language

GREETING Example—REXX

```
/************************************************************************/
/*                                                                      */
/*  GREETING  - SHOW SIMPLE EXAMPLE OF WAITING AND TRAPPING             */
/*              USING THE DATE COMMAND                                  */
/*                                                                      */
/*  NOTE: WHEN DATE IS ENTERED, THE FOLLOWING IS RETURNED:              */
/*                                                                      */
/*  CNM359I DATE : TIME = HH:MM        DATE = MM/DD/YY                  */
/************************************************************************/
'TRAP AND SUPPRESS ONLY MESSAGES CNM359I ' /* TRAP DATE MESSAGE         */
'DATE'                              /* ISSUE COMMAND                    */
'WAIT 10 SECONDS FOR MESSAGES'      /* WAIT FOR ANSWER                  */
SELECT                              /* RESULT IS BACK, PROCESS IT...    */
  WHEN (EVENT()='M') THEN           /* DID WE GET A MESSAGE?            */
    DO                              /* YES...                           */
      'MSGREAD'                     /* ... READ IT IN                   */
      HOUR=SUBSTR(MSGVAR(5),1,2)    /* ... PARSE OUT THE HOUR           */
       SELECT                       /* GIVE APPROPRIATE GREETING...     */
         WHEN (HOUR<12) THEN        /* ...BEFORE NOON?                  */
           SAY 'GOOD MORNING'
         WHEN (HOUR<18) THEN        /* ...BEFORE SIX?                   */
           SAY 'GOOD AFTERNOON'
         OTHERWISE                  /* ...MUST BE NIGHT                 */
           SAY 'GOOD EVENING'
       END                          /* OF SELECT                        */
    END                             /* OF DO                            */
  WHEN (EVENT()='E') THEN'          /* DID WE GET AN ERROR?             */
      SAY 'ERROR OCCURED WAITING FOR DATE COMMAND RESPONSE'
  WHEN (EVENT()='T') THEN           /* DID WE GET A TIMEOUT?            */
      SAY 'NO MESSAGE RETURNED FROM DATE COMMAND'
  OTHERWISE
END                                 /* OF SELECT                        */
```

Figure 125. GREETING Example—REXX

LISTVAR Example—NetView Command List Language

```
  CLIST
&CONTROL ERR
* SEE THE NETVIEW OPERATIONS MANUAL AND/OR ENTER HELP CLISTNAME
*   FOR A DESCRIPTION OF FUNCTION AND SYNTAX FOR THIS CLIST.
************************************************************************
*    (C) COPYRIGHT IBM CORP. 1986, 1987                                *
*                                                                      *
*    LAST CHANGE:    05/16/86  13:22:07       SSI=61361322             *
*                                                                      *
*    IEBCOPY   SELECT MEMBER=((CNME1006,LISTVAR,R))                    *
*                                                                      *
*    OUTPUTS:                                                          *
*        CNM353I.                                                      *
*                                                                      *
*    CNME1006 CHANGED ACTIVITY:                                        *
*    CHANGE CODE  DATE      DESCRIPTION                                *
*    -----------  --------  ------------------------------------------ *
*                                                                      *
************************************************************************
&CONTROL ERR
*************************************************************************
*     IF PARM IS A ? THEN GOTO HELP.  OTHERWISE SHOW VARIABLES AND      *
*     VALUES.
&IF .&1 EQ .? &THEN &GOTO -HELP
&IF .&1 NE . &THEN &GOTO -ERROR
&WRITE CNM353I LISTVAR : 'OPSYSTEM' = &OPSYSTEM
&WRITE CNM353I LISTVAR : 'VTAMLVL' = &VTAM
&WRITE CNM353I LISTVAR : 'APPLID' = &APPLID
&WRITE CNM353I LISTVAR : 'OPID' = &OPID
&WRITE CNM353I LISTVAR : 'LU' = &LU
&WRITE CNM353I LISTVAR : 'TASK' = &TASK
&WRITE CNM353I LISTVAR : 'DATE' = &DATE
&WRITE CNM353I LISTVAR : 'TIME' = &TIME
&WRITE CNM353I LISTVAR : 'NCCFCNT' = &NCCFCNT
&WRITE CNM353I LISTVAR : 'HCOPY' = &HCOPY
&IF .&VTAM = '.' &THEN &GOTO -MSG
&EXIT
*************************************************************************
-ERROR
MESSAGE 306E,LISTVAR,&1
&EXIT
*************************************************************************
-MSG
&WRITE CNM386I LISTVAR : VTAM IS NOT ACTIVE AT THIS TIME
&EXIT
*************************************************************************
-HELP
HELP LISTVAR
&EXIT
```

Figure 126. LISTVAR Example—NetView Command List Language

LISTVAR Example—REXX

```
/*********************************************************************/
/*                                                                   */
/* THE LISTVAR COMMAND LIST WRITTEN IN REXX                          */
/*                                                                   */
/*********************************************************************/
 SELECT
   WHEN MSGVAR(1)='?' THEN                 /* HELP REQUESTED ?        */
     'HELP LISTVAR'                        /* GO GET HELP             */
   WHEN MSGVAR(1)¬='' THEN                 /* ANY PARMS SPECIFIED ?   */
     'MESSAGE 306E,LISTVAR' MSGVAR(1)      /* NO PARMS ALLOWED        */
   OTHERWISE                               /* ALL OK, LIST OUT VARIABLES*/
     DO
       SAY "CNM353I LISTVAR : 'OPSYSTEM' = "OPSYSTEM()
       SAY "CNM353I LISTVAR : 'VTAMLVL' = "VTAM()
       SAY "CNM353I LISTVAR : 'APPLID' = "APPLID()
       SAY "CNM353I LISTVAR : 'OPID' = "OPID()
       SAY "CNM353I LISTVAR : 'LU' = "LU()
       SAY "CNM353I LISTVAR : 'TASK' = "TASK()
       SAY "CNM353I LISTVAR : 'DATE' = "DATE(USA)
       SAY "CNM353I LISTVAR : 'TIME' = "SUBSTR(TIME(),1,5)
       SAY "CNM353I LISTVAR : 'NCCFCNT' = "NCCFCNT
       SAY "CNM353I LISTVAR : 'HCOPY' = "HCOPY()
       IF VTAM() = '' THEN                 /* IS VTAM ACTIVE ?        */
         SAY 'CNM386I LISTVAR : VTAM IS NOT ACTIVE AT THIS TIME'
     END                                   /* OF THE LIST VARIABLES   */
   END                                     /* OF SELECT               */
 RETURN                                    /* RETURN TO CALLER        */
```

Figure 127. LISTVAR Example—REXX

BROWSE Example—NetView Command List Language

```
 CLIST
&CONTROL ERR
* SEE THE NETVIEW OPERATIONS MANUAL AND/OR ENTER HELP CLISTNAME
*   FOR A DESCRIPTION OF FUNCTION AND SYNTAX FOR THIS CLIST.
*********************************************************************
*    (C) COPYRIGHT IBM CORP. 1986, 1988                             *
*                                                                   *
*    LAST CHANGE:    03/31/88  15:58:51       SSI=60981558          *
*                                                                   *
*    IEBCOPY   SELECT MEMBER=((CNME5001,BROWSE,R))                  *
*                                                                   *
*    CNME5001 CHANGED ACTIVITY:                                     *
*    CHANGE CODE  DATE      DESCRIPTION                             *
*    -----------  --------  --------------------------------------- *
*    $P1=P053950,NV1R3,880331,MTS: JUST EXIT IF INVALID RETURN CODE *
*                                                                   *
*********************************************************************
**********************************************************************
*     IF FIRST PARM IS A ? THEN GOTO HELP.   OTHERWISE DISPLAY A
*     SCREEN TO THE USER IN BROWSE MODE.
&IF .&1 EQ .? &THEN &GOTO -HELP
&IF &PARMCNT NE 1 &THEN &GOTO -HELP
*     IF THE PARM IS NETLOGN THEN BROWSE THE NETLOG
&IF .&1 EQ .NETLOGA &THEN &GOTO -LOG
&IF .&1 EQ .NETLOGI &THEN &GOTO -LOG
&IF .&1 EQ .NETLOGS &THEN &GOTO -LOG
&IF .&1 EQ .NETLOGP &THEN &GOTO -LOG
VIEW 1 &1
&VIEWRC = &RETCODE
&IF &VIEWRC EQ 0 &THEN &EXIT &VIEWRC
&IF &VIEWRC EQ 4 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 8 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 12 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 16 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 24 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 28 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 32 &THEN &GOTO -SCODE
&IF &VIEWRC EQ 36 &THEN &GOTO -SCODE
&EXIT
-SCODE
SHOWCODE &VIEWRC &1
&EXIT &VIEWRC
-LOG
STATMON &1
&EXIT
-HELP
HELP BROWSE
&EXIT
```

Figure 128. BROWSE Example—NetView Command List Language

BROWSE Example—REXX

```
/***********************************************************************/
/*                                                                     */
/*   THE BROWSE EXAMPLE WRITTEN IN REXX                                */
/*                                                                     */
/***********************************************************************/
 IF  (MSGVAR(1)='?')|,                    /* HELP REQUESTED   OR       */
     (MSGVAR(2)¬='')|,                    /* 2 OR MORE PARMS  OR       */
     (MSGVAR(1) ='') THEN                 /* NO PARMS ENTERED          */
       'HELP BROWSE'                      /* GO ISSUE GET HELP         */
 ELSE
   IF MSGVAR(1)='NETLOGA'|,               /* ACTIVE       OR           */
      MSGVAR(1)='NETLOGP'|,               /* PRIMARY      OR           */
      MSGVAR(1)='NETLOGS'|,               /* SECONDARY    OR           */
      MSGVAR(1)='NETLOGI'  THEN           /* INACTIVE                  */
       'STATMON 'MSGVAR(1)                /* GO USE STATMON BROWSE     */
   ELSE
     DO
       'VIEW 1 'MSGVAR(1)                 /* GO VIEW THE MEMBER        */
       IF RC¬=0 THEN                      /* BAD RC?                   */
         'SHOWCODE ' RC  MSGVAR(1)        /* ISSUE SHOWCODE            */
     END
 RETURN
```

Figure 129. BROWSE Example—REXX

ACTLU Example—NetView Command List Languge

```
ACTLU CLIST
&CONTROL ERR
************************************************************************
*  ACTLU COMMAND LIST - NETVIEW COMMAND LIST LANGUAGE VERSION          *
*                                                                      *
*  FUNCTION : TO ACTIVATE A VTAM NODE.                                 *
*  INPUT    : 1 PARAMETER, THE NAME OF THE NODE.                       *
************************************************************************
&IF .&1 = . &THEN &GOTO -NULLPARM
&NODE = &1
-WAIT
&WAIT SUPPRESS
&WAIT 'V NET,ACT,ID=&NODE' *30=-TIMEOUT +
                           *ERROR=-ERROR +
                           IST061I=-IST061 +
                           IST093I=-IST093
-ERROR
&WRITE ERROR PROCESSING ACTIVATE COMMAND
&EXIT
-TIMEOUT
&WRITE NO RESPONSE TO ACTLU CLIST FOR &NODE
&EXIT
-IST061
&WRITE ==> LU UNKNOWN TO YOUR VTAM <==
&EXIT
-IST093
&WRITE ==> TERMINAL &1 NOW &2 <==
&EXIT
-NULLPARM
&WRITE PLEASE ENTER "GO NODENAME" OR "GO STOP" TO CONTINUE
&PAUSE VARS &P1
&IF &P1 = 'STOP' &THEN &EXIT
&NODE = &P1
&GOTO -WAIT
```

Figure 130. ACTLU Example—NetView Command List Language

ACTLU Example—REXX

```
/*  ACTLU COMMAND LIST - REXX VERSION                                 */
/*  FUNCTION : TO ACTIVATE A VTAM NODE.                               */
/*  INPUT    : 1 PARAMETER, THE NAME OF THE NODE.                     */
/*********************************************************************/
IF MSGVAR(1) = '' THEN                /* NO FIRST PARAMETER  ?        */
  DO                                  /*  THEN ISSUE REQUEST          */
    SAY 'PLEASE ENTER "GO NODENAME"',/* REQUEST NODENAME FROM USER    */
        'OR "GO STOP" TO CONTINUE'    /* OR, ALLOW USER TO STOP CLIST  */
    PARSE PULL NODE                   /* NODE = NODENAME OR STOP      */
  END                                 /*  THEN ISSUE REQUEST          */
ELSE                                  /* FIRST PARAMETER EXISTS       */
  NODE = MSGVAR(1)                    /* ASSUME IT IS A NODE NAME     */
                                      /* IF NODE='STOP' CLIST ENDS    */
IF NODE¬='STOP' THEN                  /* DID USER CHOOSE TO STOP ?    */
  DO                                  /* PROCESS NODENAME             */
    'TRAP AND SUPPRESS ONLY MESSAGES IST* ' /* TRAP ALL VTAM MSGS     */
    'V NET,ACT,ID='NODE               /* ISSUE VTAM ACTIVATE FOR NODE  */
    IF RC=0 THEN                      /* VALID NODE NAME ?            */
      DO                              /* YES, RETURN CODE = 0         */
        'WAIT 30 SECONDS FOR MESSAGES'   /* WAIT FOR 30 SECONDS       */
        IF EVENT()='M' THEN           /* OUT OF WAIT - IS THERE A MSG? */
          DO                          /* PROCESS TRAPPED MESSAGE      */
            'MSGREAD'                 /* READ IN 1ST MESSAGE          */
            DO WHILE (RC=0)           /* IF RC¬=0 THEN NO MORE MSGS   */
              SELECT                  /* DETERMINE WHICH MESSAGE HIT  */
                WHEN (MSGID() = 'IST061I')          /* NODE NOT FOUND  */
                  THEN SAY '==> LU UNKNOWN ',       /* INFORM USER     */
                           'TO YOUR VTAM <=='
                WHEN (MSGID() = 'IST093I')          /* NODE NOW ACTIVE */
                  THEN SAY '==> TERMINAL ',         /* INFORM USER     */
                       MSGVAR(1)' NOW ',
                       MSGVAR(2) '<=='
                OTHERWISE             /* IGNORE THE VTAM MESSAGE      */
                  'WAIT CONTINUE'     /* CONTINUE WAITING             */
              END                     /* OF SELECT FOR IST061I/IST093I */
              'MSGREAD'               /* READ IN THE NEXT MESSAGE     */
            END                       /* DO WHILE RC=0, LOOP BACK     */
          END                         /* PROCESS TRAPPED MESSAGE DO   */
                                      /* OUT OF DO WHILE, CHECK FOR
                                         ERROR OR TIMEOUT EVENTS      */
        SELECT                        /* CHECK RESULT OF THE WAIT     */
          WHEN (EVENT()='E') THEN     /* ERROR ENCOUNTERED ?          */
            SAY 'ERROR PROCESSING ', /* INFORM USER                   */
                'ACTIVATE COMMAND'
          WHEN (EVENT()='T') THEN     /* WAIT TIMEOUT ENCOUNTERED ?   */
            SAY 'NO RESPONSE TO ',    /* INFORM USER                  */
                'ACTLU CLIST FOR 'NODE
          OTHERWISE                   /* NO-OP                        */
        END                           /* OF SELECT FOR ERROR/TIMEOUT  */
    END                               /* IF RC=0  (VALID NODENAME)    */
  END                                 /* IF NODE¬='STOP' PROCESSING   */
```

