IBM Tivoli NetView for z/OS
Version 6 Release 1

Programming: REXX and the NetView Command List Language

SC27-2861-01

IBM Tivoli NetView for z/OS
Version 6 Release 1

Programming: REXX and the NetView Command List Language

SC27-2861-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 203.

This edition applies to version 6, release 1 of IBM Tivoli NetView for z/OS (product number 5697-NV6) and to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

This edition replaces SC27-2861-00.

Contents

Figures

About this publication

The IBM® Tivoli® NetView® for z/OS® product provides advanced capabilities that you can use to maintain the highest degree of availability of your complex, multi-platform, multi-vendor networks and systems from a single point of control. This publication, *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*, describes how to write command lists for the NetView product using either the Restructured Extended Executor (REXX) language or the NetView command list language.

Note: This publication does not provide descriptions of NetView commands. If a command is not familiar, see the NetView online help.

Intended audience

This publication is for system programmers and operators who write or run REXX programs under NetView control. A REXX program can be used as a regular command, as a subroutine, as a function from either a regular command or a subroutine, or as Data REXX. Readers should be familiar with how the NetView program is used in their network.

Publications

This section lists publications in the IBM Tivoli NetView for z/OS library and related documents. It also describes how to access Tivoli publications online and how to order Tivoli publications.

IBM Tivoli NetView for z/OS library

The following documents are available in the IBM Tivoli NetView for z/OS library:

- *Administration Reference*, SC27-2869, describes the NetView program definition statements required for system administration.
- *Application Programmer's Guide*, SC27-2870, describes the NetView program-to-program interface (PPI) and how to use the NetView application programming interfaces (APIs).
- *Automation Guide*, SC27-2846, describes how to use automated operations to improve system and network efficiency and operator productivity.
- *Command Reference Volume 1 (A-N)*, SC27-2847, and *Command Reference Volume 2 (O-Z)*, SC27-2848, describe the NetView commands, which can be used for network and system operation and in command lists and command procedures.
- *Customization Guide*, SC27-2849, describes how to customize the NetView product and points to sources of related information.
- *Data Model Reference*, SC27-2850, provides information about the Graphic Monitor Facility host subsystem (GMFHS), SNA topology manager, and MultiSystem Manager data models.
- *Installation: Configuring Additional Components*, GC27-2851, describes how to configure NetView functions beyond the base functions.
- *Installation: Configuring Graphical Components*, GC27-2852, describes how to install and configure the NetView graphics components.

- *Installation: Configuring the GDPS Active/Active Continuous Availability Solution*, SC14-7477, describes how to configure the NetView functions that are used with the GDPS Active/Active Continuous Availability solution.
- *Installation: Configuring the NetView Enterprise Management Agent*, GC27-2853, describes how to install and configure the NetView for z/OS Enterprise Management Agent.
- *Installation: Getting Started*, GI11-9443, describes how to install and configure the base NetView functions.
- *Installation: Migration Guide*, GC27-2854, describes the new functions that are provided by the current release of the NetView product and the migration of the base functions from a previous release.
- *IP Management*, SC27-2855, describes how to use the NetView product to manage IP networks.
- *Messages and Codes Volume 1 (AAU-DSI)*, GC27-2856, and *Messages and Codes Volume 2 (DUI-IHS)*, GC27-2857, describe the messages for the NetView product, the NetView abend codes, the sense codes that are included in NetView messages, and generic alert code points.
- *Programming: Assembler*, SC27-2858, describes how to write exit routines, command processors, and subtasks for the NetView product using assembler language.
- *Programming: Pipes*, SC27-2859, describes how to use the NetView pipelines to customize a NetView installation.
- *Programming: PL/I and C*, SC27-2860, describes how to write command processors and installation exit routines for the NetView product using PL/I or C.
- *Programming: REXX and the NetView Command List Language*, SC27-2861, describes how to write command lists for the NetView product using the Restructured Extended Executor language (REXX) or the NetView command list language.
- *Resource Object Data Manager and GMFHS Programmer's Guide*, SC27-2862, describes the NetView Resource Object Data Manager (RODM), including how to define your non-SNA network to RODM and use RODM for network automation and for application programming.
- *Security Reference*, SC27-2863, describes how to implement authorization checking for the NetView environment.
- *SNA Topology Manager Implementation Guide*, SC27-2864, describes planning for and implementing the NetView SNA topology manager, which can be used to manage subarea, Advanced Peer-to-Peer Networking, and TN3270 resources.
- *Troubleshooting Guide*, GC27-2865, provides information about documenting, diagnosing, and solving problems that occur in the NetView product.
- *Tuning Guide*, SC27-2874, provides tuning information to help achieve certain performance goals for the NetView product and the network environment.
- *User's Guide: Automated Operations Network*, SC27-2866, describes how to use the NetView Automated Operations Network (AON) component, which provides event-driven network automation, to improve system and network efficiency. It also describes how to tailor and extend the automated operations capabilities of the AON component.
- *User's Guide: NetView*, SC27-2867, describes how to use the NetView product to manage complex, multivendor networks and systems from a single point.
- *User's Guide: NetView Enterprise Management Agent*, SC27-2876, describes how to use the NetView Enterprise Management Agent.
- *User's Guide: NetView Management Console*, SC27-2868, provides information about the NetView management console interface of the NetView product.

- *Licensed Program Specifications*, GC31-8848, provides the license information for the NetView product.
- *Program Directory for IBM Tivoli NetView for z/OS US English*, GI11-9444, contains information about the material and procedures that are associated with installing the IBM Tivoli NetView for z/OS product.
- *Program Directory for IBM Tivoli NetView for z/OS Japanese*, GI11-9445, contains information about the material and procedures that are associated with installing the IBM Tivoli NetView for z/OS product.
- *Program Directory for IBM Tivoli NetView for z/OS Enterprise Management Agent*, GI11-9446, contains information about the material and procedures that are associated with installing the IBM Tivoli NetView for z/OS Enterprise Management Agent.
- *IBM Tivoli NetView for z/OS V6R1 Online Library*, LCD7-4913, contains the publications that are in the NetView for z/OS library. The publications are available in PDF, HTML, and BookManager® formats.

Technical changes that were made to the text since Version 6.1 are indicated with a vertical bar (|) to the left of the change.

Related publications

For more information about REXX, refer to the TSO/E REXX library.

You can find additional product information on the NetView for z/OS web site at http://www.ibm.com/software/tivoli/products/netview-zos/.

For information about the NetView Bridge function, see *Tivoli NetView for OS/390 Bridge Implementation*, SC31-8238-03 (available only in the V1R4 library).

Accessing terminology online

The IBM Terminology web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology web site at http://www.ibm.com/software/globalization/terminology/.

For NetView for z/OS terms and definitions, see the IBM Terminology web site. The following terms are used in this library:

NetView
For the following products:
- Tivoli NetView for z/OS version 6 release 1
- Tivoli NetView for z/OS version 5 release 4
- Tivoli NetView for z/OS version 5 release 3
- Tivoli NetView for z/OS version 5 release 2
- Tivoli NetView for z/OS version 5 release 1
- Tivoli NetView for OS/390® version 1 release 4

CNMCMD
For the CNMCMD member and the members that are included in it using the %INCLUDE statement

CNMSTYLE
For the CNMSTYLE member and the members that are included in it using the %INCLUDE statement

PARMLIB
For SYS1.PARMLIB and other data sets in the concatenation sequence

MVS™ For z/OS operating systems

MVS element
For the base control program (BCP) element of the z/OS operating system

VTAM®
For Communications Server - SNA Services

IBM Tivoli Network Manager
For either of these products:
- IBM Tivoli Network Manager
- IBM Tivoli OMNIbus and Network Manager

IBM Tivoli Netcool/OMNIbus
For either of these products:
- IBM Tivoli Netcool/OMNIbus
- IBM Tivoli OMNIbus and Network Manager

Unless otherwise indicated, references to programs indicate the latest version and release of the programs. If only a version is indicated, the reference is to all releases within that version.

When a reference is made about using a personal computer or workstation, any programmable workstation can be used.

Using NetView for z/OS online help

The following types of NetView for z/OS mainframe online help are available, depending on your installation and configuration:
- General help and component information
- Command help
- Message help
- Sense code information
- Recommended actions

Using LookAt to look up message explanations

LookAt is an online facility that you can use to look up explanations for most of the IBM messages you encounter, and for some system abends and codes. Using LookAt to find information is faster than a conventional search because, in most cases, LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message explanations for z/OS elements and features, z/VM®, VSE/ESA, and Clusters for AIX® and Linux systems:
- The Internet. You can access IBM message explanations directly from the LookAt web site at http://www.ibm.com/systems/z/os/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e system to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX System Services running OMVS).
- Your Microsoft Windows workstation. You can install LookAt directly from the *z/OS Collection* (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T-4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS command line) version can still be used from the directory in which you install the Windows version of LookAt.

- Your wireless handheld device. You can use the LookAt Mobile Edition from http://www.ibm.com/systems/z/os/zos/bkserv/lookat/lookatm.html with a handheld device that has wireless access and an Internet browser.

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from the following locations:

- A CD in the *z/OS Collection* (SK3T-4269).
- The *z/OS and Software Products DVD Collection* (SK3T-4271).
- The LookAt web site. Click **Download** and then select the platform, release, collection, and location that you want. More information is available in the LOOKAT.ME files that is available during the download process.

Accessing publications online

The documentation DVD, *IBM Tivoli NetView for z/OS V6R1 Online Library*, SK2T-6175, contains the publications that are in the product library. The publications are available in PDF, HTML, and BookManager formats. Refer to the readme file on the DVD for instructions on how to access the documentation.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center web site at http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that enables Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.
2. Select your country from the list and click **Go**.
3. Click **About this site** to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

For additional information, see the Accessibility appendix in the *User's Guide: NetView*.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education web site at http://www.ibm.com/software/tivoli/education.

Tivoli user groups

Tivoli user groups are independent, user-run membership organizations that provide Tivoli users with information to assist them in the implementation of Tivoli Software solutions. Through these groups, members can share information and learn from the knowledge and experience of other Tivoli users.

Access the Tivoli Users Group at http://www.tivoli-ug.org.

Downloads

Clients and agents, NetView product demonstrations, and several free NetView applications can be downloaded from the NetView for z/OS support web site:

http://www.ibm.com/software/sysmgmt/products/support/IBMTivoliNetViewforzOS.html

In the "IBM Tivoli for NetView for z/OS support" pane, click **Download** to go to a page where you can search for or select downloads.

These applications can help with the following tasks:

- Migrating customization parameters and initialization statements from earlier releases to the CNMSTUSR member and command definitions from earlier releases to the CNMCMDU member.
- Getting statistics for your automation table and merging the statistics with a listing of the automation table
- Displaying the status of a job entry subsystem (JES) job or canceling a specified JES job
- Sending alerts to the NetView program using the program-to-program interface (PPI)
- Sending and receiving MVS commands using the PPI
- Sending Time Sharing Option (TSO) commands and receiving responses

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online

Access the Tivoli Software Support site at http://www.ibm.com/software/sysmgmt/products/support/index.html?ibmprd=tivman. Access the IBM Software Support site at http://www.ibm.com/software/support/probsub.html.

IBM Support Assistant

The IBM Support Assistant is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The Support Assistant provides quick access to support-related

information and serviceability tools for problem determination. To install the Support Assistant software, go to http://www.ibm.com/software/support/isa/.

Troubleshooting information
For more information about resolving problems with the NetView for z/OS product, see the *IBM Tivoli NetView for z/OS Troubleshooting Guide*. Additional support for the NetView for z/OS product is available through the NetView user group on Yahoo at http://groups.yahoo.com/group/NetView/. This support is for NetView for z/OS customers only, and registration is required. This forum is monitored by NetView developers who answer questions and provide guidance. When a problem with the code is found, you are asked to open an official problem management record (PMR) to obtain resolution.

Conventions used in this publication

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, and command syntax.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents...

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

For workstation components, this publication uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace **$***variable* with **%***variable***%** for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Syntax diagrams

This section describes how syntax elements are shown in syntax diagrams. Read syntax diagrams from left-to-right, top-to-bottom, following the horizontal line (the main path).

Symbols

The following symbols are used in syntax diagrams:

►► Marks the beginning of the command syntax.

► Indicates that the command syntax is continued.

| Marks the beginning and end of a fragment or part of the command syntax.

►◄ Marks the end of the command syntax.

Parameters

The following types of parameters are used in syntax diagrams:

Required Required parameters are shown on the main path.

Optional Optional parameters are shown below the main path.

Default Default parameters are shown above the main path. In parameter descriptions, default parameters are underlined.

Syntax diagrams do not rely on highlighting, brackets, or braces. In syntax diagrams, the position of the elements relative to the main syntax line indicates whether an element is required, optional, or the default value.

Parameters are classified as keywords or variables. Keywords are shown in uppercase letters. Variables, which represent names or values that you supply, are shown in lowercase letters and are either italicized or, in NetView help and BookManager publications, displayed in a differentiating color.

In the following example, the USER command is a keyword, the *user_id* parameter is a required variable, and the *password* parameter is an optional variable.

Punctuation and parentheses

You must include all punctuation that is shown in the syntax diagram, such as colons, semicolons, commas, minus signs, and both single and double quotation marks.

When an operand can have more than one value, the values are typically enclosed in parentheses and separated by commas. For a single value, the parentheses typically can be omitted. For more information, see "Multiple operands or values" on page xviii.

If a command requires positional commas to separate keywords and variables, the commas are shown before the keywords or variables.

When examples of commands are shown, commas are also used to indicate the absence of a positional operand. For example, the second comma indicates that an optional operand is not being used:

```
COMMAND_NAME opt_variable_1,,opt_variable_3
```

You do not need to specify the trailing positional commas. Trailing positional and non-positional commas either are ignored or cause a command to be rejected. Restrictions for each command state whether trailing commas cause the command to be rejected.

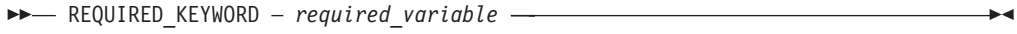
Abbreviations

Command and keyword abbreviations are listed in synonym tables after each command description.

Syntax examples

This section show examples for the different uses of syntax elements.

Required syntax elements: Required keywords and variables are shown on the main syntax line. You must code required keywords and variables.



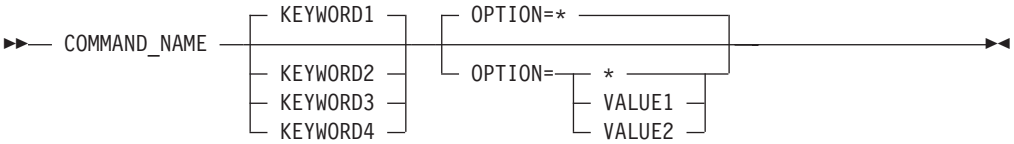
A required choice (two or more items) is shown in a vertical stack on the main path. The items are shown in alphanumeric order.

Optional syntax elements: Optional keywords and variables are shown below the main syntax line. You can choose not to code optional keywords and variables.

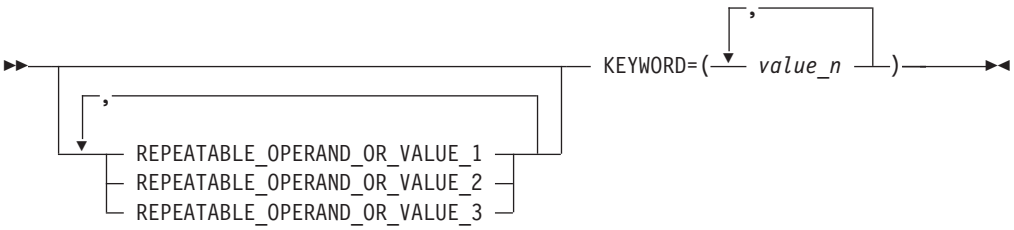
A required choice (two or more items) is shown in a vertical stack below the main path. The items are shown in alphanumeric order.

Default keywords and values: Default keywords and values are shown above the main syntax line in one of the following ways:

- A default keyword is shown only above the main syntax line. You can specify this keyword or allow it to default. The following syntax example shows the default keyword KEYWORD1 above the main syntax line and the rest of the optional keywords below the main syntax line.
- If an operand has a default value, the operand is shown both above and below the main syntax line. A value below the main syntax line indicates that if you specify the operand, you must also specify either the default value or another value shown. If you do not specify the operand, the default value above the main syntax line is used. The following syntax example shows the default values for operand OPTION=* above and below the main syntax line.



Multiple operands or values: An arrow returning to the left above a group of operands or values indicates that more than one can be selected or that a single one can be repeated.



Syntax that is longer than one line: If a diagram is longer than one line, each line that is to be continued ends with a single arrowhead and the following line begins with a single arrowhead.

Syntax fragments: Some syntax diagrams contain syntax fragments, which are used for lengthy, complex, or repeated sections of syntax. Syntax fragments follow the main diagram. Each syntax fragment name is mixed case and is shown in the main diagram and in the heading of the fragment. The following syntax example shows a syntax diagram with two fragments that are identified as Fragment1 and Fragment2.

Fragment1

Fragment2

Chapter 1. Getting Started

The Tivoli NetView for z/OS program can be used to manage complex, multivendor networks and systems from a single point. A command list is a set of commands and special instructions that are grouped under one name, like a computer program. For the Tivoli NetView for z/OS program, a command list can be written in either Restructured Extended Executor language (REXX) or the NetView command list language.

REXX has many functions and enhancements that are not available in 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 processed. You can also run command lists in other ways. For example, you can issue a timer command to run a command list at a specified time or at time intervals. You can also run more than one command list at the same time under different tasks. See "Running Command Lists" on page 6 for more information.