Figure 131. ACTLU Example—REXX

GETCG Example—NetView Command List Language

```
GETCG CLIST
&CONTROL ERR
******************************************************************
* GETCG COMMAND LIST - NETVIEW COMMAND LIST LANGUAGE VERSION    *
*                                                               *
* GETCG COMMAND LIST GETS THE VALUE OF A COMMON GLOBAL          *
* VARIABLE AND DISPLAYS IT TO THE REQUESTING TASK               *
******************************************************************
&CGLOBAL &1
MESSAGE 309I GETCG COMMON GLOBAL VARIABLE &1 HAS VALUE &&1
&EXIT
```

Figure 132. GETCG Example—NetView Command List Language

GETCG Example—REXX

```
/******************************************************************/
/* GETCG COMMAND LIST - REXX VERSION                              */
/*                                                                */
/* GETCG COMMAND LIST GETS THE VALUE OF A COMMON GLOBAL           */
/* VARIABLE AND DISPLAYS IT TO THE REQUESTING TASK                */
/******************************************************************/
 TRACE E
GLOBALV GETC  MSGVAR(1)
'MESSAGE 309I GETCG COMMON GLOBAL VARIABLE' MSGVAR(1) ,
    'HAS VALUE ' VALUE(MSGVAR(1))
 EXIT
```

Figure 133. GETCG Example—REXX

PPTUPDAT Example—NetView Command List Language

```
PPTUPDAT CLIST
&CONTROL ERR
**********************************************************************
* THIS COMMAND LIST SETS ANY COMMON GLOBAL VARIABLE ON THE PPT.
* USE THE EXCMD COMMAND TO RUN IT ON THE PPT.
* EXAMPLE:  EXCMD PPT PPTUPDAT AAAAAAAA BBBBBBBB.....
*    WHERE AAAAAAAA IS THE NAME OF THE COMMON GLOBAL VARIABLE.
*    WHERE BBBBBBBB..... IS THE VALUE FOR THE COMMON GLOBAL VARIABLE
**********************************************************************
&CGLOBAL &1
&IF .&TASK NE .PPT &THEN &GOTO -ERROR
&&1 = &2
MESSAGE 309I PPTUPDAT COMMON GLOBAL VARIABLE UPDATE COMPLETE
&EXIT
-ERROR
MESSAGE 309I PPTUPDAT CANNOT UPDATE COMMON GLOBAL VARIABLES +
FROM NON-PPT TASK
&EXIT
```

Figure 134. PPTUPDAT Example—NetView Command List Language

PPTUPDAT Example—REXX

```
/*********************************************************************/
/* THIS COMMAND LIST SETS ANY COMMON GLOBAL VARIABLE ON THE PPT.     */
/* USE THE EXCMD COMMAND TO RUN IT ON THE PPT.                       */
/* EXAMPLE: EXCMD PPT PPTUPDAT AAAAAAAA BBBBBBBB.....                */
/*  WHERE AAAAAAAA IS THE NAME OF THE COMMON GLOBAL VARIABLE.        */
/*  WHERE BBBBBBBB...... IS THE VALUE FOR THE COMMON GLOBAL VARIABLE */
/*********************************************************************/
TRACE E
/* IF TASK IS NOT THE PPT, ISSUE AN ERROR MESSAGE TO USER            */
IF TASK() ¬= 'PPT' THEN
  'MESSAGE 309I PPTUPDAT CANNOT UPDATE COMMON GLOBAL VARIABLES ',
          'FROM NON-PPT TASK'
ELSE
  DO                                        /* TASK IS PPT           */
    INTERPRET VALUE(MSGVAR(1)) '=' MSGVAR(2) /* BUILD &&1 = &2 CMD   */
    GLOBALV PUTC  MSGVAR(1)                 /* PUT COMMON GLOBAL VAR*/
    'MESSAGE 309I PPTUPDRX COMMON GLOBAL VARIABLE UPDATE COMPLETE'
  END                                       /* TASK IS PPT           */
EXIT
```

Figure 135. PPTUPDAT Example—REXX

ACTAPPLS Example—NetView Command List Language

```
  CLIST
&CONTROL ERR
*********************************************************************
*                                                                   *
* ACTAPPLS - NETVIEW COMMAND LIST LANGUAGE VERSION                  *
*                                                                   *
* DISPLAY ONLY THE ACTIVE APPLS                                     *
*                                                                   *
*********************************************************************
* WRITE THE HEADER
&WRITE ACTIVE APPLICATIONS:
&WRITE ====================
* WAIT ON THE DISPLAY COMMAND
&WAIT CONTWAIT SUPPRESS
&WAIT 'D NET,APPLS',IST350I=-FIRST,*=-ALLELSE
* ALL NON-INFORMATIONAL MESSAGES GO HERE
-ALLELSE
&WAIT CONTINUE
* THE MULTILINE WTO WITH THE APPL INFORMATION COMES HERE
-FIRST
* DETERMINE THE NUMBER OF LINES
GETMSIZE NUMLINES
*INITIALIZE LINE NUMBER COUNTER, AND TOTAL ACTIVE APPLS
&I = 0
&TOTALACT = 0
* TOP OF THE MLWTO LOOP
-LOOP
* SET NUMBER OF ACTIVE APPLS FOUND ON THIS LINE TO 0
&NUMACT = 0
* IF WE HAVE PARSED ALL THE LINES WE ARE DONE
&IF &NUMLINES = &I &THEN &GOTO -ALLDONE
* DETERMINE THE NUMBER OF LINES IN THE MLWTO
GETMLINE LINE &I
* PARSE OUT THE LINE, A1 A2 A3 ARE APPL NAMES, S1 S2 S3 ARE STATUS
PARSEL2R LINE MSG A1 S1 A2 S2 A3 S3
* IF THERE IS NO STATUS, DONE WITH THIS LINE
&IF &S1. = . &THEN &GOTO -NOCHECK
* CHECK TO SEE IF THE STATUS OF APPL 1 IS ACTIVE
&IF &S1 = 'ACTIV' &THEN &GOTO -A1ACT
&IF &S1 = ' ACTIV' &THEN &GOTO -A1ACT
&IF &S1 = '  ACTIV' &THEN &GOTO -A1ACT
* APPL1 IS NOT ACTIVE, BLANK IT OUT
&S1 = ''
&A1 = ''
* CHECK THE NEXT APPL
&GOTO -CHECKA2
* THIS APPL IS ACTIVE
-A1ACT
* BUMP NUMBER ACTIVE COUNT
&NUMACT = &NUMACT + 1
```

Figure 136 (Part 1 of 2). ACTAPPLS Example—NetView Command List Language

```
-CHECKA2
* CHECK TO SEE IF THE STATUS OF APPL 2 IS ACTIVE
&IF &S2. = . &THEN &GOTO -CHECKA3
&IF &S2 = 'ACTIV' &THEN &GOTO -A2ACT
&IF &S2 = ' ACTIV' &THEN &GOTO -A2ACT
&IF &S2 = '  ACTIV' &THEN &GOTO -A2ACT
* APPL2 IS NOT ACTIVE, BLANK IT OUT
&S2 = ''
&A2 = ''
&GOTO -CHECKA3
* THIS APPL IS ACTIVE
-A2ACT
* BUMP NUMBER ACTIVE COUNT
&NUMACT = &NUMACT + 1
-CHECKA3
&IF &S3. = . &THEN &GOTO -NOCHECK
&IF &S3 = 'ACTIV' &THEN &GOTO -A3ACT
&IF &S3 = ' ACTIV' &THEN &GOTO -A3ACT
&IF &S3 = '  ACTIV' &THEN &GOTO -A3ACT
* APPL3 IS NOT ACTIVE, BLANK IT OUT
&S3 = ''
&A3 = ''
&GOTO -NOCHECK
* THIS APPL IS ACTIVE
-A3ACT
* BUMP NUMBER ACTIVE COUNT
&NUMACT = &NUMACT + 1
-NOCHECK
* ENABLE THE DISPLAYING
&WAIT CONTWAIT DISPLAY
* ANY ACTIVE ON THIS LINE ?
&IF &NUMACT = 0 &THEN &GOTO -NOWRITE
* DISPLAY THE ACTIVE APPLICATIONS
&WRITE &A1 &A2 &A3
-NOWRITE
* BUMP THE LINE COUNTER
&I = &I + 1
* BUMP THE TOTAL ACTIVE COUNTER
&TOTALACT = &TOTALACT + &NUMACT
* GO PROCESS THE NEXT LINE
&GOTO -LOOP
* THE PARSING IS DONE
-ALLDONE
* DISPLAY THE NUMBER OF ACTIVE APPLICATIONS
&WRITE NUMBER OF APPLICATIONS ACTIVE: &TOTALACT
GO
&EXIT
```

Figure 136 (Part 2 of 2). ACTAPPLS Example—NetView Command List Language

ACTAPPLS Example—REXX

```
/**********************************************************************/
/*                                                                    */
/* APPLRX : DISPLAY ONLY THE ACTIVE APPLS                             */
/*                                                                    */
/*      THIS COMMAND LIST PROCESSES THE FOLLOWING VTAM MESSAGES       */
/*          --------------------------------------------------        */
/*            IST350I DISPLAY TYPE = APPL MAJ NODES/NAMES             */
/*            IST089I _______ TYPE=APPL SEGMENT   ,   ACTIV           */
/*            IST360I APPLICATIONS:                                   */
/*            IST080I APPL1 STATUS1 APPL2 STATUS2 APPL3 STATUS3       */
/*                                                                    */
/*          ALL ARE IGNORED, EXCEPT FOR IST080I WHERE THE APPLS &     */
/*            THEIR RESPECTIVE STATUS ARE DISPLAYED (IF ACTIVE)       */
/*                                                                    */
/**********************************************************************/
TRACE E
/* WRITE THE HEADER */
'CLEAR'                                           /* CLEAR THE SCREEN */
SAY '    ACTIVE APPLICATIONS:'                    /* 1ST HEADER MSG   */
SAY '==============================='             /* 2ND HEADER MSG   */
'TRAP SUPPRESS ONLY MESSAGES IST097I IST350I'     /* TRAP HEADER &    */
                                                  /* DATA MESSAGES    */
'D NET,APPLS'                                     /* ISSUE D NET CMD  */
'WAIT 5 SECONDS FOR MESSAGES'                     /* WAIT FOR RESP.   */
'MSGREAD'                                         /* READ 1ST MESSAGE */
DO WHILE (MSGID() ¬= 'IST350I')                   /* IST350I ?        */
  'WAIT CONTINUE'                                 /* NO- CONT. WAIT   */
  'MSGREAD'                                       /* READ NEXT MSG    */
END                                               /* END IST350I      */
/* AT THIS POINT AN IST350I MLWTO BUFFER MSG HAS BEEN RECEIVED        */
NUMACT = 0                                        /* INIT. TOTAL CNTR */
OUTLINE=' '                                       /* CLEAR OUTLINE    */
GETMSIZE NUMLINES                                 /* GET NUM LINES    */
                                                  /* IN MLWTO BUFFER  */
/*                                                                    */
DO I=1 TO NUMLINES                                /* LOOP THRU BUFFER */
  'GETMLINE LINE' I                               /* GET MLWTO LINE   */
  'PARSEL2R LINE',                                /* PARSE LINE INTO  */
    'MSG',                                        /* MESSAGE ID       */
    'A1 S1',                                      /* 1ST APPL/STATUS  */
    'A2 S2',                                      /* 2ND APPL/STATUS  */
    'A3 S3'                                       /* 3RD APPL/STATUS  */
  IF MSG ='IST080I' THEN                          /* IST080I?         */
  DO                                              /* YES --           */
    IF LEFT(S1,8)= 'ACTIV' THEN                   /* APPL1 ACTIVE ?   */
      CALL ADDACT(A1)                             /* ADD APPL A1      */
    IF LEFT(S2,8)= 'ACTIV' THEN                   /* APPL2 ACTIVE ?   */
      CALL ADDACT(A2)                             /* ADD APPL A2      */
    IF LEFT(S3,8)= 'ACTIV' THEN                   /* APPL3 ACTIVE ?   */
      CALL ADDACT(A3)                             /* ADD APPL A3      */
  END                                             /* IST080I PROCESS  */
END                                               /* LOOP THRU BUFFER */
```

Figure 137 (Part 1 of 2). ACTAPPLS Example—REXX

```
IF NUMACT//3¬=0 THEN                                    /* NEED ANOTHER SAY?*/
  SAY OUTLINE                                           /* PUT LINE OUT     */
SAY '   >==== END OF DISPLAY ====<'                     /* EOD MESSAGE      */
SAY 'NUMBER OF APPLICATIONS ACTIVE: ' NUMACT            /* STATUS MESSAGE   */
EXIT
ADDACT:
/* ADD THE APPL NAME GIVEN TO THE OUTPUT LIST.  IF THERE ARE EXACTLY
    FIVE ENTRIES, THEN PUT OUT THE LINE                                 */
ARG APPL
OUTLINE=OUTLINE7||'  '||LEFT(APPL,8)
NUMACT=NUMACT+1
IF NUMACT//3=0 THEN
  DO
    SAY OUTLINE
    OUTLINE=' '
  END
RETURN
```

Figure 137 (Part 2 of 2). ACTAPPLS Example—REXX

Appendix D. Converting Command Lists Written in the NetView Command List Language to REXX

The NetView program provides a sample REXX EXEC that you can use to convert command lists written in the NetView command list language to REXX. The EXEC is shipped as NetView sample CNMS8001. You can rename the sample, but this section assumes you use the CNMS8001 name.