This chapter describes how to:
- Create command lists
- Run command lists
- Use command lists

The Benefits of Using Command Lists

Command lists help you to automate and manage your network and to improve the efficiency of operators. Command lists obtain information from operators, other tasks, system resources, or messages. The command list uses this information to perform processing or to decide the next action. With this flexibility, you can automate repetitive or complex operations, perform resource recovery, and handle operations consistently among different operators. For example, system programmers or operators can write command lists to:

- Automatically issue command lists at a specified time or time interval using the NetView timer commands AT, EVERY, CHRON, and AFTER.
- Under certain conditions, reword, delete, or reply to a message before the operator sees it.
- Provide for command lists to be issued automatically when specific messages or management services units (MSUs) are received during the operation of systems, networks, and applications.
- Wait for the NetView program to receive a message or group of messages and act based on the message content.
- Speed backup and recovery procedures, for example, automatic recovery of a failing resource.
- Monitor and restart subsystems and programs (for example, VTAM, CICS®, and TSO).
- Display information about an operator screen.
- Simplify the entry of operator commands.
- Tailor operator commands and procedures for your network.

- Ensure completeness and correct order when a sequence of commands must be issued.
- Implement specialized operator dialogs that extend the role of the operator or increase the efficiency and productivity of operators.

Before you write a command list, analyze your system, network operating procedures, and the tasks that operators regularly perform. Decide which of these jobs you want to perform using command lists. Start by writing simple command lists and add the more complex functions as you gain experience.

Note: This document does not describe how to use NetView operator commands. If you need information about a specific command, refer to the NetView online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)*.

Examples of Common Startup Command Lists

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

Figure 1, written in REXX, establishes 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. STARTUP1 Command List

Instead of having to remember and enter three commands, operators can enter the command list name STARTUP1. The command list starts the three sessions and operators receive the same messages they receive if they issue all three commands.

Examples of Activating a Network Control Program

You can write a command list to simplify the activation of a Network Control Program (NCP). Figure 2 is an example of a REXX command list that activates an NCP.

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

Figure 2. Example of Activating an NCP

Creating Command Lists

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

The PDS member name is the name you enter to run the command unless you define another name for the command list on a CMDSYN keyword on the CMDDEF statement. For more information about CMDSYN, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

The command list name must begin with a nonnumeric character and can be from one to eight characters. The following characters are valid: 0- 9, A - Z, and the special characters @ $ #.

After a command list is created and saved as a PDS member, it is ready for the operator to use.

Note: To avoid naming conflicts, give your user-written command lists names other than the command synonym (CMDSYN) names used for the command lists that are provided with the NetView program. Also, do not begin your command list names with any of the three-character prefixes used by the NetView program: AAU, BNH, BNJ, CNM, DSI, DUI, EZL, FLC, FKX, and FKV.

The NetView program supports command lists in data sets that are concatenated across volumes.

1. Create the data set to be used to store the command lists.
2. 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 startup procedure, code the JCL DD statement for the DSICLD as follows:

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

3. Concatenate data sets by coding the DSICLD statement as shown in Figure 3.

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

Figure 3. Example of Concatenating Data Sets with the DSICLD Statement

4. Ensure that the first command list data set defined under DSICLD has the largest block size of any concatenated command list data sets, or that the first DD statement has a DCB=(BLKSIZE=*xxxx*) statement, where *xxxx* is equal to the largest block size of the concatenated data sets.

When the Tivoli NetView for z/OS program is operating on a z/OS system and you plan to update or create command lists while it is running, define your command list data sets without secondary extents. Otherwise, a command list might be filed in a new extent. If this occurs, a secondary extent failure can occur causing error recovery and loss of a single instance of running the command list. If the error recovery succeeds and a second attempt to call the command list is made, the command list is then available.

If the data set becomes full and you must compress the data set to add more command lists, use the REACC command; for more information, see the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)*. You can also call a command list in any data set by reading the data set using the pipeline QSAM and saving the procedure in memory using INSTORE DSICLD.*clistname*; for more information, see *IBM Tivoli NetView for z/OS Programming: Pipes*.

The block size must be an even multiple of the record length and the record length must be 80. The records must be formatted as fixed or fixed block at 80.

Ensure that the block size is 3920 or less to reduce paging caused by the block size exceeding the size of a page of memory.

Controlling Access to Command Lists

You can use command authorization to specify which operators can issue a particular command list.

The following command-authorization methods are available:
- NetView command authorization table
- NETCMDS class of a System Authorization Facility product such as RACF®

Use the LIST SECOPTS command to determine which method is in effect.

The method for restricting access to commands is defined in the CNMSTYLE member and can be changed dynamically using the NetView REFRESH command.

The Tivoli NetView for z/OS program automatically checks the operator authorization for a command list before it is called. However, authorization for keywords and values must be checked in the following ways by command list statements coded for that purpose:
- The called command list can check the parameters that are passed and check the operator that called it. Then the called command list is coded to exit if it was called inappropriately.
- REXX command lists can use the AUTHCHK() function.
- You can start a command list from a PL/I or C command processor. Use the command processor to authorize a keyword and value, start the command list, and pass the operands to the command list. The command list must then be coded to verify that it was called properly. You can use the opid *S* function or the genealogy list available from the PIPE ENVDATA command to do this verification; for more information, see *IBM Tivoli NetView for z/OS Programming: Pipes*.

Generally, commands or command lists that are called from a command list are also eligible for command authorization. An exception to this rule is when a command list is called from the automation table, and AUTOSEC=BYPASS is in effect; refer to the DEFAULTS command in the NetView online help for more information. Another exception is when a command list is privileged by inclusion in the DSIAUTB member of DSIPARM and uses the AUTBYPAS function described in the *IBM Tivoli NetView for z/OS Security Reference*.

For more information about how to protect command lists from unauthorized users, refer to the *IBM Tivoli NetView for z/OS Security Reference*. For more information about writing command processors in PL/I or C, refer to the *IBM Tivoli NetView for z/OS Programming: PL/I and C* book.

Loading Command Lists into Storage

The Tivoli NetView for z/OS program provides several means to preload data into main storage to reduce I/O and improve performance. Two methods, MEMSTORE and LOADCL, are appropriate for REXX or NetView Command List Language procedures.

MEMSTORE runs as an autotask and monitors usage of various members. It loads members into storage based on frequency of use, weighted by age. Usually, MEMSTORE is started by your Style processing, with limits and exceptions appropriate to your installation having been set by your system programmer. If you change any REXX or NetView Command List Language procedures, use the

MEMSTOUT REFRESH procname command to ensure that MEMSTORE has not cached an older copy. When making many changes, as with testing, you might want to issue MEMSTOUT UNLOAD DSICLD.procname to permanently prevent MEMSTORE from loading your procedure. For more information about MEMSTORE and MEMSTOUT, see the online help for these commands; additional information can also be found in *IBM Tivoli NetView for z/OS Installation: Getting Started*.

Note: If you call a command list that has not been preloaded, the command list is loaded into main storage, run, and then dropped from main storage. Therefore, each time the command list is run, it must be retrieved from the auxiliary storage device where it resides. If you preload the command list, it can be run multiple times without having to be retrieved from auxiliary storage each time.

The action of MEMSTORE affects all standard means of accessing a member of any of the NetView standard DD names. For example, if a REXX procedure has been loaded into main storage by MEMSTORE, and you use the BROWSE command to view the member, then BROWSE shows the data previously loaded, and you see DATASET: 0 where the concatenation level is listed. Similarly, the LISTA DSICLD membname command shows an INSTORE level listing for the member. When the procedure is called, the member is "read" or copied into another location, where it is formatted for interpretation. You can avoid this copying and formatting step and obtain a small additional performance enhancement by using LOADCL. Use LOADCL only for procedures in DD=DSICLD. After LOADCL, the BROWSE command still shows the member as it exists on auxiliary storage. However, you can use the LIST CLIST command to review the member as it exists in main storage.

These NetView commands can be used to move command lists into and out of main storage, and to list command lists that are currently in main storage:

LOADCL
Works with DROPCL and MAPCL to provide information and control about procedures that have been loaded.

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

For more information about the LOADCL, DROPCL, or MAPCL commands, refer to the NetView online help.

The Tivoli NetView for z/OS program provides a REXX command list, AUTODROP (CNMS8003), that can help you manage the number of command lists that are loaded into storage using the LOADCL command. AUTODROP uses the MAPCL and DROPCL commands to conditionally drop commands from main storage.

Note: The NetView program confirms the existence of a REXX/CLIST by loading it before authorization checking occurs. This prevents an erroneous security violation, such as message ICH4081 from RACF or a system management facilities (SMF) record (or both a RACF message and an SMF record) when an invalid command is entered.

Running Command Lists

Design command lists that run with little assistance from operators. The following list shows some of the ways you can run command lists:

- When the Tivoli NetView for z/OS program is started
- When the operator logs on
- After receiving a message or MSU
- From a terminal
- At a specified time or time interval
- From another command list
- From a user-written command processor
- From the NetView management console
- From a web page
- From a remote NetView domain
- From user exits and optional tasks

Running Command Lists When NetView Is Started

You can specify that command procedures run automatically when the Tivoli NetView for z/OS program is started by defining them in the CNMSTUSR or C*xx*STGEN that is included in the CNMSTYLE member using auxInitCmd statements. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information. These commands run under the primary program operator interface task (PPT). See "Primary POI Task Restrictions" on page 12 for information about PPT restrictions.

Running Command Lists When Logging On

You can define a command list to run automatically after an operator successfully logs on. You can define only one command list to run when an operator logs on, but this command list can activate other command lists. See "Running Command Lists from Another Command List" on page 8 for rules that apply when calling another command list.

Code the name of the command list you want to run in the profile for the operator using the IC operand of the PROFILE statement, or the IC field in the NETVIEW segment of the SAF product. For example, if you want to run the HELLO command list each time an operator logs on, and if the operator has a profile of PROFBEG, the IC operand can be added to the profile for the operator in the following way:

```
PROFBEG  PROFILE  IC=HELLO
```

For more information about the PROFILE definition statement, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

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

- To start autotasks, use AUTOTASK commands. Use START commands to start other tasks, such as DSTs (data services tasks).
- To restore all task global variables that were saved using the GLOBALV SAVET command, include:

  ```
  GLOBALV RESTORET *
  ```

  Note: The RESTORET depends on two things: the task having previously performed a SAVET, and the DSISVRT DST being active.

- To set operator-specific defaults that override NetView-wide values set using the DEFAULTS command, use the OVERRIDE command.

Running Command Lists after Receiving a Message or MSU

The NetView automation table can initiate a command list upon receipt of a message or MSU. These command lists can automatically respond to the message or MSU, saving an operator from having to respond to them.

Command lists that the Tivoli NetView for z/OS program initiates upon receipt of a message or MSU can contain a series of commands to perform a function as a result of the message or MSU. For example, if the message or MSU reported that an NCP failed, the command list can issue the VTAM command to reactivate the NCP. Chapter 5, "Automation Resource Management," on page 91 contains additional information about performing NetView automation using command lists.

Running Command Lists from a Terminal

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

Running Command Lists at a Specified Time or Time Interval

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

AFTER
Instructs the Tivoli NetView for z/OS program to run the command list after a specified period.

AT Instructs the Tivoli NetView for z/OS program to run the command list at a particular time.

CHRON
Instructs the Tivoli NetView for z/OS program to run commands in varied and flexible ways.

EVERY
Instructs the Tivoli NetView for z/OS program to run the command list repeatedly at a certain time interval.

You can also issue the AFTER, AT, CHRON, and EVERY commands from a command list.

You can set up the AT, EVERY, CHRON, and AFTER commands so the command list runs even if the operator is not logged on at the time. Do this by choosing an autotask that will be active and by coding the *oper* operand of the ROUTE command in the timer command. You can run command lists under the primary POI task (PPT); restrictions in doing so are described in "Primary POI Task Restrictions" on page 12. However, whenever possible, run command lists under an autotask; restrictions in using the AUTOTASK command are described in "AUTOTASK OST Restrictions" on page 13.

You can define command lists so that they always interrupt the processing of other command lists. You do this using the TYPE=H (high priority) operand of the CMDDEF statement, or by prefixing your command with `CMD HIGH`. For more

information about how to code CMDDEF statements, see the *IBM Tivoli NetView for z/OS Administration Reference*. For more information about the CMD command, see the online help.

To learn more about the AT, EVERY, CHRON, and AFTER commands, refer to the NetView online help.

Running Command Lists from 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 theTivoli NetView for z/OS program reaches a statement with the name of a command list, it starts running the nested command list. When the end of the nested command list is reached, control is returned to the calling command list, as shown in Figure 4.

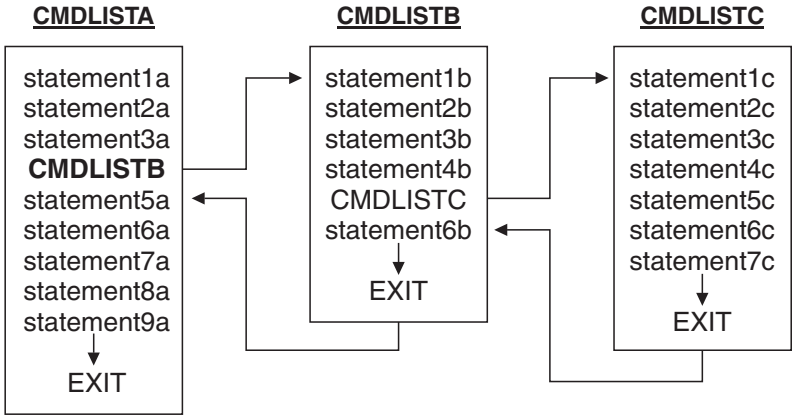


Figure 4. Nested Command Lists

When planning to create command lists that run other command lists, keep in mind the following considerations:

- A REXX command list can be called 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 either the NetView command list language or in a REXX command list as a command.
- Command lists written in REXX and command lists written in the NetView command list language can call each other.
- You can have 250 levels of externally nested command lists.

  Only REXX command lists called as commands, external subroutines, or external functions count as one of the 250 levels of externally nested command lists. You can call 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.
- Test each command list before running the command list as part of a nested chain of command lists.

For information about REXX subroutines and functions, refer to the REXX library.

Passing Information from One Command List to Another

When REXX command lists and command lists written in the NetView command list language call each other, operands 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 sent to the calling command list.

To pass additional information, you can use the pipeline SAFE, STEM, or VAR stages, or the global variable pool.

SAFE stage
NetView messages, or any other data in a pipeline, can be placed into a named SAFE stage by either the calling procedure or the called procedure. Named SAFE stages endure as long as any member of the nested family is still running. Either the calling procedure or the called procedure can add to, modify, or delete data from the SAFE. See *IBM Tivoli NetView for z/OS Programming: Pipes* for more information about the SAFE stage

KEEP stage
The pipeline KEEP stage is similar to the SAFE stage, and you can use it to define a task-global place to store messages. See *IBM Tivoli NetView for z/OS Programming: Pipes* for more information about the KEEP stage.

STEM stage and VAR stage
Using pipeline STEM and VAR stages, a called procedure can read from and write to the variables of the calling procedure, if those variable names are known. A calling procedure written in REXX can control such access using REXX built-in PROCEDURE and EXPOSE keywords. For example, the WINDOW (CNME1505) procedure calls customer-defined subcommands from an internal procedure named *exterCmd* and limits variable access to a list named *toShow*. See CNME1505 and the example subcommand CNMEXEC for additional details on using this procedure.

Global variables
Procedures can store and modify data in COMMON or TASK global variables. The GLOBALV command provides one way to do this in a manner similar to using the pipeline VAR and STEM stages. Using the COMMON or TASK global variables allows data to be shared with other procedures, whether they are called directly or called by any other means. Be sure to provide a means of verifying the accuracy of any data so shared in case any of the procedures involved encounters a failure.

Error Handling

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.

Note: A list of return codes can be found at "Return Codes in REXX Command Lists" on page 33.

If the calling command list is written in REXX, it might be able to recover from the error passed to it from the nested command list. For information about coding REXX command lists that can recover from errors, see "Recovering from Errors in REXX Command Lists" on page 33.

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.

Running Command Lists from a User-Written Command Processor

You can write a command processor that calls a command list. Command processors are programs written in assembler, PL/I, or C. For information about how to write command processors, refer to *IBM Tivoli NetView for z/OS Programming: Assembler* or *IBM Tivoli NetView for z/OS Programming: PL/I and C*.

Using Network Commands in Command Lists

This section describes how you can use network commands in command lists. The following commands are 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 subject to command authorization, unless either SEC=BY was specified on the CMDDEF statement, AUTOSEC=BYPASS is in effect, or the command list is privileged by inclusion in the DSIAUTB member of DSIPARM and uses the AUTBYPAS function described in this manual. For more information, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Notes:

1. The NetView RETURN command is not valid in a command list.
2. You can use Tivoli NetView for z/OS and user-written commands that are defined on the CMDDEF statement as regular, high, or both (TYPE=R, TYPE=H, or TYPE=B). You can also use command lists that are undefined.
3. You must use the appropriate prefix:
   - NLDM for session monitor commands
   - NPDA for hardware monitor commands
   - STATMON for status monitor commands

Using System Commands in Command Lists

You can use system commands in command lists. For example, use one of the following NetView MVS commands to enter MVS commands:

- `MVS S jobname`
- `MVS D A,L`

Refer to the Tivoli NetView for z/OS online help for more information.