You can use CNMS8001 to convert a command list written in the NetView command list language list to a bilingual command list or a REXX-only command list. The bilingual command list contains both the existing NetView command list language version of the command list and the new REXX version. During the conversion, CNMS8001 reads each line of the NetView command list language version of the command list and writes the equivalent line in the REXX version. The records in the bilingual command list and the REXX-only command list are up to 80 characters in length and do not have sequence numbers.

There are some NetView command list language conditions that CNMS8001 cannot convert and there are other conditions that might not be converted correctly. When these conditions occur, CNMS8001 writes messages to a warning file to notify you of any conversion problems. See "Conditions CNMS8001 Cannot Convert" on page 213 and "Conditions CNMS8001 Might Not Convert Correctly" on page 213 for information on conversion restrictions.

CNMS8001 can be executed on the following operating systems:

- VM
- TSO/E running under MVS/XA.

Notes:

1. CNMS8001 cannot run under the NetView environment.
2. To run CNMS8001, the REXX interpreter must be installed.

Executing CNMS8001 Command List on TSO/E

You can execute CNMS8001 on a TSO/E system running under MVS/XA in one of two ways. The method you use depends on whether your SYS1.CNMSAMP data set is allocated to the SYSPROC or SYSEXEC DD statement for the TSO/E session. The CNMS8001 EXEC is contained in the SYS1.CNMSAMP data set.

Figure 138 shows the syntax for running CNMS8001 if SYS1.CNMSAMP is not allocated to SYSPROC or SYSEXEC. Figure 139 on page 211 shows the syntax for running CNMS8001 if SYS1.CNMSAMP is allocated to SYSPROC or SYSEXEC.

```
EXEC  'SYS1.CNMSAMP(CNMS8001)' '?'|'''inputds''
      ''outputds'' ''warnds'' [(NOSUPP|(SUPP]'
```

Figure 138. Syntax to Run CNMS8001 when SYS1.CNMSAMP is not Allocated to SYSPROC or SYSEXEC

?
indicates that the CNMS8001 help screens should be displayed.

inputds
specifies the MVS data set name of the NetView command list language command list being converted.

outputds
specifies the MVS data set name that you want to contain the converted command list.

warnds
specifies the MVS data set name that you want to contain the warning file created during the conversion.

(NOSUPP|(SUPP

(NOSUPP this optional parameter specifies that you want *outputds* to be a bilingual command list. If you do not specify (NOSUPP or (SUPP, (NOSUPP is the default.

(SUPP this optional parameter specifies that you want *outputds* to be a REXX only command list.

Notes:

1. The string of *inputds, outputds, warnds*, and optionally the (NOSUPP or (SUPP parameter are enclosed in a set of single quotes. Within that set of single quotes, each individual data set name is enclosed within two more sets of single quotes. If a data set name is not enclosed in two sets of single quotes, the high level qualifier for that data set name defaults to the user ID.

   For example, if the following command was entered by OPER1:

   ```
   EXEC 'SYS1.CNMSAMP(CNMS8001)' '''USER.CLIST(CLIST1)''
   ''USER.CLIST1(REXX1)'' CLIST(WARN1)'
   ```

   CNMS8001 would convert the NetView command list language command list in USER.CLIST(CLIST1) to a bilingual command list and place the bilingual command list in USER.CLIST(REXX1). Any warning or error messages would be placed in OPER1.CLIST(WARN1). The high level identifier for CLIST(WARN1) defaults to OPER1, because the data set name is not enclosed in two sets of single quotes.

2. The members for the converted command list and the warning file cannot be in the same data set. The data sets for the converted command list and the warning file cannot be allocated by another job (for example, if NetView is running you cannot put the output into the allocated NetView command list data sets).

Figure 139 shows the syntax for running CNMS8001 if SYS1.CNMSAMP is allocated to SYSPROC or SYSEXEC.

CNMS8001 ?|*'inputds'* *'outputds'* *'warnds'* **[(NOSUPP|(SUPP]**

Figure 139. Syntax to Run CNMS8001 when SYS1.CNMSAMP is Allocated to SYSPROC or SYSEXEC

?
: indicates that the CNMS8001 help screens should be displayed.

inputds
: specifies the MVS data set name of the NetView command list language command list being converted.

outputds
: specifies the MVS data set name that you want to contain the converted command list.

warnds
: specifies the MVS data set name of the warning file created during the conversion.

(NOSUPP|(SUPP

(NOSUPP	this optional parameter specifies that you want *outputds* to be a bilingual command list. If you do not specify (NOSUPP or (SUPP, (NOSUPP is the default.
(SUPP	this optional parameter specifies that you want *outputds* to be a REXX only command list.

Notes:

1. Each individual data set name is enclosed within a set of single quotes. If a data set name is not enclosed in single quotes, the high level qualifier for that data set name defaults to the user ID.

   For example, if the following command was entered by OPER1:

   ```
   CNMS8001 'USER.CLIST(CLIST1)' 'USER.CLIST(REXX1)' CLIST1(WARN1) (SUPP
   ```

   CNMS8001 would convert the NetView command list language command list in USER.CLIST(CLIST1) to a REXX command list and place the REXX command list in USER.CLIST(REXX1). Any warning or error messages would be placed in OPER1.CLIST(WARN1). The high level identifier for CLIST(WARN1) defaults to OPER1, because the data set name is not enclosed in single quotes.

2. The members for the converted command list and the warning file cannot be in the same data set. The data sets for the converted command list and the warning file cannot be allocated by another job (for example, if NetView is running you cannot put the output into the allocated NetView command list data sets).

Executing CNMS8001 Command List on VM Operating System

Figure 140 shows the syntax of the statement used to run the CNMS8001 command list on the VM operating system.

```
CNMS8001  ?|inputfn [inputft [inputfm]]
          [outputfn outputft outputfm
          warnfn warnft warnfm]
          [(NOSUPP|(SUPP]
```

Figure 140. Syntax to Run CNMS8001 Command List on VM Operating System

?
: indicates that the CNMS8001 help screens should be displayed.

inputfn
: specifies the VM file name of the NetView command list language command list being converted.

inputft
: specifies the VM file type of the NetView command list language command list being converted. If you do not code a file type or you code an asterisk (*), the file type defaults to CLIST.

inputfm
: specifies the VM file mode of the NetView command list language command list being converted. If you do not code a file mode or you code an asterisk (*), the file mode defaults to A.

Note: If you want to change any of the default file specifications for the output file or the warning file, you must specify a value or asterisk (*) for each of the following six operands:

outputfn
: specifies the VM file name of the command list created during the conversion. If you use the default file specifications for the output and warning files or you code an asterisk (*) for this parameter, the file name defaults to the name specified in *inputfn*.

outputft
: specifies the VM file type of the command list created during the conversion. If you use the default file specifications for the output and warning files or you code an asterisk (*) for this parameter, the file type defaults to EXEC.

outputfm
: specifies the VM file mode of the command list created during the conversion. If you use the default file specifications for the output and warning files or you code an asterisk (*) for this parameter, the file mode defaults to A.

warnfn
: specifies the VM file name of the warning file created during the conversion. If you use the default file specifications for the output and warning files or you code an asterisk (*) for this parameter, the file name defaults to the name specified in *inputfn*.

warnft
: specifies the VM file type of the warning file created during the conversion. If you use the default file specifications for the output and warning files or you code an asterisk (*) for this parameter, the file type defaults to WARNING.

warnfm

specifies the VM file mode of the warning file created during the conversion. If you use the default file specifications for the output and warning files or you code an asterisk (*) for this parameter, the file mode defaults to A.

(NOSUPP|(SUPP

(NOSUPP	this optional parameter specifies that you want *outputfn* to be a bilingual command list. If you do not specify (NOSUPP or (SUPP, (NOSUPP is the default.
(SUPP	this optional parameter specifies that you want *outputfn* to be a REXX only command list.

Conditions CNMS8001 Cannot Convert

There are some conditions that CNMS8001 cannot convert. When these conditions occur, the line in the NetView command list language command list is not converted. Each unconverted line is highlighted in the output file with three exclamation marks (!!!) at the beginning and the end of the line. A message containing the output file name and the line number of each line that could not be converted is written in the warning file.

The conditions that cannot be converted are:

- &BEGWRITE control statements having a label that is a variable.
- NetView command list language variables that CNMS8001 converts to REXX functions and that are objects of assignment statements.

Conditions CNMS8001 Might Not Convert Correctly

There are some conditions that CNMS8001 might not be able to convert correctly. When these conditions occur, messages are written in the warning file that instruct you to check the output file to verify that the conversion was done correctly. In most cases, the messages list the line numbers of the lines that might not have been converted correctly.

Some of the conditions that might not convert correctly are:

- &TGLOBAL and &CGLOBAL commands

  A list of the variables specified on &CGLOBAL or &TGLOBAL commands is provided in the warning file. Check the REXX portion of the output file to verify that these variables are assigned correctly.

- &WAIT commands

  The &WAIT commands are converted, and messages are written to the warning file to inform you that various &WAIT commands were converted. Check the REXX portion of the output file to verify that the new TRAP, WAIT, and MSGREAD commands can perform the desired operations.

- Parentheses and logical operators that are in a string.

  These lines are converted, and messages are written to the warning file that contain the line numbers of the converted lines. Check the REXX portion of the output file to ensure that the parentheses and logical operators were converted correctly.

- Strings that contain a single quote followed by the letter X

  These lines are converted, but, in some cases, the REXX interpreter perceives this condition as hexadecimal. Messages containing the line numbers of any lines having strings that contain a single quote followed by the letter X are written in the warning file. Check the REXX portion of the output file to verify that these strings were not interpreted as hexadecimal.

- Lines that contain an odd number of quotes in a string

  These lines are converted, but quote placement may not be correct. Messages are written to the warning file that list the line number of each line containing an odd number of quotes. Check the REXX portion of the output file to verify that the quotes were placed correctly.

- &A&B type variables

  CNMS8001 converts these variables. However, if the value that is put into the &B portion of the variable when the command list is run contains invalid REXX symbols, the REXX command list created by CNMS8001 will not execute.

Another condition that might cause problems when you run a converted command list is the default initial value of variables. No messages are written to the warning file for this condition. If a REXX variable is not assigned an initial value, its value defaults to the name of the variable. If a NetView command list language variable is not assigned an initial value, its value defaults to null. Therefore, if the logic of a command list written in the NetView command list language depends on a variable having a default initial value of null, the REXX version of the command list created by CNMS8001 will not run correctly.

Also, CNMS8001 assumes that the converted command list will always be called as a command and not as a subroutine or function. Therefore, the NetView command list language &1 - &31 variables are always converted to the REXX MSGVAR(1) - MSGVR(31) functions and not ARG(1) - ARG(31).

Improving the Performance of Converted Command Lists

When the CNMS8001 conversion tool converts a command list, it does not attempt to optimize the performance of the REXX command list it creates. REXX has many more features and functions than the NetView command list language. Because CNMS8001 only converts each line in the NetView command list language command list to its REXX equivalent, you can still enhance the performance of the REXX command list by manually adding additional REXX features and functions. After fixing those conditions that CNMS8001 could not convert or that might not have been converted correctly, go back and review the REXX command list to determine if its performance could be improved.

Some of the things you can do to improve the performance of a converted command list are:

- Improve the program structure by using REXX instructions such as CALL, DO, IF, and SELECT.
- Use REXX arithmetic operators that are not available in the NetView command list language. For example, multiply (*) and divide (/).
- Use REXX functions that have no NetView command list language equivalent. For example, REVERSE(), POS(), DATATYPE(), WORDS(), and many others.

- Reduce the use of the REXX functions provided by NetView. Using these functions is inefficient in terms of performance. If the same function, provided by NetView, is used several times in the command list without a change in value, use the function once to set a local variable to the value of the function. After setting the REXX function provided by NetView to a local variable, use the local variable in place of the function. If the value of the function changes during execution of the command list, you need to use the function each time to access its current value.

For complete information about the features and functions of REXX, see *REXX Reference* or *REXX User's Guide*.