Using Long-Running Commands in Command Lists

Most NetView commands and some customer-written commands block a calling procedure to wait for either operator input, data from an optional task, or some other asynchronous event. These commands are called *long-running commands*. You can use *long-running commands* in your command lists. Using long-running commands in your command list enables other commands on the low-priority queue to begin running. Refer to the NetView online help for more information about command priorities.

Long-running commands use assembler macro DSIPUSH to accomplish the required wait while allowing essential NetView functions like message automation to

continue. Additional information about the DSIPUSH macro can be found in *IBM Tivoli NetView for z/OS Programming: Assembler*. Examples of long-running commands are NPDA and VIEW.

The type of long-running command, 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 processing priority. This is described further in "Queuing Long-Running Commands."

When coding a long-running command, it is important to plan for two kinds of events that can occur during the extended period before the next instruction of your procedure:

Implied Cancellation
: The operator generally has an opportunity to enter commands (although some exceptions are noted in the *IBM Tivoli NetView for z/OS Customization Guide*). If the operator calls the same long-running command (or VIEW with the same component name), then that procedure is halted. For this reason, REXX procedures that call long-running commands always need to code SIGNAL ON HALT. At that HALT label, code an EXIT –5 in addition to any other cleanup processing that you need to do. The –5 return code propagates the halt condition to the caller of the procedure, if any. You can also exercise some degree of control over this halt condition using the UNIQUE command. See the *IBM Tivoli NetView for z/OS Customization Guide* for more information about UNIQUE.

Interruption
: Usually, your procedure is not interrupted by another command queued from message automation, operator action, a timed command, or an EXCMD from another commands (although some exceptions are noted in the command help for the DEFAULTS command). When a long-running command is called, the wait for operator input can be protracted, so the Tivoli NetView for z/OS program calls any other commands that are queued. That can affect your procedure if you use global variables or if you query and then take action on the status of any external resource.

Queuing Long-Running Commands

You can control the processing of long-running commands by using the NetView CMD command to queue them. 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 delay the processing of NLDM until your command list is stacked, canceled, interrupted, or completed, use `CMD LOW NLDM`.

Using Tivoli NetView for z/OS Pipelines

Tivoli NetView for z/OS pipelines provide another level of function and flexibility to command lists. Among many pipeline capabilities is the automation of full-screen applications. For more information, refer to *IBM Tivoli NetView for z/OS Programming: Pipes* or the NetView online help.

Using the VIEW Command

You can use the VIEW command 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 and to NetView local variables.

Refer to the *IBM Tivoli NetView for z/OS Customization Guide* for more information about local and global variables, and using the VIEW command with commands and command lists.

Using Full-Screen Commands

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

- Code a CMDDEF definition statement with ECHO=N for the command list. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about coding a CMDDEF statement.
- Code TRACE ERRORS or TRACE OFF at the beginning of a REXX command list. Refer to the REXX library for information about the TRACE instruction.
- Do not code SAY instructions in a REXX command list.
- Code &CONTROL ERR at the beginning of a command list written in the NetView command list language.
- Do not code &WRITE or &BEGWRITE control statements in a command list written in the NetView command list language.
- Do not issue commands that have line mode output.

Note: If a command list encounters a statement with a timeout greater than 30 seconds while a full-screen command processor is running, the following message is issued:

```
DSI594A COMMAND PROCEDURE cmdlistname WARNING - type STATE ENTERED
```

You can then enter the necessary information if the command list is waiting for an operator response, or ensure that a WAIT or &WAIT is satisfied before rolling to other components.

Primary POI Task Restrictions

Command lists can run under the primary POI task (PPT). However, when possible, run command lists under an autotask. See "AUTOTASK OST Restrictions" on page 13 for more information about running command lists under an autotask.

You can run command lists under the PPT when the command lists meet any of the following criteria:

- Routed to the PPT for processing as a result of NetView automation.
- Coded on a CNMSTYLE definition statement to run when the Tivoli NetView for z/OS program is initialized.

- Called with an AT, EVERY, AFTER, or EXCMD command that uses the PPT as an operand. (PPT on AT, EVERY, and AFTER enables the command to be run even when the operator who scheduled it is not logged on.)

Many commands cannot run under the PPT. These are some restrictions that apply to command lists that run under the PPT:

- In general, you cannot use full-screen commands and immediate commands. Do not use the following NetView commands:
  - AUTOWRAP
  - BGNSESS
  - INPUT
  - LOGOFF
  - ROUTE
  - SET
  - SUBMIT
  - TRAP
  - WAIT
- Do not use the following REXX instructions:
  - FLUSHQ
  - MSGREAD
  - PARSE EXTERNAL
  - PARSE PULL if nothing is in the REXX data stack
  - PULL if nothing is in the REXX data stack
- Do not use the following NetView command list language control statements:
  - &PAUSE
  - &WAIT

When command lists running under the PPT generate messages, the messages go to the authorized receiver, if any. If no authorized receiver exists, these messages go to the system console and they should not contain non-Latin characters, such as double-byte characters.

AUTOTASK OST Restrictions

This restriction applies to automation tasks that are started with the AUTOTASK command:

- Use the ATTACH command to automate full screen commands.

Because autotasks have fewer restrictions than the PPT, use them instead of the PPT whenever possible.

Controlling Command List Output

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

To control the amount of data that is displayed to the operator during the processing of a REXX command list, use the NetView PIPE CONSOLE command, TRAP instruction, or the suppression character (see “Suppressing Display of Non-REXX Commands” on page 26). Refer to the NetView online help and the REXX library for information about the TRAP instruction.

To control the amount of data that is displayed to the operator during the processing of a command list written in the NetView command list language, use

the &CONTROL statement (see “&CONTROL Statement” on page 110), the &WAIT SUPPRESS control statement (see “Customizing the &WAIT Statement” on page 141) or the suppression character (see “Conventions for Suppression Characters” on page 99).

The commands and messages that are displayed during processing of a command list are shown in the message area of the NetView command facility panel. Generally, output from the command list is preceded by a type code of `C`. For a complete description of the NetView panel layout and the format of messages sent to the panel, refer to the *IBM Tivoli NetView for z/OS User's Guide: NetView*.

Working with Messages

A *message* is an object that consists of the following parts:

- An optional reply ID
- One or more lines of text
- Attributes associated with each line of text
- Attributes associated with the message as a whole

At most, only one *current message* exists at any given time. The current message is set by the most recent occurrence of any of these events:

- Message automation. The message that triggered the automation is the current message when the procedure is called. See Chapter 5, “Automation Resource Management,” on page 91 for more information about NetView automation.
- A pipeline. When calling of the procedure is triggered by a message passed from a previous stage, then that message becomes the current message.
- For REXX, a MSGREAD that occurs after a successful TRAP of a message. For NetView CLIST language, an &WAIT. See “Control and Parameter Variables Used with &WAIT” on page 139 for more information about using control variables with &WAIT.
- When the procedure calls a NetView pipeline with SAFE * operating as a "not first" stage.
- When the results of a LINKPD command are processed. See “LINKPD Results” on page 157 for more information about the LINKPD command.

All of the previously listed means of setting a current message return data from the current message regardless of how that message was made the current message. For a multiline write-to-operator message (MLWTO), only the first line of the message is considered by these functions.

Information about the *current message* can be retrieved using the REXX functions or NetView CLIST variables (refer to “Message Processing Information Functions” on page 52), MSU functions (refer to “REXX Management Services Unit Information Functions” on page 67), or message information commands (refer to “Commands Used in Command Lists” on page 166).

Notes:

1. Usually, the first blank-delimited word from the first line of text is considered the message ID. A reply ID, if present, is ignored.
2. To test for the existence of a current message, issue this command:

   ```
   PIPE SAFE * | COUNT LINES | VAR LINECOUNT
   ```

   If variable *LINECOUNT* contains 0, no current message exists; if *LINECOUNT* contains a value other a 0, a current message does exist.

3. REXX functions that return data pertinent to the current message do so regardless of how the message was made current. For multiline messages, only the first line is considered by these functions, unless documentation for that function states otherwise.
4. When a `MSGREAD`, `&WAIT`, `PIPE SAFE *`, or `LINKPD` occurs, the resulting message becomes the current message, even if the message is null (such as when MSGREAD has RC=4).

Chapter 2. REXX Language Overview

This chapter includes a brief introduction to REXX. Not all of the features and syntax rules of REXX are described in this document. This document focuses primarily on the REXX instructions and functions provided by the Tivoli NetView for z/OS program.

Note: For complete information about REXX, refer to the TSO/E REXX library.

Introduction to the REXX Language

REXX is often used as an *interpretive* language; the REXX interpreter operates directly on the program as it runs, line-by-line and word-by-word. An interpreted language is different from other programming languages, such as COBOL, because it is not necessary to compile a REXX command list before running it. However, you can choose to compile a REXX command list before running it to reduce processing time.

Each NetView REXX command list or Data REXX file must begin with a comment. REXX comments are marked with `/*` at the beginning and `*/` at the end, and can be used in your REXX command list wherever necessary.