Example of a Converted Command List

The command list in Figure 141 on page 216 is an example of a bilingual command list created using CNMS8001. The NetView command list language portion of the command list was used as input to CNMS8001.

```
/* CLIST TO BE CONVERTED
  CLIST
&CONTROL ERR
* SEE THE NETVIEW OPERATION MANUAL AND/OR ENTER HELP CLISTNAME
*   FOR A DESCRIPTION OF FUNCTION AND SYNTAX FOR THIS CLIST.
**********************************************************************
*    (C) COPYRIGHT IBM CORP. 1986, 1987                              *
*                                                                    *
*    LAST CHANGE:    04/08/86  15:41:52       SSI=60981541           *
*                                                                    *
*    IEBCOPY   SELECT MEMBER=((CNME1007,LSESS,R))                    *
*                                                                    *
*    OUTPUTS:                                                        *
*        ONE OF:                                                     *
*            LISTSESS                                                *
*            LISTSESS OPCTL                                          *
*            LISTSESS FLSCN                                          *
*            LISTSESS APPLID=&2                                      *
*            LISTSESS SRCLU=&2                                       *
*                                                                    *
*    CNME1007 CHANGED ACTIVITY:                                      *
*    CHANGE CODE  DATE      DESCRIPTION                              *
*    -----------  --------  ---------------------------------------- *
*                                                                    *
**********************************************************************
&P1 = &1
&P2 = &2
&IF .&P1 EQ .? &THEN &GOTO -HELP
&IF .&P1 EQ . &THEN &GOTO -HELP
&IF .&P1 EQ .ALL &THEN &GOTO -LALL
&IF .&P1 EQ .O &THEN &GOTO -OPCT
&IF .&P1 EQ .OPCTL &THEN &GOTO -OPCT
&IF .&P1 EQ .F &THEN &GOTO -FLSC
&IF .&P1 EQ .FLSCN &THEN &GOTO -FLSC
&IF .&P1 EQ .A &THEN &GOTO -APPL
&IF .&P1 EQ .S &THEN &GOTO -SRCL
*     INVALID PARAMETER
MESSAGE 306E,LSESS,&1
&EXIT
```

Figure 141 (Part 1 of 4). Bilingual Command List Created by CNMS8001

```
************************************************************************
-LALL
*    COMMAND ENTERED: LSESS ALL
&CONTROL CMD
LISTSESS
&CONTROL ERR
&EXIT

************************************************************************
-OPCT
*    COMMAND ENTERED: LSESS O OR LSESS OPCTL
&CONTROL CMD
LISTSESS OPCTL
&CONTROL ERR
&EXIT
************************************************************************
-FLSC
*    COMMAND ENTERED: LSESS F OR LSESS FLSCN
&CONTROL CMD
LISTSESS FLSCN
&CONTROL ERR
&EXIT
************************************************************************
-APPL
*    COMMAND ENTERED: LSESS APPLID  (OF FIVE CHARACTERS)
&IF .&2 EQ . &THEN &GOTO -ERROR
&CONTROL CMD
LISTSESS APPLID=&P2
&CONTROL ERR
&EXIT
************************************************************************
-SRCL
*    COMMAND ENTERED: LSESS O OR LSESS OPCTL
&IF .&2 EQ . &THEN &GOTO -ERROR
&CONTROL CMD
LISTSESS SRCLU=&P2
&CONTROL ERR
&EXIT
************************************************************************
-ERROR
MESSAGE 330E,LSESS,SECOND
&EXIT
************************************************************************
-HELP
HELP LSESS
&EXIT
       END OF CLIST */
/* REXX CONVERSIONS */
```

Figure 141 (Part 2 of 4). Bilingual Command List Created by CNMS8001

```
/*   CLIST */
 TRACE E
/* SEE THE NETVIEW OPERATIONS MANUAL AND/OR ENTER HELP CLISTNAME*/
/*   FOR A DESCRIPTION OF FUNCTION AND SYNTAX FOR THIS CLIST.*/
/*********************************************************************/
/*    (C) COPYRIGHT IBM CORP. 1986, 1987                           **/
/*                                                                 **/
/*    LAST CHANGE:    04/08/86  15:41:52       SSI=60981541        **/
/*                                                                 **/
/*    IEBCOPY   SELECT MEMBER=((CNME1007,LSESS,R))                 **/
/*                                                                 **/
/*    OUTPUTS:                                                     **/
/*        ONE OF:                                                  **/
/*            LISTSESS                                             **/
/*            LISTSESS OPCTL                                       **/
/*            LISTSESS FLSCN                                       **/
/*            LISTSESS APPLID=&2                                   **/
/*            LISTSESS SRCLU=&2                                    **/
/*                                                                 **/
/*    CNME1007 CHANGED ACTIVITY:                                   **/
/*    CHANGE CODE  DATE      DESCRIPTION                           **/
/*    -----------------------------------------                    **/
/*                                                                 **/
/*********************************************************************/
 P1 = MSGVAR(1)
 P2 = MSGVAR(2)
 IF '.'P1 = '.?' THEN SIGNAL HELP
 IF '.'P1 = '.' THEN SIGNAL HELP
 IF '.'P1 = '.ALL' THEN SIGNAL LALL
 IF '.'P1 = '.O' THEN SIGNAL OPCT
 IF '.'P1 = '.OPCTL' THEN SIGNAL OPCT
 IF '.'P1 = '.F' THEN SIGNAL FLSC
 IF '.'P1 = '.FLSCN' THEN SIGNAL FLSC
 IF '.'P1 = '.A' THEN SIGNAL APPL
 IF '.'P1 = '.S' THEN SIGNAL SRCL
/*     INVALID PARAMETER*/
 'MESSAGE 306E,LSESS,'MSGVAR(1)
 EXIT
/*********************************************************************/
LALL:
/*    COMMAND ENTERED: LSESS ALL*/
 TRACE C
 'LISTSESS'
 TRACE E
 EXIT
```

Figure 141 (Part 3 of 4). Bilingual Command List Created by CNMS8001

```
/*********************************************************************/
OPCT:
/*    COMMAND ENTERED: LSESS O OR LSESS OPCTL*/
 TRACE C
 'LISTSESS OPCTL'
 TRACE E
 EXIT
/*********************************************************************/
FLSC:
/*    COMMAND ENTERED: LSESS F OR LSESS FLSCN*/
 TRACE C
 'LISTSESS FLSCN'
 TRACE E
 EXIT
/*********************************************************************/
APPL:
/*    COMMAND ENTERED: LSESS APPLID  (OF FIVE CHARACTERS)*/
 IF '.'MSGVAR(2) = '.' THEN SIGNAL ERROR
 TRACE C
 'LISTSESS APPLID='P2
 TRACE E
 EXIT
/*********************************************************************/
SRCL:
/*    COMMAND ENTERED: LSESS O OR LSESS OPCTL*/
 IF '.'MSGVAR(2) = '.' THEN SIGNAL ERROR
 TRACE C
 'LISTSESS SRCLU='P2
 TRACE E
 EXIT
/*********************************************************************/
ERROR:
 'MESSAGE 330E,LSESS,SECOND'
 EXIT
/*********************************************************************/
HELP:
 'HELP LSESS'
 EXIT
```

Figure 141 (Part 4 of 4). Bilingual Command List Created by CNMS8001

Glossary, Bibliography, and Index

Glossary

This glossary defines important NCP, NetView, NetView/PC, SSP, and VTAM abbreviations and terms. It includes information from the *IBM Dictionary of Computing*, SC20-1699. Definitions from the *American National Dictionary for Information Processing* are identified by an asterisk (*). Definitions from draft proposals and working papers under development by the International Standards Organization, Technical Committee 97, Subcommittee 1 are identified by the symbol **(TC97)**. Definitions from the *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the Consultative Committee on International Telegraph and Telephone of the International Telecommunication Union, Geneva, 1980 are preceded by the symbol **(CCITT/ITU)**. Definitions from published sections of the *ISO Vocabulary of Data Processing*, developed by the International Standards Organization, Technical Committee 97, Subcommittee 1 and from published sections of the *ISO Vocabulary of Office Machines*, developed by subcommittees of ISO Technical Committee 95, are preceded by the symbol **(ISO)**.

For abbreviations, the definition usually consists only of the words represented by the letters; for complete definitions, see the entries for the words.

Reference Words Used in the Entries

The following reference words are used in this glossary:

Deprecated term for. Indicates that the term should not be used. It refers to a preferred term, which is defined.

Synonymous with. Appears in the commentary of a preferred term and identifies less desirable or less specific terms that have the same meaning.

Synonym for. Appears in the commentary of a less desirable or less specific term and identifies the preferred term that has the same meaning.

Contrast with. Refers to a term that has an opposed or substantively different meaning.

See. Refers to multiple-word terms that have the same last word.

See also. Refers to related terms that have similar (but not synonymous) meanings.

ACB name. (1) The name of an ACB macroinstruction. (2) A name specified in the ACBNAME parameter of a VTAM APPL statement. Contrast with *network name*.

accept. For a VTAM application program, to establish a session with a logical unit (LU) in response to a CINIT request from a system services control point (SSCP). The session-initiation request may begin when a terminal user logs on, a VTAM application program issues a macroinstruction, or a VTAM operator issues a command. See also *acquire (1)*.

access method. A technique for moving data between main storage and input/output devices.

accounting exit routine. In VTAM, an optional installation exit routine that collects statistics about session initiation and termination.

ACF. Advanced Communications Function.

ACF/NCP. Advanced Communications Function for the Network Control Program. Synonym for *NCP*.

ACF/SSP. Advanced Communications Function for the System Support Programs. Synonym for *SSP*.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method. Synonym for *VTAM*.

acquire. (1) For a VTAM application program, to initiate and establish a session with another logical unit (LU). The acquire process begins when the application program issues a macroinstruction. See also *accept*. (2) To take over resources that were formerly controlled by an access method in another domain, or to resume control of resources that were controlled by this domain but released. Contrast with *release*. See also *resource takeover*.

activate. To make a resource of a node ready to perform the functions for which it was designed. Contrast with *deactivate*.

active. (1) The state a resource is in when it has been activated and is operational. Contrast with *inactive*, *pending*, and *inoperative*.. (2) Pertaining to a major or minor node that has been activated by VTAM. Most resources are activated as part of VTAM start processing or as the result of a VARY ACT command.

adaptive session pacing. Synonym for *adaptive session-level pacing*.

adaptive session-level pacing. A form of session-level pacing in which session components exchange pacing windows that may vary in size during the course of a session. This allows transmission to adapt dynamically to variations in availability and demand of buffers on a session by session basis. Session pacing occurs within independent stages along the session path according to local congestion at the intermediate nodes. Synonymous with *adaptive session pacing*.

See *pacing*, *session-level pacing*, and *virtual route pacing*.

Advanced Communications Function (ACF). A group of IBM licensed programs (principally VTAM, TCAM, NCP, and SSP) that use the concepts of Systems Network Architecture (SNA), including distribution of function and resource sharing.

alert. (1) In SNA, a record sent to a system problem management focal point to communicate the existence of an alert condition. (2) In the NetView program, a high priority event that warrants immediate attention. This data base record is generated for certain event types that are defined by user-constructed filters.

allocate. A logical unit (LU) 6.2 application program interface (API) verb used to assign a session to a conversation for the conversation's use. Contrast with *deallocate*.

API. Application program interface.

application program. (1) A program written for or by a user that applies to the user's work. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application program interface (API). (1) The formally defined programming language interface between an IBM system control program or licensed program and its user. (2) The interface through which an application program interacts with an access method. In VTAM, it is the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

attaching device. Any device that is physically connected to a network and can communicate over the network.

authorization exit routine. In VTAM, an optional installation exit routine that approves or disapproves requests for session initiation.

authorized receiver. In the NetView program, an authorized operator who receives all the unsolicited and authorized-receiver messages not assigned to a specific operator.

automatic logon. (1) A process by which VTAM automatically creates a session-initiation request to establish a session between two logical units (LUs). The session will be between a designated primary logical unit (PLU) and a secondary logical unit (SLU) that is neither queued for nor in session with another PLU. See also *controlling application program* and *controlling logical unit*. (2) In VM, a process by which a virtual machine is initiated by other than the user of that virtual machine. For example, the primary VM operator's virtual machine is activated automatically during VM initialization.

available. In VTAM, pertaining to a logical unit that is active, connected, enabled, and not at its session limit.

BIU segment. In SNA, the portion of a basic information unit (BIU) that is contained within a path information unit (PIU). It consists of either a request/response header (RH) followed by all or a portion of a request/response unit (RU), or only a portion of an RU.

blocking of PIUs. In SNA, an optional function of path control that combines multiple path information units (PIUs) into a single basic transmission unit (BTU).

boundary function. (1) A capability of a subarea node to provide protocol support for attached peripheral nodes, such as: (a) interconnecting subarea path control and peripheral path control elements, (b) performing session sequence numbering for low-function peripheral nodes, and (c) providing session-level pacing support. (2) The component that provides these capabilities. See also *boundary node*, *network addressable unit (NAU)*, *peripheral path control*, *subarea node*, and *subarea path control*.

boundary node. (1) A subarea node with boundary function. See *subarea node* (including illustration). See also *boundary function*. (2) The programming component that performs FID2 (format identification type 2) conversion, channel data link control, pacing, and channel or device error recovery procedures for a locally attached station. These functions are similar to those performed by a network control program for an NCP-attached station.

browse. A way of looking at a file that does not allow you to change it.

buffer. A portion of storage for temporarily holding input or output data.

call. (1) * (ISO) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (2) To transfer control to a procedure, program, routine, or subroutine. (3) The actions necessary to make a connection between two stations. (4) To attempt to contact a user, regardless of whether the attempt is successful.

CALLOUT. The logical channel type on which the data terminal equipment (DTE) can send a call, but cannot receive one.

calling. * (ISO) The process of transmitting selection signals in order to establish a connection between data stations.

chain. (1) A group of logically linked records. (2) See *RU chain*.

channel-attached. (1) Pertaining to the attachment of devices directly by input/output channels to a host processor. (2) Pertaining to devices attached to a controlling unit by cables, rather than by telecommunication lines. Contrast with *link-attached*. Synonymous with *local*.

character-coded. Synonym for *unformatted*.

CICS. Customer Information Control System.

CLIST. Command list.

command. (1) A request from a terminal for the performance of an operation or the execution of a particular program. (2) In SNA, any field set in the transmission header (TH), request header (RH), and sometimes portions of a request unit (RU), that initiates an action or that begins a protocol; for example: (a) Bind Session (session-control request unit), a command that activates an LU-LU session, (b) the change-direction indicator in the RH of the last RU of a chain, (c) the virtual route reset window indicator in a FID4 transmission header. See also *VTAM operator command*.

command facility. The component of the NetView program that is a base for command processors that can monitor, control, automate, and improve the operation of a network.

command list. A list of commands and statements designed to perform a specific function for the user. Command lists can be written in REXX or in NetView command list language.

command processor. (1) A program that performs an operation specified by a command. (2) In the NetView program, a user-written module designed to perform a specific function. Command processors, which can be written in assembler or a high-level language (HLL), are invoked as commands.

communication line. Deprecated term for *telecommunication line* and *transmission line*.

communication management configuration host node. The type 5 host processor in a communication management configuration that does all network-control functions in the network except for the control of devices channel-attached to data hosts. Synonymous with *communication management host*. Contrast with *data host node*.

communication management host. Synonym for *communication management configuration host node*. Contrast with *data host*.

component. A command that (a) controls the terminal's screen (using the DSIPSS macro (TYPE=ASYPANEL) or the VIEW command), (b) allows the operator to enter NetView commands, and (c) can resume when such commands are complete.

composite end node (CEN). A group of nodes made up of a single type 5 node and its subordinate type 4 nodes that together support type 2.1 protocols. To a type 2.1 node, a CEN appears as one end node. For example, NCP and VTAM act as a composite end node.

configuration. (1) (TC97) The arrangement of a computer system or network as defined by the nature, number, and the chief characteristics of its functional units. The term may refer to a hardware or a software configuration. (2) The devices and programs that make up a system, subsystem, or network. (3) In CCP, the arrangement of controllers, lines, and terminals attached to an IBM 3710 Network Controller. Also, the collective set of item definitions that describe such a configuration.

configuration services. In SNA, one of the types of network services in the control point (CP) and in the physical unit (PU); configuration services activate, deactivate, and maintain the status of physical units, links, and link stations. Configuration services also shut down and restart network elements and modify path control routing tables and address-translation tables. See also *maintenance services*, *management services*, *network services*, and *session services*.

connection. Synonym for *physical connection*.

control program (CP). The VM operating system that manages the real processor's resources and is responsible for simulating System/370s for individual users.

controlling application program. In VTAM, an application program with which a secondary logical unit (other than an application program) is automatically put in session whenever the secondary logical unit is available. See also *automatic logon* and *controlling logical unit*.

controlling logical unit. In VTAM, a logical unit with which a secondary logical unit (other than an application program) is automatically put in session whenever the secondary logical unit is available. A controlling logical unit can be either an application program or a device-type logical unit. See also *automatic logon* and *controlling application program*.

control statement. A statement in a command list that controls the processing sequence of the command list or allows the command list to send messages to the operator and receive input from the operator.

converted command. An intermediate form of a character-coded command produced by VTAM through use of an unformatted system services definition table. The format of a converted command is fixed; the unformatted system services definition table must be constructed in such a manner that the character-coded command (as entered by a logical unit) is converted into the predefined, converted command format. See also *unformatted*.

cross-domain. In SNA, pertaining to control of resources involving more than one domain.