A REXX command list or Data REXX file 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 run 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. Use assignment clauses to give a value to a variable. Variables 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 identify the target statement for a transfer of control.

Use the REXX language to call internal or external routines, called functions. REXX function names must always be followed by parentheses. There can be up to 10 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 or Data REXX file at the location where you want the result to be accessed. Several built-in functions are also included in the REXX language that perform predefined operations. Refer to the REXX library for a complete description of the features of the REXX language.

Compiling and Running REXX Command Lists

REXX command lists can be compiled to significantly improve performance.

The IBM REXX/370 Compiler product must be installed on the system where the command lists is to be compiled.

Refer to the REXX library for directions on how to compile a REXX command list. For additional performance information about compiled REXX command lists, refer to the *IBM Tivoli NetView for z/OS Tuning Guide*.

Notes:

1. You do not need to install or start the compiler on the system where the Tivoli NetView for z/OS program resides.
2. The compiled executable file might be larger (take up more space) than the original uncompiled command list.
3. The Tivoli NetView for z/OS program supports the CEXEC (compiled EXEC) and OBJECT (object deck) output formats of the REXX/370 Compiler.
4. When creating a load module from an object deck, note the following items:
   - The object deck must be created and saved from a REXX compiler.
   - Two DDNAMEs in the REXXL cataloged procedure are particularly important:
     - The SYSIN DD statement must refer to the object deck (input).
     - The SYSLMOD DD statement must refer to the load library specified with the load module (output).
     - The object deck must be link-edited with the EFPL stub to create a load module, and the load module name cannot conflict with any NetView, REXX, or other load module name.
     - The REXXL cataloged procedure is used to create a load module; the procedure can be found in REXX.V6R1M0.EAGPRC.
     - The load module can be called only through or by a REXX CALL instruction, or as a REXX function.
5. To run CEXEC format compiled REXX command lists, place the output file into a member of one of the DSICLD data sets.
6. Install the compiler runtime library in an authorized library on the system that runs the compiled REXX command lists.

Using %INCLUDE with Interpreted REXX

To facilitate code reuse, you can create members of the DSICLD data definition with segments of REXX code and use the `%INCLUDE` capability provided by the NetView program. To include a prepared code segment, the first line of your program must begin with `/*%NETVINCL` followed by comments of your choice. Then, anywhere in your program, on a separate line, code `%INCLUDE` followed by the member name you have prepared. The prepared code segment is present when your program is interpreted, just as if it had been embedded in your program.

Do not use the INCLUDE function if you intend to compile your REXX program.

Using Data REXX

Data REXX files are special REXX programs that send data (rather than commands) to an environment external to the Data REXX program.

For any NetView application that supports %INCLUDE processing, you can substitute a Data REXX program file for the member that the application ordinarily reads. The data generated by your program becomes the data delivered to the application.

The REXX language supports an external ADDRESS environment. Generally, this is for commands to be called. In Data REXX, the only ADDRESS environment supported is NETVDATA. The character strings submitted to the NETVDATA environment become data instead of commands. This data is delivered to the application after the Data REXX program file is read and processed. No trace of the REXX language elements in the Data REXX program is seen by the application.

Note: In this discussion, the term *generated data* is used to mean data that contains REXX variables or function calls that must be evaluated before the data is ready to be sent to the receiving application. One example of a receiving application is a phase of style sheet processing during Tivoli NetView for z/OS initialization.

By definition, Data REXX files are any NetView data files that begin with the Data REXX file directives /*%DATA or /*%LOGIC in the first column of the first record. The Data REXX file directives determine the format of the Data REXX code that follows the directive. These directives provide a convenient way of switching between the logic and data formats. The following paragraphs further describe these directives.

When the /*%LOGIC Data REXX directive is in effect, data is embedded or generated in a Data REXX program the same way commands are in a traditional REXX program. This is also referred to as *logic mode*.

When the /*%DATA directive is in effect, embedded or literal data is programmed exactly as the application program expects to see it. REXX programming, including generated data clauses for the NETVDATA address environment, is written with %> in columns 1 and 2 of each record. This is also referred to as *data mode*.

Sample BNJMBDST is an example of a logic file. Sample CNMCMSYS is an example of a data file. Sample CNMSTASK uses both data and logic formats.

Data mode is a convenient way to write large sections of the Data REXX program that contain literal data that can be written exactly as the receiving application is to receive the data. Data that you program dynamically generated in whole or in part must be preceded with %> in columns 1 and 2 of the Data REXX program file. For example, to create a data record with the domain name of the Tivoli NetView for z/OS program on which your application is currently running, you might code the following statement starting in column 1:

```
%>   'My domain name is' domain()'.'
```

If the domain name of the Tivoli NetView for z/OS program on which your application is currently running is CNM01, then the following data record is read:

```
My domain name is CNM01.
```

If a Data REXX generated data record is written to be continued on the next record, include %> at the beginning of column 1 of the next record. For example, to write a continuation for the previous example that contains the version and release of the Tivoli NetView for z/OS program on which your application is currently running, code the following statements starting in column 1:

```
%>  'My domain name is' domain() 'and my NetView version is',
%> substr(netview(),3,1) 'and release is,
%> substr(netview(),4,1)'.'
```

If the domain name of the Tivoli NetView for z/OS program on which your application is currently running is CNM01, and the Tivoli NetView for z/OS program version is 5 and the release is 4, then the following data record is read:

```
My domain name is CNM01 and my NetView version is 5 and release is 4.
```

Processing Data REXX Files

When a NetView application reads a Data REXX program file, processing occurs in three steps:

1. A REXX program is loaded. A NetView application must use the NetView DSIDKS services to access Data REXX files and have them processed as Data REXX files. When DSIDKS services finds the file member, it reads the first record of the file and determines if the file begins with one of the two Data REXX directives. If it does, then the Data REXX file is loaded into storage in its entirety. As it is loaded, the file is reformatted into a form that the REXX interpreter can run. If in data mode, the records are loaded so that the %> symbols are not seen by the REXX interpreter. Records without leading %> symbols are loaded in such a way that they are presented to the NETVDATA address environment without being evaluated by the REXX interpreter.

   Records that are read in logic mode are read into storage unchanged. The Data REXX directives are not removed because they are seen by REXX as REXX comments and are ignored, but not removed. %INCLUDE statements are not processed at this time.
2. The REXX program runs to completion. After loading the REXX program, the program is passed to the REXX interpreter to run. While running, any REXX clauses that are seen by REXX as external commands are sent to the NETVDATA address environment. These commands are actually the data records that are collected by the Tivoli NetView for z/OS program and passed to the NetView application. Note that %INCLUDE statements can be generated by the Data REXX program during this step.
3. The resulting data is read by the application. After all the data records have been collected by the Tivoli NetView for z/OS program, they are passed to the NetView application. It is transparent to the NetView application whether the file being read is a Data REXX program. During this step that any %INCLUDE statements are processed as they are encountered by DSIDKS services. Included files are also checked to determine if they are Data REXX files (as determined by the first line) and are processed, independently, in the same three-step manner if they are.

During the first step, each line of the source file is processed individually. However, in the second step, REXX interprets the entire Data REXX program according to the rules of the REXX language. Thus, REXX elements on one line can affect how the following lines are interpreted. For example, consider the following Data REXX file:

```
1   /*%DATA    Put any comments you want here.        */
2   %>  x = cglobal('y');  /* getting a value for x
3   PASSWORD = XYZ
4   %>          any comment */
5   %>   IF  tower('AON') THEN
6   AUTOCMD.mytable.ORDER  = C
```

Note the absence of %> coding in lines 3 and 6.

If the `tower('AON')` function is evaluated as true, then the following record is generated by the example:

```
AUTOCMD.mytable.ORDER  = C
```

During the first step, line 3 is loaded. However, line 3 is never seen during step 3 because the REXX comment start delimiter, /*, in line 2 matches the REXX comment end delimiter, */, in line 4.

Line 6 is conditional on the previous IF-THEN evaluation, although it is not prefixed with the %> data REXX designator. Line 6 is not evaluated by the Data REXX interpreter, despite its similarity to REXX code. This line is sent exactly as written to the NETVDATA address environment (if the `tower('AON')` function is evaluated as true). The same is true if line 6 was an %INCLUDE instruction. In such a case, the %INCLUDE statement is processed by the Tivoli NetView for z/OS program after the Data REXX program completes.