Customer Information Control System (CICS). A licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It also includes facilities for building, using, and maintaining data bases.

data host. Synonym for *data host node*. Contrast with *communication management configuration host*.

data host node. In a communication management configuration, a type 5 host node that is dedicated to processing applications and does not control network resources, except for its channel-attached or communication adapter-attached devices. Synonymous with *data host*. Contrast with *communication management configuration host node*.

data link. In SNA, synonym for *link*.

data link control (DLC) layer. In SNA, the layer that consists of the link stations that schedule data transfer over a transmission medium connecting two nodes and perform error control for the link connection. Examples of data link control are SDLC for serial-by-bit link connection and data link control for the System/370 channel.

data set. The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

DBCS. Double-byte character set.

ddname. Data definition name.

deactivate. To take a resource of a node out of service, rendering it inoperable, or to place it in a state in which it cannot perform the functions for which it was designed. Contrast with *activate*.

deallocate. A logical unit (LU) 6.2 application program interface (API) verb that terminates a conversation, thereby freeing the session for a future conversation. Contrast with *allocate*.

definite response (DR). In SNA, a value in the form-of-response-requested field of the request header. The value directs the receiver of the request to return a response unconditionally, whether positive or negative, to that request. Contrast with *exception response* and *no response*.

definition statement. (1) In VTAM, the statement that describes an element of the network. (2) In NCP, a type of instruction that defines a resource to the NCP. See Figure 142, Figure 143, and Figure 144. See also *macroinstruction*.

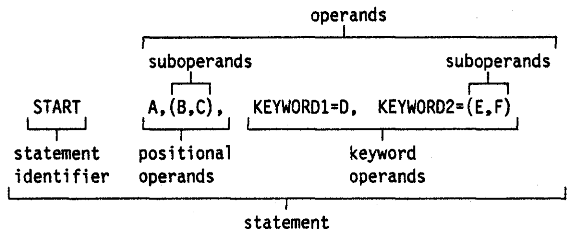


Figure 142. Example of a Language Statement

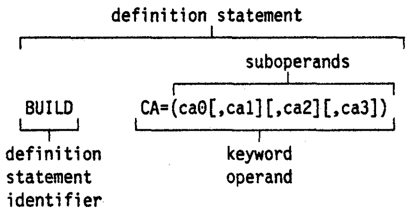


Figure 143. NCP Examples

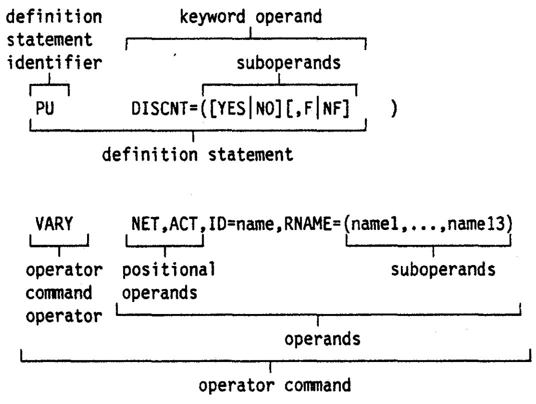

definition statement identifier
keyword operand
suboperands
PU
DISCNT=([YES|NO][,F|NF])
definition statement
VARY
NET,ACT,ID=name,RNAME=(name1,...,name13)
operator command operator
positional operands
suboperands
operands
operator command

Figure 144. VTAM Examples

device. An input/output unit such as a terminal, display, or printer. See *attaching device*.

dial-out. Refers to the direction in which a switched connection is requested by a host or an NCP.

directory. In VM, a control program (CP) disk that defines each virtual machine's normal configuration.

disabled. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is temporarily not ready to establish LU-LU sessions. An initiate request for a session with a disabled logical unit (LU) can specify that the session be queued by the SSCP until the LU becomes enabled. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or a secondary logical unit (SLU). See also *enabled* and *inhibited*.

display. (1) To present information for viewing, usually on a terminal screen or a hard-copy device. (2) A device or medium on which information is presented, such as a terminal screen. (3) Deprecated term for *panel*.

domain. (1) An access method, its application programs, communication controllers, connecting lines, modems, and attached terminals. (2) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests. See *system services control point domain* and *type 2.1 node control point domain..* See also *single-domain network* and *multiple-domain network*.

domain operator. In a multiple-domain network, the person or program that controls the operation of the resources controlled by one system services control point. Contrast with *network operator* (2).

double-byte character set (DBCS). A character set, such as Japanese, in which each character is represented by a two-byte code.

downstream. In the direction of data flow from the host to the end user. Contrast with *upstream*.

DR. (1) In NCP and CCP, dynamic reconfiguration. (2) In SNA, definite response.

drop. In the IBM Token-Ring Network, a cable that leads from a faceplate to the to the distribution panel in a wiring closet. When the IBM Cabling System is used with the IBM Token-Ring Network, a drop may form part of a lobe.

dynamic reconfiguration (DR). The process of changing the network configuration (peripheral PUs and LUs) without regenerating complete configuration tables.

EBCDIC. * Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

echo. The return of characters to the originating SS device to verify that a message was sent correctly.

element. (1) A field in the network address. (2) The particular resource within a subarea identified by the element address. See also *subarea*.

Emulation Program (EP). An IBM control program that allows a channel-attached 3705 or 3725 communication controller to emulate the functions of an IBM 2701 Data Adapter Unit, an IBM 2702 Transmission Control, or an IBM 2703 Transmission Control. See also *network control program*.

enabled. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is now ready to establish LU-LU sessions. The LU can separately indicate whether this prevents it from acting as a primary logical unit (PLU) or as a secondary logical unit (SLU). See also *disabled* and *inhibited*.

end node. A type 2.1 node that does not provide any intermediate routing or session services to any other node. For example, APPC/PC is an end node. See *composite end node*, *node*, and *type 2.1 node*.

EP. Emulation Program.

ER. (1) Explicit route. (2) Exception response.

error-to-traffic (E/T). The number of temporary errors compared to the traffic associated with a resource.

E/T. Error-to-traffic.

event. (1) In the NetView program, a record indicating irregularities of operation in physical elements of a network. (2) An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as an input/output operation.

exception response (ER). In SNA, a value in the form-of-response-requested field of a request header (RH). An exception response is sent only if a request is unacceptable as received or cannot be processed. Contrast with *definite response* and *no response*. See also *negative response*.

EXEC. In a VM operating system, a user-written command file that contains CMS commands, other user-written commands, and execution control statements, such as branches.

exit routine. Any of several types of special-purpose user-written routines. See *accounting exit routine, authorization exit routine, logon-interpret routine, virtual route selection exit routine, EXLST exit routine*, and *RPL exit routine*.

EXLST exit routine. In VTAM, a routine whose address has been placed in an exit list (EXLST) control block. The addresses are placed there with the EXLST macroinstruction, and the routines are named according to their corresponding operand; hence DFASY exit routine, TPEND exit routine, RELREQ exit routine, and so forth. All exit list routines are coded by the VTAM application programmer. Contrast with *RPL exit routine*.

explicit route (ER). In SNA, the path control network elements, including a specific set of one or more transmission groups, that connect two subarea nodes. An explicit route is identified by an origin subarea address, a destination subarea address, an explicit route number, and a reverse explicit route number. Contrast with *virtual route (VR)*. See also *path* and *route extension*.

feature. A particular part of an IBM product that a customer can order separately.

FID. Format identification.

field-formatted. Pertaining to a request or response that is encoded into fields, each having a specified format such as binary codes, bit-significant flags, and symbolic names. Contrast with *character-coded*.

flow control. In SNA, the process of managing the rate at which data traffic passes between components of the network. The purpose of flow control is to optimize the rate of flow of message units, with minimum congestion in the network; that is, to neither overflow the buffers at the receiver or at intermediate routing nodes, nor leave the receiver waiting for more message units. See also *adaptive session-level pacing*, *pacing*, *session-level pacing*, and *virtual route pacing*.

flushing. In logical unit (LU) 6.2, the process of sending through the network all remaining buffered data generated by a transaction program. The transaction program issues the flush verb to begin the process. It also occurs if the network operator issues the command.

format identification (FID) field. In SNA, a field in each transmission header (TH) that indicates the format of the TH; that is, the presence or absence of certain fields. TH formats differ in accordance with the types of nodes between which they pass. The six FID types are:

FID0, used for traffic involving non-SNA devices between adjacent subarea nodes when either or both nodes do not support explicit route and virtual route protocols.

FID1, used for traffic between adjacent subarea nodes when either or both nodes do not support explicit route and virtual route protocols.

FID2, used for traffic between a subarea node and an adjacent type 2 peripheral node.

FID3, used for traffic between a subarea node and an adjacent type 1 peripheral node.

FID4, used for traffic between adjacent subarea nodes when both nodes support explicit route and virtual route protocols.

FIDF, used for certain commands (for example, for transmission group control) sent between adjacent subarea nodes when both nodes support explicit route and virtual route protocols.

frame. (1) The unit of transmission in some local area networks, including the IBM Token-Ring Network. It includes delimiters, control characters, information, and checking characters. (2) In SDLC, the vehicle for every command, every response, and all information that is transmitted using SDLC procedures.

full-screen mode. A form of panel presentation in the NetView program where the contents of an entire terminal screen can be displayed at once. Full-screen mode can be used for fill-in-the-blanks prompting. Contrast with *line mode*.

GCS. Group control system.

generation. The process of assembling and link editing definition statements so that resources can be identified to all the necessary programs in a network.

generic alert. Encoded alert information that uses code points (defined by IBM and possibly customized by users or application programs) stored at an alert receiver, such as the NetView program.

group. In the NetView/PC program, to identify a set of application programs that are to run concurrently.

group control system (GCS). A component of VM that provides multiprogramming and shared memory support to virtual machines. It is a saved system intended for use with SNA products.

half-session. In SNA, a component that provides function management data (FMD) services, data flow control, and transmission control for one of the sessions of a network addressable unit (NAU). See also *primary half-session* and *secondary half-session*.

hard copy. A printed copy of machine output in a visually readable form; for example, printed reports, listings, documents, summaries, or network logs.

hardware monitor. The component of the NetView program that helps identify network problems, such as hardware, software, and microcode, from a central control point using interactive display techniques.

HCF. Host Command Facility.

help panel. An online display that tells you how to use a command or another aspect of a product. See *task panel*.

High Performance Option (HPO). A licensed program that is an extension of VM/SP. It provides performance and operation enhancements for large system environments.

Host Command Facility (HCF). An IBM licensed program that enables a user at a System/370 terminal to access applications in systems such as the 8100 or System/36.

host node. A node providing an application program interface (API) and a common application interface. See *boundary node*, *node*, *peripheral node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

HPO. High Performance Option.

IMS. Information Management System/Virtual Storage. Synonymous with *IMS/VS*.

IMS/VS. Information Management System/Virtual Storage. Synonym for *IMS*.

inactive. Describes the state of a resource that has not been activated or for which the VARY INACT command has been issued. Contrast with *active*. See also *inoperative*.

information (I) format. A format used for information transfer.

Information/Management. A feature of the Information/System licensed program that provides interactive systems management applications for problem, change, and configuration management.

Information Management System (IMS). A general purpose system whose full name is Information Management System/Virtual Storage (IMS/VS). It enhances the capabilities of OS/VS for batch processing and telecommunication and allows users to access a computer-maintained data base through remote terminals.

inhibited. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is not ready to establish LU-LU sessions. An initiate request for a session with an inhibited LU will be rejected by the SSCP. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or as a secondary logical unit (SLU). See also *enabled* and *disabled*.

initiate. A network services request sent from a logical unit (LU) to a system services control point (SSCP) requesting that an LU-LU session be established.

inoperative. The condition of a resource that has been active, but is not. The resource may have failed, received an INOP request, or is suspended while a reactivate command is being processed. See also *inactive*.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialogue manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

interface. * A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs.

ISPF. Interactive System Productivity Facility.

item. In CCP, any of the components, such as communication controllers, lines, cluster controllers, and terminals, that comprise an IBM 3710 Network Controller configuration.

JCL. Job control language.

job control language (JCL). * A problem-oriented language designed to express statements in a job that are used to identify the job or describe its requirements to an operating system.

Kanji. An ideographic character set used in Japanese. See also *double-byte character set*.

keyword. (1) (TC97) A lexical unit that, in certain contexts, characterizes some language construction. (2) * One of the predefined words of an artificial language. (3) One of the significant and informative words in a title or document that describes the content of that document. (4) A name or symbol that identifies a parameter. (5) A part of a command operand that consists of a specific character string (such as DSNAME=). See also *definition statement* and *keyword operand*. Contrast with *positional operand*.

keyword operand. An operand that consists of a keyword followed by one or more values (such as DSNAME=HELLO). See also *definition statement*. Contrast with *positional operand*.

keyword parameter. A parameter that consists of a keyword followed by one or more values.

line. See *communication line*.

line mode. A form of screen presentation in which the information is presented a line at a time in the message area of the terminal screen. Contrast with *full-screen mode*.

link. In SNA, the combination of the link connection and the link stations joining network nodes; for example: (1) a System/370 channel and its associated protocols, (2) a serial-by-bit connection under the control of Synchronous Data Link Control (SDLC). A link connection is the physical medium of transmission. A link, however, is both logical and physical. Synonymous with *data link*. See Figure 145 on page 230.

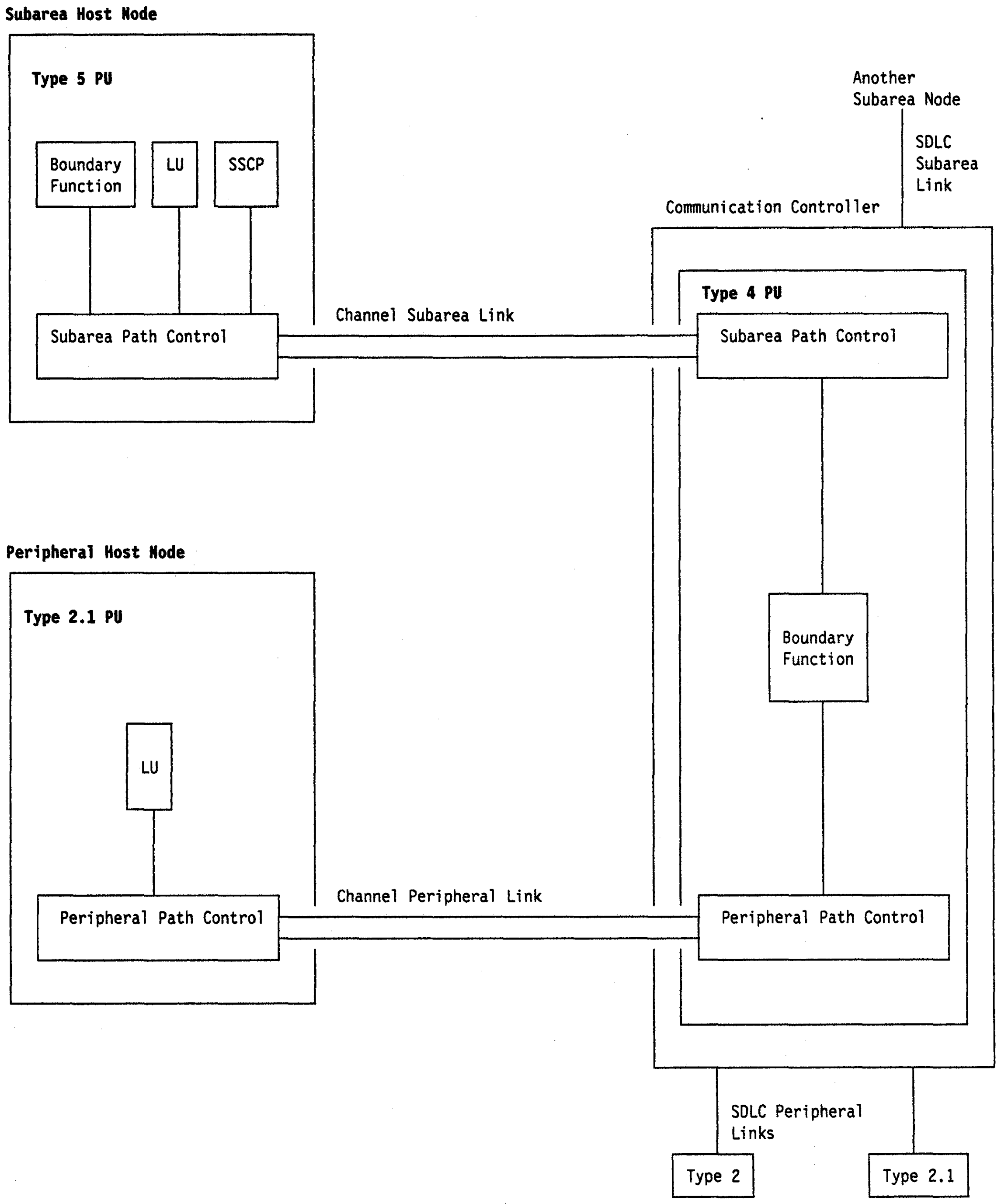


Figure 145. Links and Path Controls

link-attached. Pertaining to devices that are physically connected by a telecommunication line. Contrast with *channel-attached.* Synonymous with *remote.*

link connection segment. A portion of the configuration that is located between two resources listed consecutively in the service point command service (SPCS) query link configuration request list.

load module. (ISO) A program unit that is suitable for loading into main storage for execution; it is usually the output of a linkage editor.

local. Pertaining to a device that is attached to a controlling unit by cables, rather than by a telecommunication line. Synonymous with *channel-attached.*

local address. In SNA, an address used in a peripheral node in place of an SNA network address and transformed to or from an SNA network address by the boundary function in a subarea node.

logical unit (LU). In SNA, a port through which an end user accesses the SNA network and the functions provided by system services control points (SSCPs). An LU can support at least two sessions—one with an SSCP and one with another LU—and may be capable of supporting many sessions with other LUs. See also *network addressable unit (NAU)*, *peripheral LU*, *physical unit (PU)*, *system services control point (SSCP)*, *primary logical unit (PLU)*, and *secondary logical unit (SLU)*.

logical unit (LU) services. In SNA, capabilities in a logical unit to: (1) receive requests from an end user and, in turn, issue requests to the system services control point (SSCP) in order to perform the requested functions, typically for session initiation; (2) receive requests from the SSCP, for example to activate LU-LU sessions via Bind Session requests; and (3) provide session presentation and other services for LU-LU sessions. See also *physical unit (PU) services*.

logical unit (LU) 6.2. A type of logical unit that supports general communication between programs in a distributed processing environment. LU 6.2 is characterized by (1) a peer relationship between session partners, (2) efficient utilization of a session for multiple transactions, (3) comprehensive end-to-end error processing, and (4) a generic application program interface (API) consisting of structured verbs that are mapped into a product implementation.

logoff. In VTAM, an unformatted session termination request.

logon. In VTAM, an unformatted session initiation request for a session between two logical units. See *automatic logon* and *simulated logon*. See also *session-initiation request*.

logon-interpret routine. In VTAM, an installation exit routine, associated with an interpret table entry, that translates logon information. It may also verify the logon.

LU. Logical unit.

LU type. In SNA, the classification of an LU-LU session in terms of the specific subset of SNA protocols and options supported by the logical units (LUs) for that session, namely:

The mandatory and optional values allowed in the session activation request.

The usage of data stream controls, function management headers (FMHs), request unit (RU) parameters, and sense codes.

Presentation services protocols such as those associated with FMH usage.

LU types 0, 1, 2, 3, 4, 6.1, 6.2, and 7 are defined.

LU-LU session. In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two end users, or between an end user and an LU services component.

LU-LU session type. A deprecated term for *LU type*.

LU 6.2. Logical unit 6.2.

macroinstruction. (1) An instruction that when executed causes the execution of a predefined sequence of instructions in the same source language. (2) In assembler programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macro definition. The statements normally produced from the macro definition replace the macroinstruction in the program. See also *definition statement*.

maintenance services. In SNA, one of the types of network services in system services control points (SSCPs) and physical units (PUs). Maintenance services provide facilities for testing links and nodes and for collecting and recording error information. See also *configuration services, management services, network services*, and *session services*.

major node. In VTAM, a set of resources that can be activated and deactivated as a group. See *node* and *minor node*.

management services. In SNA, one of the types of network services in control points (CPs) and physical units (PUs). Management services are the services provided to assist in the management of SNA networks, such as problem management, performance and accounting management, configuration management and change management. See also *configuration services, maintenance services, network services*, and *session services*.

message. (1) (TC97) A group of characters and control bit sequences transferred as an entity. (2) In VTAM, the amount of function management data (FMD) transferred to VTAM by the application program with one SEND request.

migration. Installing a new version or release of a program when an earlier version or release is already in place.

minidisk. Synonym for *virtual disk*.

minor node. In VTAM, a uniquely-defined resource within a major node. See *node* and *major node*.

module. * A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine.

monitor. In the IBM Token-Ring Network, the function required to initiate the transmission of a token on the ring and to provide soft-error recovery in case of lost tokens, circulating frames, or other difficulties. The capability is present in all ring stations.

multiple-domain network. In SNA, a network with more than one system services control point (SSCP). Contrast with *single-domain network.*

Multiple Virtual Storage (MVS). An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for System/370. It is a software operating system controlling the execution of programs.

Multiple Virtual Storage for Extended Architecture (MVS/XA). An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for Extended Architecture. Extended architecture allows 31-bit storage addressing. MVS/XA is a software operating system controlling the execution of programs.

MVS. Multiple Virtual Storage operating system.

MVS/OCCF. Multiple Virtual Storage/Operator Communication Control Facility.

MVS/XA. Multiple Virtual Storage for Extended Architecture operating system.

NAU. Network addressable unit.

NCCF. Network Communications Control Facility.

NCP. (1) Network Control Program (IBM licensed program). Its full name is Advanced Communications Function for the Network Control Program. Synonymous with *ACF/NCP.* (2) Network control program (general term).

negative response (NR). In SNA, a response indicating that a request did not arrive successfully or was not processed successfully by the receiver. Contrast with *positive response.* See *exception response.*

NetView. A system 370-based IBM licensed program used to monitor a network, manage it, and diagnose its problems.

NetView command list language. An interpretive language unique to the NetView program that is used to write command lists.

NetView-NetView task (NNT). The task under which a cross-domain NetView operator session runs. See *operator station task.*

NetView/PC. A PC-based IBM licensed program through which application programs can be used to monitor, manage, and diagnose problems in IBM Token-Ring networks, non-SNA communication devices, and voice networks.

network. (1) (TC97) An interconnected group of nodes. (2) In data processing, a user application network. See *path control network, public network, SNA network, subarea network, type 2.1 network,* and *user-application network.*

network address. In SNA, an address, consisting of subarea and element fields, that identifies a link, a link station, or a network addressable unit. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See *local address.* See also *network name.*

network addressable unit (NAU). In SNA, a logical unit, a physical unit, or a system services control point. It is the origin or the destination of information transmitted by the path control network. Each NAU has a network address that represents it to the path control network. See also *network name, network address,* and *path control network.*

Network Communications Control Facility (NCCF). An IBM licensed program that is a base for command processors that can monitor, control, automate, and improve the operations of a network. Its function is included and enhanced in NetView's command facility.

network control (NC). In SNA, an RU category used for requests and responses exchanged for such purposes as activating and deactivating explicit and virtual routes and sending load modules to adjacent peripheral nodes. See also *data flow control layer* and *session control.*

Network Control Program (NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Its full name is Advanced Communications Function for the Network Control Program.

network control program. A program, generated by the user from a library of IBM-supplied modules, that controls the operation of a communication controller.

network identifier (network ID). The network name defined to NCPs and hosts to indicate the name of the network in which they reside. It is unique across all communicating SNA networks. communication among domains.

Network Logical Data Manager (NLDM). An IBM licensed program that collects and correlates session-related data and provides online access to this information. It runs as an NCCF communication network management (CNM) application program. Its function is included and enhanced in NetView's session monitor.

network name. (1) In SNA, the symbolic identifier by which end users refer to a network addressable unit (NAU), a link, or a link station. See also *network address*. (2) In a multiple-domain network, the name of the APPL statement defining a VTAM application program is its network name and it must be unique across domains. Contrast with *ACB name*. See *uninterpreted name*.

network operator. (1) A person or program responsible for controlling the operation of all or part of a network. (2) The person or program that controls all the domains in a multiple-domain network. Contrast with *domain operator*.

Network Problem Determination Application (NPDA). An IBM licensed program that helps you identify network problems, such as hardware, software, and microcode, from a central control point using interactive display techniques. It runs as an NCCF communication network management (CNM) application program. Its function is included and enhanced in NetView's hardware monitor.

network services (NS). In SNA, the services within network addressable units (NAUs) that control network operation through SSCP-SSCP, SSCP-PU, and SSCP-LU sessions. See *configuration services, maintenance services, management services*, and *session services*.

NLDM. Network Logical Data Manager.

NNT. NetView-NetView task.

node. (1) In SNA, an endpoint of a link or junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities. See *boundary node, host node, peripheral node*, and *subarea node* (including illustration). (2) In VTAM, a point in a network defined by a symbolic name. See *major node* and *minor node*.

node name. In VTAM, the symbolic name assigned to a specific major or minor node during network definition.

node type. In SNA, a designation of a node according to the protocols it supports and the network addressable units (NAUs) that it can contain. Five types are defined: 1, 2.0, 2.1, 4, and 5. Type 1, type 2.0, and type 2.1 nodes are peripheral nodes; type 4 and type 5 nodes are subarea nodes. See also *type 2.1 node*.

no response. In SNA, a value in the form-of-response-requested field of the request header (RH) indicating that no response is to be returned to the request, whether or not the request is received and processed successfully. Contrast with *definite response* and *exception response*.

notify. A network services request that is sent by an SSCP to a logical unit (LU) to inform the LU of the status of a procedure requested by the LU.

NPDA. Network Problem Determination Application.

OCCF. Operator Communication Control Facility.

online. Stored in a computer and accessible from a terminal.

operand. (1) (ISO) An entity on which an operation is performed. (2) * That which is operated upon. An operand is usually identified by an address part of an instruction. (3) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor. (4) An expression to whose value an operator is applied. See also *definition statement, keyword, keyword parameter*, and *parameter*.

operator. (1) In a language statement, the lexical entity that indicates the action to be performed on operands. (2) A person who operates a machine. See *network operator*. See also *definition statement*.

Operator Communication Control Facility (OCCF). A licensed program that allows communication with and the operation of remote MVS or VSE systems.

operator profile. In the NetView program, the resources and activities a network operator has control over. The statements defining these resources and activities are stored in a file that is activated when the operator logs on.

operator station task (OST). The NetView task that establishes and maintains the online session with the network operator. There is one operator station task for each network operator who logs on to the NetView program. See *NetView-NetView task*.