Additional Information

The following information applies to both data and logic files:

- Data REXX is supported only in the following places:
  - CNMCMD
  - CNMSTYLE
  - DSICNM
  - DSIOPF
  - Automation tables
  - Code point tables
  - Command authorization tables
  - Command facility panel-format definition members, such as CNMSCNFT
  - Hardware monitor (NPDA) members (but not BNJHEADR)
  - Members read using NetView disk services with the INCL option
  - Operator profile (DSIPRF) members
  - Session monitor (NLDM) members
  - Span tables
  - HELPMAP
- Data REXX is not supported in VSAM files.
- REXX clauses that are processed as commands in a typical REXX environment are instead passed to the NETVDATA environment. As a result, these clauses are seen as data by the application reading the member.
- Data REXX files must be small because they are read entirely into storage. %INCLUDE files that are referenced in a Data REXX file are also read into storage if they are Data REXX files; files that are not Data REXX files are not read entirely into storage. Therefore, to save storage, do not use Data REXX files in %INCLUDE files.
- Output strings that result from SAY and TRACE instructions are written to the network log.
- The REXX keyword instruction ADDRESS can be used; however, only the NETVDATA address is supported.
- The following REXX functions cannot be used in Data REXX files:
  - GETMSG
  - LISTDSI
  - MSG
  - MVSVAR
  - OUTTRAP
  - PROMPT
  - SETLANG
  - STORAGE
  - SYSDSN
  - SYSVAR

- The following REXX keyword instructions cannot be used in Data REXX files:
  - TRACE ? (interactive trace)
  - PARSE EXTERNAL
- The PULL and PARSE PULL instructions can be used only to access data from the REXX data stack. Do not use PULL and PARSE PULL to pause for operator input. Data REXX has no operator input facility.
- REXX functions called from Data REXX using synonym names are not supported during NetView initialization CNMSTYLE processing.

Data REXX Directives

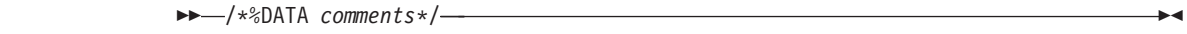
The Data REXX file directives determine the format of Data REXX code. These directives provide a convenient way of switching between the logic and data formats.

/*%DATA

The NetView REXX /*%DATA directive specifies data mode for the Data REXX code that follows it. The file remains in data mode until either the /*%LOGIC directive is encountered or the end of file (EOF) is reached.

The NetView REXX /*%DATA directive has the following syntax:

/*%DATA



where:

/*%DATA
The /*%DATA directive must begin with the slash symbol (/) in column one. A space is required after DATA and the word DATA must be in all capital letters. The file must begin with either a DATA or LOGIC directive in the first line.

comments
Specifies any comments that you want to include. Comments can span multiple lines.

*/ Specifies the end of the DATA directive.

Usage Notes

- In data mode, all REXX instructions must be preceded by %> in column one. Lines that do not begin with %> are treated as data.

Example: The following example shows how to code Data REXX in data mode:

```
/*%DATA --- demonstrate data mode          */
  Data line (this line has two leading blanks)
%>IF CGLOBAL('ABC') = 1 THEN
  %INCLUDE ABCFILE
%>ELSE
%>  DO
%>  '%INCLUDE' CGLOBAL(XYZfilenameVar)
Another data line
%>  END
Final data line
```

Related Statements: /*%LOGIC

/*%LOGIC

The NetView REXX /*%LOGIC directive specifies logic mode for the Data REXX code that follows it. The file remains in logic mode until either the /*%DATA directive is encountered or the end of file (EOF) is reached.

The NetView REXX /*%LOGIC directive has the following syntax:

/*%LOGIC

where:

/*%LOGIC
The /*%LOGIC directive must begin with the slash symbol (/) in column 1. A space is required after LOGIC and the word LOGIC must be in all capital letters. The file must begin with either a DATA or LOGIC directive in the first line.

comments
Specifies any comments that you want to include. Comments can span multiple lines.

*/ Specifies the end of the LOGIC directive.

Usage Notes

- REXX clauses are processed as commands in a normal REXX environment are instead passed to the NETVDATA environment. As a result, these clauses are seen as data by the application reading the member.
- In logic mode, REXX instructions must not be preceded by %>.

Example: The following example shows how to code Data REXX in logic mode:

```
/*%LOGIC --- demonstrate logic mode            */
'  Data line (this line has two leading blanks)'
IF CGLOBAL('ABC') = 1 THEN
  '    %INCLUDE ABCFILE'
ELSE
  DO
   '%INCLUDE' CGLOBAL(XYZfilenameVar)
   'Another data line'
  END
'Final data line'
```

Related Statements: /*%DATA

Coding Conventions for REXX Command Lists and Data REXX Files

This section describes the syntax rules that apply when coding REXX command lists or Data REXX files for the Tivoli NetView for z/OS program.

Record Size

The data portion of records in REXX command lists or Data REXX files for the Tivoli NetView for z/OS program can be up to 80 characters in length (the records must be a fixed length of 80 characters). If the first record of a REXX command list or Data REXX file contains numeric characters in columns 73 through 80, the command list or Data REXX file is assumed to contain sequence numbers. The Tivoli NetView for z/OS program removes the sequence numbers from lines

containing executable statements or Data REXX control statements. That is, sequence numbers are preserved on lines that are known to be data-only (those lines that are contained within a /*%DATA section and are not prefixed with %>). The Tivoli NetView for z/OS program also truncates trailing blanks from all REXX records in REXX command lists and Data REXX files. Blank REXX records are not discarded, but are truncated to one blank character.

Using Quotation Marks

To avoid variable substitution on a string in a REXX command list or Data REXX file in logic mode, enclose the string in either single quotation marks (') or double quotation marks ("). The quotation marks 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 beginning quotation mark (single or double) on a command list statement or Data REXX statement, it stops interpreting until it reaches a matching ending quotation mark.

Do not enclose REXX instructions in quotation marks. REXX recognizes its own instructions and does not perform variable substitution on REXX instructions. The following examples show how to use quotation marks to prevent variable substitution with the REXX SAY instruction:

```
SAY 'THIS IS A STRING WITH SINGLE QUOTATION MARKS'
SAY "THIS IS A STRING WITH DOUBLE QUOTATION MARKS"
```

These two instructions display the following text at your terminal when using REXX or write to the network log when using Data REXX:

```
THIS IS A STRING WITH SINGLE QUOTATION MARKS
THIS IS A STRING WITH DOUBLE QUOTATION MARKS
```

To use an apostrophe or double quotation marks within the text of a string enclosed in quotation marks, you can code the following lines:

```
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"""
```

In the following example, either of the first two instructions displays the first line or writes to the network log when using Data REXX. Either of the last two instructions display the second line:

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

Generally, you enclose any NetView commands, or system commands recognized by the Tivoli NetView for z/OS program, in quotation marks. 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 the quotation marks.

Note: NetView commands cannot be issued from Data REXX files. The only address environment supported by Data REXX is ADDRESS NETVDATA. When REXX clauses are treated as external commands by REXX, they are treated as external data by Data REXX.

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

```
'INACT NODE1'
```

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, code:

```
'INACT ' NODE
```

The next example uses quotation marks to have REXX perform variable substitution for only part of a command. The example assumes that the DDNAME has already been allocated. This example first parses the user 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 quotation marks. The quotation marks begin before EXECIO because it is a TSO/E command. The quotation marks end before DDNAME to enable REXX to substitute the current value of the DDNAME variable into the EXECIO command. The rest of the EXECIO command is enclosed in quotation marks so that variable substitution does not take place on the STEM and LINE operands.

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

Notes:

1. Use caution when writing REXX clauses that have quoted strings that span multiple records. Because the Tivoli NetView for z/OS program truncates trailing blanks from all REXX command list records before running the command list, REXX clauses that have quoted strings that span multiple records might not run as expected. For example, in the following set of REXX clauses that span records, The Tivoli NetView for z/OS program removes the blanks in the middle of the quoted string from the output.

   Enter the following command:

   ```
   say 'ABC
   DEF'
   ```

   The output is the following text:

   ```
   ABCDEF
   ```

   All the trailing blanks were removed between the characters C and D.

   Blanks that are to be retained are coded on the next line as in the following example:

   ```
   say 'ABC
                                              DEF'
   ```

   The output is the following text:

   ```
   ABC                                       DEF
   ```

2. When it is necessary to continue a quoted string on the next line in a NetView command list, code the following statement:

   ```
   SAY 'THIS IS AN EXAMPLE OF A LONG',
   'QUOTED STRING'
   ```

   The output is the following text:

   ```
   THIS IS AN EXAMPLE OF A LONG QUOTED STRING
   ```

   Notice that the continuation comma displays a blank at your terminal after displaying the first quoted string. If you do not want the space, you can use concatenation bars to eliminate it. This is useful when you code long system commands in your command list. Code the concatenation bars in the following way:

   ```
   SAY 'THIS IS AN EXAMPLE OF A LONG QUO'||,
   'TED STRING'
   ```

   The output is the following text:

   ```
   THIS IS AN EXAMPLE OF A LONG QUOTED STRING
   ```

Suppressing Display of Non-REXX Commands

Use the REXX TRACE command to control the echoing of REXX instructions. Use the SUPPCHAR statement in the CNMSTYLE member to suppress non-REXX (for example, NetView) commands.

Notes:

1. IGNRLSUP is ignored for commands issued from a REXX command list.
2. FOLDUP is ignored for commands issued from a REXX command list.