OST. Operator station task.

pacing. In SNA, a technique by which a receiving component controls the rate of transmission of a sending component to prevent overrun or congestion. See *session-level pacing, send pacing*, and *virtual route (VR) pacing*. See also *flow control*.

page. (1) The portion of a panel that is shown on a display surface at one time. (2) To move back and forth among the pages of a multiple-page panel. See also *scroll*. (3) (ISO) In a virtual storage system, a fixed-length block that has a virtual address and that can be transferred between real storage and auxiliary storage. (4) To transfer instructions, data, or both between real storage and external page or auxiliary storage.

panel. (1) A formatted display of information that appears on a terminal screen. See also *help panel* and *task panel*. Contrast with *screen*. (2) In computer graphics, a display image that defines the locations and characteristics of display fields on a display surface.

parameter. (1) (ISO) A variable that is given a constant value for a specified application and that may denote the application. (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (3) Data passed to a program or procedure by a user or another program, namely as an operand in a language statement, as an item in a menu, or as a shared data structure. See also *keyword*, *keyword parameter*, and *operand*.

path. (1) In SNA, the series of path control network components (path control and data link control) that are traversed by the information exchanged between two network addressable units (NAUs). See also *explicit route (ER)*, *route extension*, and *virtual route (VR)*. (2) In VTAM when defining a switched major node, a potential dial-out port that can be used to reach that node. (3) In the NetView/PC program, a complete line in a configuration that contains all of the resources in the service point command service (SPCS) query link configuration request list.

path control (PC). The function that routes message units between network addressable units (NAUs) in the network and provides the paths between them. It converts the BIUs from transmission control (possibly segmenting them) into path information units (PIUs) and exchanges basic transmission units (BTUs) and one or more PIUs with data link control. Path control differs for peripheral nodes, which use local addresses for routing, and subarea nodes, which use network addresses for routing. See *peripheral path control* and *subarea path control*. See also *link*, *peripheral node*, and *subarea node*.

path control (PC) layer. In SNA, the layer that manages the sharing of link resources of the SNA network and routes basic information units (BIUs) through it. See also *BIU segment*, *blocking of PIUs*, *data link control layer*, and *transmission control layer*.

path control (PC) network. In SNA, the part of the SNA network that includes the data link control and path control layers. See *SNA network* and *user application network*. See also *boundary function*.

PC. (1) Path control. (2) Personal Computer. Its full name is the IBM Personal Computer.

peripheral host node. A node that provides an application program interface (API) for running application programs but does not provide SSCP functions and is not aware of the network configuration. The peripheral host node does not provide subarea node services. It has boundary function provided by its adjacent subarea. See *boundary node*, *host node*, *node*, *peripheral node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

peripheral LU. In SNA, a logical unit representing a peripheral node.

peripheral node. In SNA, a node that uses local addresses for routing and therefore is not affected by changes in network addresses. A peripheral node requires boundary-function assistance from an adjacent subarea node. A peripheral node is a physical unit (PU) type 1, 2.0, or 2.1 node connected to a subarea node with boundary function within a subarea. See *boundary node*, *host node*, *node*, *peripheral host node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

peripheral path control. The function in a peripheral node that routes message units between units with local addresses and provides the paths between them. See *path control* and *subarea path control*. See also *boundary function*, *peripheral node*, and *subarea node*.

peripheral PU. In SNA, a physical unit representing a peripheral node.

Personal Computer (PC). The IBM Personal Computer line of products including the 5150 and subsequent models.

physical connection. In VTAM, a point-to-point connection or multipoint connection. Synonymous with *connection*.

physical unit (PU). In SNA, a type of network addressable unit (NAU). A physical unit (PU) manages and monitors the resources (such as attached links) of a node, as requested by a system services control point (SSCP) through an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links. See also *peripheral PU* and *subarea PU*.

physical unit (PU) services. In SNA, the components within a physical unit (PU) that provide configuration services and maintenance services for SSCP-PU sessions. See also *logical unit (LU) services*.

PLU. Primary logical unit.

POI. Programmed operator interface.

positional operand. An operand in a language statement that has a fixed position. See also *definition statement*. Contrast with *keyword operand*.

positive response. A response indicating that a request was received and processed. Contrast with *negative response*.

PPT. Primary POI task.

primary half-session. In SNA, the half-session that sends the session activation request. See also *primary logical unit*. Contrast with *secondary half-session*.

primary logical unit (PLU). In SNA, the logical unit (LU) that contains the primary half-session for a particular LU-LU session. Each session must have a PLU and secondary logical unit (SLU). The PLU is the unit responsible for the bind and is the controlling LU for the session. A particular LU may contain both primary and secondary half-sessions for different active LU-LU sessions. Contrast with *secondary logical unit (SLU)*.

primary POI task (PPT). The NetView subtask that processes all unsolicited messages received from the VTAM program operator interface (POI) and delivers them to the controlling operator or to the command processor. The PPT also processes the initial command specified to execute when the NetView program is initialized and timer request commands scheduled to execute under the PPT.

problem determination. The process of identifying the source of a problem; for example, a program component, a machine failure, telecommunication facilities, user or contractor-installed programs or equipment, an environment failure such as a power loss, or a user error.

profile. In the Conversational Monitor System (CMS) or the group control system (GCS), the characteristics defined by a PROFILE EXEC file that executes automatically after the system is loaded into a virtual machine. See also *operator profile*.

programmed operator interface (POI). A VTAM function that allows programs to perform VTAM operator functions.

PU. Physical unit.

public network. A network established and operated by communication common carriers or telecommunication Administrations for the specific purpose of providing circuit-switched, packet switched, and leased-circuit services to the public. Contrast with *user-application network*.

PU-PU flow. In SNA, the exchange between physical units (PUs) of network control requests and responses.

receive pacing. In SNA, the pacing of message units that the component is receiving. See also *send pacing*.

record. (1) (ISO) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (2) (TC97) A set of data treated as a unit. (3) A set of one or more related data items grouped for processing. (4) In VTAM, the unit of data transmission for record mode. A record represents whatever amount of data the transmitting node chooses to send.

release. For VTAM, to relinquish control of resources (communication controllers or physical units). See also *resource takeover*. Contrast with *acquire (2)*.

remote. Concerning the peripheral parts of a network not centrally linked to the host processor and generally using telecommunication lines with public right-of-way.

remove. In the IBM Token-Ring Network, to take an attaching device off the ring.

reset. On a virtual circuit, reinitialization of data flow control. At reset, all data in transit are eliminated.

resource. (1) Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs. (2) In the NetView program, any hardware or software that provides function to the network.

resource takeover. In VTAM, action initiated by a network operator to transfer control of resources from one domain to another. See also *acquire (2)* and *release*. See *takeover*.

response. A reply represented in the control field of a response frame. It advises the primary or combined station of the action taken by the secondary or other combined station to one or more commands. See also *command*.

Restructured Extended Executor (REXX). An interpretive language used to write command lists.

return code. * A code [returned from a program] used to influence the execution of succeeding instructions.

REXX. Restructured Extended Executor.

route. See *explicit route* and *virtual route*.

route extension (REX). In SNA, the path control network components, including a peripheral link, that make up the portion of a path between a subarea node and a network addressable unit (NAU) in an adjacent peripheral node. See also *path, explicit route (ER)* and *virtual route (VR)*.

routing. The assignment of the path by which a message will reach its destination.

RPL exit routine. In VTAM, an application program exit routine whose address has been placed in the EXIT field of a request parameter list (RPL). VTAM invokes the routine to indicate that an asynchronous request has been completed. See *EXLST exit routine*.

RU chain. In SNA, a set of related request/response units (RUs) that are consecutively transmitted on a particular normal or expedited data flow. The request RU chain is the unit of recovery: if one of the RUs in the chain cannot be processed, the entire chain is discarded. Each RU belongs to only one chain, which has a beginning and an end indicated by means of control bits in request/response headers within the RU chain. Each RU can be designated as first-in-chain (FIC), last-in-chain (LIC), middle-in-chain (MIC), or only-in-chain (OIC). Response units and expedited-flow request units are always sent as only-in-chain.

scope of commands. In the NetView program, the facility that provides the ability to assign different responsibilities to various operators.

screen. An illuminated display surface; for example, the display surface of a CRT or plasma panel. Contrast with *panel*.

scroll. To move all or part of the display image vertically to display data that cannot be observed within a single display image. See also *page (2)*.

secondary half-session. In SNA, the half-session that receives the session-activation request. See also *secondary logical unit (SLU)*. Contrast with *primary half-session*.

secondary logical unit (SLU). In SNA, the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. An LU may contain secondary and primary half-sessions for different active LU-LU sessions. Contrast with *primary logical unit (PLU)*.

secondary logical unit (SLU) key. A key-encrypting key used to protect a session cryptography key during its transmission to the secondary half-session.

segment. (1) In the IBM Token-Ring Network, a section of cable between components or devices on the network. A segment may consist of a single patch cable, multiple patch cables connected together, or a combination of building cable and patch cables connected together. (2) See *link connection segment*.

send pacing. In SNA, pacing of message units that a component is sending. See also *receive pacing*.

sequence number. A number assigned to a particular frame or packet to control the transmission flow and receipt of data.

service point (SP). An entry point that supports applications that provide network management for resources not under the direct control of itself as an entry point. Each resource is either under the direct control of another entry point or not under the direct control of any entry point. A service point accessing these resources is not required to use SNA sessions (unlike a focal point). A service point is needed when entry point support is not yet available for some network management function.

service point command service (SPCS). An extension of the command facility in the NetView program that allows the host processor to communicate with a service point by using the communication network management (CNM) interface.

session. In SNA, a logical connection between two network addressable units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. Each session is uniquely identified in a transmission header (TH) by a pair of network addresses, identifying the origin and destination NAUs of any transmissions exchanged during the session. See *half-session*, *LU-LU session*, *SSCP-LU session*, *SSCP-PU session*, and *SSCP-SSCP session*. See also *LU-LU session type* and *PU-PU flow*.

session control (SC). In SNA, (1) One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification. (2) A request unit (RU) category used for requests and responses exchanged between the session control components of a session and for session activation and deactivation requests and responses.

session data. Data about a session, collected by the NetView program, that consists of session awareness data, session trace data, and session response time data.

session-initiation request. In SNA, an Initiate or logon request from a logical unit (LU) to a control point (CP) that an LU-LU session be activated.

session-level pacing. In SNA, a flow control technique that permits a receiver to control the data transfer rate (the rate at which it receives request units) on the normal flow. It is used to prevent overloading a receiver with unprocessed requests when the sender can generate requests faster than the receiver can process them. See also *pacing* and *virtual route pacing*.

session monitor. The component of the NetView program that collects and correlates session-related data and provides online access to this information.

session services. In SNA, one of the types of network services in the control point (CP) and in the logical unit (LU). These services provide facilities for an LU or a network operator to request that the SSCP initiate or terminate sessions between logical units. See *configuration services*, *maintenance services*, and *management services*.

shared. Pertaining to the availability of a resource to more than one use at the same time.

simulated logon. A session-initiation request generated when a VTAM application program issues a SIMLOGON macroinstruction. The request specifies a logical unit (LU) with which the application program wants a session in which the requesting application program will act as the primary logical unit (PLU).

single-domain network. In SNA, a network with one system services control point (SSCP). Contrast with *multiple-domain network*.

SLU. Secondary logical unit.

SMF. System management facility.

SNA. Systems Network Architecture.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network addressable units (NAUs), boundary function components, and the path control network.

solicited message. A response from VTAM to a command entered by a program operator. Contrast with *unsolicited message*.

SP. Service point.

SPCS. Service point command service.

span. In the NetView program, a user-defined group of network resources within a single domain. Each major or minor node is defined as belonging to one or more spans. See also *span of control*.

span of control. The total network resources over which a particular network operator has control. All the network resources listed in spans associated through profile definition with a particular network operator are within that operator's span of control.

SSCP. System services control point.

SSCP-LU session. In SNA, a session between a system services control point (SSCP) and a logical unit (LU); the session enables the LU to request the SSCP to help initiate LU-LU sessions.

SSCP-PU session. In SNA, a session between a system services control point (SSCP) and a physical unit (PU); SSCP-PU sessions allow SSCPs to send requests to and receive status information from individual nodes in order to control the network configuration.

SSCP-SSCP session. In SNA, a session between the system services control point (SSCP) in one domain and the SSCP in another domain. An SSCP-SSCP session is used to initiate and terminate cross-domain LU-LU sessions.

SSP. System Support Programs (IBM licensed program). Its full name is Advanced Communications Function for System Support Programs. Synonymous with *ACF/SSP*.

statement. A language syntactic unit consisting of an operator, or other statement identifier, followed by one or more operands. See *definition statement*.

station. (1) One of the input or output points of a network that uses communication facilities; for example, the telephone set in the telephone system or the point where the business machine interfaces with the channel on a leased private line. (2) One or more computers, terminals, or devices at a particular location.

status monitor. A component of the NetView program that collects and summarizes information on the status of resources defined in a VTAM domain.

subarea. A portion of the SNA network consisting of a subarea node, any attached peripheral nodes, and their associated resources. Within a subarea node, all network addressable units, links, and adjacent link stations (in attached peripheral or subarea nodes) that are addressable within the subarea share a common subarea address and have distinct element addresses.

subarea host node. A host node that provides both subarea function and an application program interface (API) for running application programs. It provides system services control point (SSCP) functions, subarea node services, and is aware of the network configuration. See *boundary node*, *communication management configuration host node*, *data host node*, *host node*, *node*, *peripheral node*, and *subarea node*. See also *boundary function* and *node type*.

subarea node. In SNA, a node that uses network addresses for routing and whose routing tables are therefore affected by changes in the configuration of the network. Subarea nodes can provide gateway function, and boundary function support for peripheral nodes. Type 4 and type 5 nodes are subarea nodes. See *boundary node*, *host node*, *node*, *peripheral node*, and *subarea host node*. See also *boundary function* and *node type*.

subarea path control. The function in a subarea node that routes message units between network addressable units (NAUs) and provides the paths between them. See *path control* and *peripheral path control*. See also *boundary function*, *peripheral node*, and *subarea node*.

subarea PU. In SNA, a physical unit (PU) in a subarea node.

subsystem. A secondary or subordinate system, usually capable of operating independent of, or asynchronously with, a controlling system.

supervisor call (SVC). A request that serves as the interface into operating system functions, such as allocating storage. The SVC protects the operating system from inappropriate user entry. All operating system requests must be handled by SVCs.

supervisor call (SVC) instruction. An instruction that interrupts the program being executed and passes control to the supervisor so that it can perform a specific service indicated by the instruction.

suppression character. In the NetView program, a user-defined character that is coded at the beginning of a command list statement or a command to prevent the statement or command from appearing on the operator's terminal screen or in the network log.

SVC. (1) Supervisor call. (2) Switched virtual circuit.

switched virtual circuit (SVC). An X.25 circuit that is dynamically established when needed. The X.25 equivalent of a switched line.

system management facility (SMF). A standard feature of MVS that collects and records a variety of system and job-related information.

system services control point (SSCP). In SNA, a central location point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network. Multiple SSCPs, cooperating as peers, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its domain.

system services control point (SSCP) domain. The system services control point and the physical units (PUs), logical units (LUs), links, link stations and all the resources that the SSCP has the ability to control by means of activation requests and deactivation requests.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

System Support Programs (SSP). An IBM licensed program, made up of a collection of utilities and small programs, that supports the operation of the NCP.

TAF. Terminal access facility.

takeover. The process by which the failing active subsystem is released from its extended recovery facility (XRF) sessions with terminal users and replaced by an alternate subsystem. See *resource takeover*.

task. A basic unit of work to be accomplished by a computer. The task is usually specified to a control program in a multiprogramming or multiprocessing environment.

task panel. Online display from which you communicate with the program in order to accomplish the program's function, either by selecting an option provided on the panel or by entering an explicit command. See *help panel*.

TCAS. Terminal control address space.

telecommunication line. Any physical medium such as a wire or microwave beam, that is used to transmit data. Synonymous with *transmission line*.

terminal. A device that is capable of sending and receiving information over a link; it is usually equipped with a keyboard and some kind of display, such as a screen or a printer.

terminal access facility (TAF). In the NetView program, a facility that allows a network operator to control a number of subsystems. In a full-screen or operator control session, operators can control any combination of such subsystems simultaneously.

terminal control address space (TCAS). The part of TSO/VTAM that provides logon services for TSO/VTAM users.

TERMINATE. In SNA, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure to end one or more designated LU-LU sessions.

time-out. (1) (ISO) An event that occurs at the end of a predetermined period of time that began at the occurrence of another specified event. (2) A time interval allotted for certain operations to occur; for example, response to polling or addressing before system operation is interrupted and must be restarted.

time sharing option (TSO). An optional configuration of the operating system that provides conversational time sharing from remote stations.

token. A sequence of bits passed from one device to another along the token ring. When the token has data appended to it, it becomes a frame.

transmission control (TC) layer. In SNA, the layer within a half-session that synchronizes and paces session-level data traffic, checks session sequence numbers of requests, and enciphers and deciphers end-user data. Transmission control has two components: the connection point manager and session control. See also *half-session.*

transmission line. Synonym for *telecommunication line.*

TSO. Time sharing option.

type 2.1 node (T2.1 node). A node that can attach to an SNA network as a peripheral node using the same protocols as type 2.0 nodes. Type 2.1 nodes can be directly attached to one another using peer-to-peer protocols. See *end node*, *node*, and *subarea node*. See also *node type*.

type 2.1 node (T2.1 node) control point domain. The CP, its logical units (LUs), links, link stations, and all resources that it activates and deactivates.

unformatted. In VTAM, pertaining to commands (such as LOGON or LOGOFF) entered by an end user and sent by a logical unit in character form. The character-coded command must be in the syntax defined in the user's unformatted system services definition table. Synonymous with *character-coded.* Contrast with *field-formatted.*

uninterpreted name. In SNA, a character string that a system services control point (SSCP) is able to convert into the network name of a logical unit (LU). Typically, an uninterpreted name is used in a logon or Initiate request from a secondary logical unit (SLU) to identify the primary logical unit (PLU) with which the session is requested.

unsolicited message. A message, from VTAM to a program operator, that is unrelated to any command entered by the program operator. Contrast with *solicited message*.

upstream. In the direction of data flow from the end user to the host. Contrast with *downstream*.

user. Anyone who requires the services of a computing system.

user-application network. A configuration of data processing products, such as processors, controllers, and terminals, established and operated by users for the purpose of data processing or information exchange, which may use services offered by communication common carriers or telecommunication Administrations. Contrast with *public network*.

USERVAR. Contains an application name used to route a session-establishment request to the currently active application subsystem.

value. (1) (TC97) A specific occurrence of an attribute, for example, "blue" for the attribute "color." (2) A quantity assigned to a constant, a variable, a parameter, or a symbol.

variable. In the NetView program, a character string beginning with & that is coded in a command list and is assigned a value during execution of the command list.

verb. (1) In SNA, the general name for a transaction program's request for communication services. (2) In VTAM, a programming language element in the logical unit (LU) 6.2 application program interface (API) that causes an LU 6.2 function to be performed.

virtual disk. (1) A logical subdivision (or all) of a physical disk pack in the VM operating system that has its own virtual device address, consecutive virtual cylinders, and a volume table of contents (VTOC) or disk label identifier. (2) Synonymous with *minidisk*.

Virtual Machine (VM). A licensed program whose full name is the Virtual Machine/System Product (VM/SP). It is a software operating system that manages the resources of a real processor to provide virtual machines to end users. As a time-sharing system control program, it consists of the virtual machine control program (CP), the conversational monitor system (CMS), the group control system (GCS), and the interactive problem control system (IPCS).

virtual route (VR). In SNA, a logical connection (1) between two subarea nodes that is physically realized as a particular explicit route, or (2) that is contained wholly within a subarea node for intranode sessions. A virtual route between distinct subarea nodes imposes a transmission priority on the underlying explicit route, provides flow control through virtual-route pacing, and provides data integrity through sequence numbering of path information units (PIUs). See also *explicit route (ER), path*, and *route extension*.

virtual route (VR) pacing. In SNA, a flow control technique used by the virtual route control component of path control at each end of a virtual route to control the rate at which path information units (PIUs) flow over the virtual route. VR pacing can be adjusted according to traffic congestion in any of the nodes along the route. See also *pacing* and *session-level pacing*.

virtual route selection exit routine. In VTAM, an optional installation exit routine that modifies the list of virtual routes associated with a particular class of service before a route is selected for a requested LU-LU session.

Virtual Storage Extended (VSE). An IBM licensed program whose full name is the Virtual Storage Extended/Advanced Function. It is a software operating system controlling the execution of programs.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VM. Virtual Machine operating system. Its full name is Virtual Machine/System Product. Synonymous with *VM/SP*.

VM/SP. Virtual Machine/System Product operating system. Synonym for *VM*.

VR. Virtual route.

VSE. Virtual Storage Extended operating system. Synonymous with *VSE/AF*.

VSE/AF. Virtual Storage Extended/Advanced Function operating system. Synonym for *VSE*.

VTAM. Virtual Telecommunications Access Method (IBM licensed program). Its full name is Advanced Communications Function for the Virtual Telecommunications Access Method. Synonymous with *ACF/VTAM*.

VTAM operator command. A command used to monitor or control a VTAM domain. See also *definition statement*.

Bibliography

NetView Publications

Learning About NetView: Operator Training (SK2T-0292) is an interactive PC-based operator training package that teaches SNA and basic network management concepts to new and inexperienced NetView operators. This training package uses graphics, animation and interactive NetView product simulations in a series of lessons to teach the basics of NetView operation.

NetView Installation and Administration Guide (SC31-6018) helps system programmers install and prepare the NetView program for operation. It is arranged in a simplified, step-by-step style and is meant to be used in conjunction with the sample network documented in *Network Program Products Samples*.

NetView Administration Reference (SC31-6014) is for system programmers and network operators who need a complete understanding of the NetView resource definition statements. This book lists each statement in alphabetical order giving its purpose and location.

NetView Tuning Guide (SC31-6079)[1] describes methods for controlling and improving the performance of the NetView Release 3 program. It is designed for system programmers who need to understand how NetView tuning values are determined and optimized.

NetView Customization Guide (SC31-6016) is designed for system programmers and others who want to customize the NetView program to reflect their network's needs or operating procedures. This book focuses on the different application programming interfaces that can be customized and explains how to modify NetView help panels and problem determination displays.

NetView Customization: Using PL/I and C (SC31-6037) describes the ways system programmers can tailor the NetView program to satisfy unique requirements or operating procedures. It discusses the uses and advantages of user-written programs (exit routines, command processors, and subtasks). It also provides instructions in designing, writing, and installing user-written programs in PL/I and C.

NetView Customization: Using Assembler (SC31-6078) describes the ways system programmers can tailor the NetView program to satisfy unique requirements or operating procedures. It discusses the uses and advantages of user-written programs (exit routines, command processors, and subtasks). It also provides instructions in designing, writing, and installing user-written programs in Assembler.

NetView Customization: Writing Command Lists (SC31-6015) explains how to simplify network operator tasks by using command lists. It provides step-by-step instructions for writing simple and advanced command lists and for migrating from NCCF message automation to NetView message automation.

NetView Operation Primer (SC31-6020) provides a basic description of the network management task for new network operators. Topics include starting and stopping a network, controlling resources, monitoring a network, and gathering the necessary data to report a problem.

NetView Operation (SC31-6019) provides system programmers and experienced network operators a comprehensive explanation of network management using the NetView program. Topics include detailed command explanation and panel flows, as well as information on how the various components interact with each other.

NetView Command Summary (SX75-0026) is a reference card that provides network operators with the format of all the commands and the commonly used NetView command lists. The commands are listed in alphabetical order by component.

NetView Problem Determination and Diagnosis (LY43-0001) aids system programmers in identifying a NetView problem, classifying it, and describing it to an IBM Support Center.

NetView Problem Determination Supplement for Management Services Major Vectors 0001 and 0025 (LD21-0023) describes major vectors 0001 and 0025 for system programmers and network operators involved in problem determination or diagnosis. The supplement may be used for the generic alert option and other problem determination tasks.

NetView Resource Alerts Reference (SC31-6024) lists the messages sent by NetView-supported hardware and software resources. It helps system programmers analyze the messages into their component parts: action codes, event types, message text, and qualifiers. The book is a reference for those who need more information than online help provides.

[1] When available.

NetView Storage Estimates (SK2T-1988) is an interactive PC-based tool that helps the user estimate storage requirements for NetView. This tool can be used for planning, installation, and tuning purposes. It is intended for network planners, system programmers, and IBM service personnel.

Console Automation Using NetView: Planning (SC31-6058) describes an approach to automate the way a system handles messages and responses to alerts. It includes information you should know before beginning such automation, as well as sample plans and proposals you might find useful in promoting your automation concept. This book includes planning information for MVS, VM, and VSE users of the NetView program.

NetView/PC Publications

NetView/PC Planning, Installation, and Customization (SC31-6002) provides planning, installation, and customization information on NetView/PC and explains the communication requirements upstream to the host and downstream to supported devices. Information relating to the required PC environment and host products that support NetView/PC is also provided. It also discusses topics that are of general interest when you are ordering your equipment.

NetView/PC Application Program Interface/Communications Services Reference (SC31-6004) is a reference for OS/2 programmers who use the API/CS and for system programmers who write command processors to run under NetView. The API/CS provides a means for vendor and other external applications to use the communication services of NetView/PC.

NetView/PC Operation (SC31-6003) describes how to operate the program and diagnose problems in NetView/PC.

NetView/PC Quick Reference (SX75-0016) describes all of the functions of the F-keys throughout the NetView/PC program.

Other Network Program Products Publications

For more information about the books listed in this section, see *Bibliography and Master Index for NetView, NCP, and VTAM.*

Network Program Products General Information (GC30-3350)

Network Program Products Planning (SC30-3351)

Network Program Products Samples (SC30-3352)

Bibliography and Master Index for NetView, NCP, and VTAM (GC31-6081)[2]

VTAM Publications

The following list shows the books for VTAM V3R2. For information about the books for VTAM V3R1, V3R1.1, or V3R1.2, see any VTAM V3R2 book or the *Network Program Products Bibliography and Master Index.*

VTAM Installation and Resource Definition (SC23-0111)

VTAM Customization (LY30-5614)

VTAM Directory of Programming Interfaces for Customers (GC31-6403)

VTAM Operation (SC23-0113)

VTAM Messages and Codes (SC23-0114)

VTAM Programming (SC23-0115)

VTAM Programming for LU 6.2 (SC30-3400)

VTAM Diagnosis Guide (LY30-5601)

VTAM Data Areas for MVS (LY30-5592)

VTAM Data Areas for VM (LY30-5593)

VTAM Data Areas for VSE (LY30-5594)

VTAM Reference Summary (LY30-5600)

NCP, SSP, and EP Publications

The following list shows the related books for NCP V4 and NCP V5.

NCP, SSP, and EP Generation and Loading Guide (SC30-3348)

NCP, SSP, and Related Products Directory of Programming Interfaces for Customers (GC31-6202)

NCP Migration Guide (SC30-3252 for NCP V4 and SC30-3440 for NCP V5)

[2] This book will be available by December 1989.

NCP, SSP, and EP Resource Definition Guide (SC30-3349 for NCP V4 and SC30-3447 for NCP V5)

NCP, SSP, and EP Resource Definition Reference (SC30-3254 for NCP V4 and SC30-3448 for NCP V5)

NCP and EP Reference Summary and Data Areas (LY30-5570 for NCP V4 and LY30-5603 for NCP V5)

NCP Customization Guide (LY30-5571 for NCP V4 LY30-5606 for NCP V5)

NCP Customization Reference (LY30-5612 for NCP V4 and LY30-5607 for NCP V5)

SSP Customization (LY43-0021)

NCP, SSP, and EP Messages and Codes (SC30-3169)

NCP, SSP, and EP Diagnosis Guide (LY30-5591)

NCP and EP Reference (LY30-5569 for NCP V4 and LY30-5605 for NCP V5)

Related Publications

Planning for a 9370 SNA Distributed Network (GC30-3475)

MVS System Programming Library: System Macros and Facilities (GC28-1151)

VM/SP System Product Interpreter User's Guide (referred to in this book as *REXX User's Guide*) (SC24-5238)

VM/SP System Product Interpreter Reference (referred to in this book as *REXX Reference*) (SC24-5239)

TSO/E REXX User's Guide (referred to in this book as *REXX User's Guide*) (SC28-1882)

TSO/E REXX Reference (referred to in this book as *REXX Reference*) (SC28-1883)

Index

Reader's Comment Form

NetView™
Customization: Writing Command Lists
Release 3

Publication No. SC31-6015-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments: ______________________________

____

____

____

____

____

____

What is your occupation? ______________________________

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC31-6015-0

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E15
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®