See the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* and the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for more information about the IGNRLSUP and FOLDUP 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 twice. If the suppression character is not known, or it might change, or a suppression character is not explicitly defined in the CNMSTYLE member, use the following general form for suppression:

```
SUPPCHAR()||SUPPCHAR()||'SET PF24 IMMED RETRIEVE'
```

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

Use the double suppression character when sufficient status is provided by the return code and to enhance performance of commands that produce line mode messages synchronously. 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).

See the SUPPCHAR() function in “Session Information Functions” on page 83 for more information about suppression characters. You can also do suppression with the HOLE stage of the PIPE command. Refer to *IBM Tivoli NetView for z/OS Programming: Pipes* for information.

Tivoli NetView for z/OS Restrictions on REXX Instructions

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

Pausing for Operator Input

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

Note: You can use this method only if it is directly issued by an operator on a 3270-type screen. Do not use this method to communicate with the operator if your command list might have its origin in any of these items:
- the NetView management console
- a pipeline,
- an autotask
- some attended MVS console interfaces
- a call issued remotely by the EXCMD or RMTCMD command

Using the PARSE EXTERNAL 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

must enter. Code the PARSE EXTERNAL or PARSE PULL instruction after the SAY instruction to temporarily stop the command list (unless, in the case of PARSE PULL, data exists on the REXX data stack). After the command list has temporarily stopped, the operator enters the NetView GO command before it continues. Any data to be passed to the command list is to 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, code the following SAY and PARSE EXTERNAL instructions:

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

The operator responds 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 EXTERNAL instruction by placing the value in the variable ANSWER.

For restrictions on using PARSE EXTERNAL, PARSE PULL, PULL, and TRACE in Data REXX files, see "Using Data REXX" on page 18.

Using the SAY Instruction

The REXX SAY instruction enables a character string of any length; however, Tivoli NetView for z/OS can display only 32 000 characters at a time.

When you issue a REXX SAY instruction in a REXX command list for the Tivoli NetView for z/OS program, a header precedes the data displayed on the operator screen. The header is defined by the screen format member in effect and, by default, is 12 characters in length.

For Data REXX files, output strings that result from the SAY and TRACE instructions are written to the network log.

Be careful when you specify MSGID() as the first item of output from a SAY instruction because the text of the SAY instruction is processed as a regular NetView message. This processing can cause the message to be trapped by a TRAP instruction and can incorrectly satisfy a WAIT instruction, or cause automation processing to loop.

Using the CALL Instruction

Use the REXX CALL instruction to pass data (arguments) in a way that can be parsed more easily by the called routine. For more information about the CALL instruction, see the TSO/E REXX library. However, when you use the CALL instruction, the called program cannot access variables in the calling program by PIPE VAR and STEM stages.

Be careful when you use the CALL instruction to call a REXX command list from another REXX command list. The command list you call is treated like a subroutine, and some data is shared between the initial command list and the called command list. For example, trapped message queues, values of NetView commands (such as GETMLINE), and the values of message processing REXX functions (such as MSGID) are shared between the two command lists. To prevent this sharing of data, call the routine as a NetView command rather than using the CALL instruction. Note that this option is not available when using Data REXX.

NetView Restrictions on REXX Functions

This section describes the restrictions that apply when coding REXX functions in REXX command lists or Data REXX files for the Tivoli NetView for z/OS program.

The REXX LINESIZE() function always returns the value 32 728 when used in REXX command lists or Data REXX files for Tivoli NetView for z/OS.

For restrictions on using REXX functions in Data REXX files, see "Using Data REXX" on page 18.

Use the REXXSTRF keyword on the DEFAULTS or OVERRIDE command to enable the REXX STORAGE() function.

The REXX STORAGE() function cannot be used in Data REXX files.

Writing REXX Function Packages

You can write your own REXX function packages for the Tivoli NetView for z/OS program. The Tivoli NetView for z/OS program supplies two dummy directories to help you write function packages for use with NetView REXX command lists. One directory is for a user function package (DSIRXUFP), and the other directory is for a local function package (DSIRXLFP).

Link edit the real directory and function code into load module DSIRXUFP for a user function package or into DSIRXLFP for a local function package. As part of coding the interface to your function code, use the NetView DSIRXEBS macro to obtain a new EVALBLOCK.

See the REXX library for instructions on coding a real directory and coding the interface to your function code.

See the *IBM Tivoli NetView for z/OS Programming: Assembler* for information about the DSIRXEBS macro and function packages, and about writing function package directories.

See the *IBM Tivoli NetView for z/OS Installation: Configuring Additional Components* and the *IBM Tivoli NetView for z/OS Tuning Guide* for information about improving the performance of REXX function packages for the Tivoli NetView for z/OS program.

Changing the Environment Addressed by REXX Command Lists

REXX command lists for the Tivoli NetView for z/OS program use Tivoli NetView for z/OS 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 run MVS subcommands, first change the addressing environment with an ADDRESS MVS instruction.

In ADDRESS MVS, you can use the following commands:
- DELSTACK
- DROPBUF
- EXECIO
- MAKEBUF
- NEWSTACK
- QBUF

- QELEM
- QSTACK
- SUBCOM
- TE
- TS

See the TSO/E REXX library for more information about these commands and the REXX ADDRESS instruction.

In the NETVIEW addressing environment, the entire command string is converted to uppercase characters. If you want to issue a command using lowercase characters, change the addressing environment to NETVASIS, in the following way:

```
address netvasis 'WTO This is a mixed case message.'
```

Notes:

1. The only valid addressing environments recognized in a NetView REXX command list are NETVIEW, NETVASIS, and those supported by TSO/E REXX in any MVS address space.
2. Programs such as SDSF that do program calls abend if they are linked or attached. For example, the NetView status monitor and the subsystem router are examples of programs that can perform program calls. The Tivoli NetView for z/OS program cannot link or attach to any such program that can perform program calls.
3. The Tivoli NetView for z/OS program does not support a TSO/E environment in the NetView address space.
4. The Tivoli NetView for z/OS program returns an error if you try to run a command that is routed to an incorrect addressing environment.
5. The ADDRESS command supports only address environment NETVDATA when issued from Data REXX files.

Data REXX Host Command Environment

NETVDATA is the host command environment for Data REXX. No other environments are supported. This host command environment does not support commands.

Using the EXECIO Command

If you use the 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. You can code SIGNAL ON HALT to ensure that the FINIS option gets an opportunity to run.

If the EXECIO command encounters an error, it sets the RC variable to a nonzero return code. See the REXX library for information about return codes used by the EXECIO command.

If you use EXECIO to read or write a member of a partitioned data set (PDS) and are not sure whether the member exists, use the NetView REXX FNDMBR(...) function to determine the members existence before issuing the EXECIO command.

See “PRINT Example” on page 197 and “TYPE Example” on page 199 for examples of how EXECIO can be used in a REXX command list.

Note: The EXECIO command cannot be used in Data REXX files.

Using MVS and VTAM Commands

MVS and VTAM commands are examples of asynchronous commands. To obtain the output of these commands for processing by your procedure, use either NetView pipelines or the techniques described for TRAP and WAIT.

Notes:

1. When REXX clauses are treated as external commands by REXX, they are treated as external data by Data REXX.
2. Pipelines are not supported with Data REXX files.

Using the NetView ALLOCATE and FREE Commands

The NetView ALLOCATE and FREE commands dynamically allocate and deallocate data sets from the Tivoli NetView for z/OS program.

These commands closely resemble the TSO/E commands for allocating and deallocating data sets. However, because these commands are provided by the Tivoli NetView for z/OS program, you do not need to use the ADDRESS MVS instruction when using these commands in a command list. Enclose the commands in quotation marks as you do for other NetView commands. The TYPE, TYPEIT, and PRINT examples in Appendix G, “Examples of REXX Command Lists for NetView,” on page 169 use the NetView ALLOCATE command.

See the NetView online help for the syntax of the ALLOCATE and FREE commands.

Note: When REXX clauses are treated as external commands by REXX, they are treated as external data by Data REXX.

Using REXX Command Lists

Each time a REXX command list is run in the Tivoli NetView for z/OS program, REXX sets up a REXX environment for the Tivoli NetView for z/OS program. When the command list ends, this unique environment can be held for reuse by the same task. If two command lists are running at the same time on one operator task (for example, one command list is suspended while the other is running), two environments are required. Any REXX command lists called from another REXX command list use the REXX environment of the caller.

Before running REXX command lists, consider how many concurrent REXX command lists are usually active for any given NetView task. NetView retains up to 10 REXX environments and their associated storage until you log off, unless you use the DEFAULTS or OVERRIDE command to change the number of REXX environments retained. See the NetView online help for additional information about the DEFAULTS and OVERRIDE commands.

The Tivoli NetView for z/OS program retains REXX environments to improve REXX environment initialization performance. If more than one REXX environment is available when a REXX command list is run, the REXX command list can run

using a different REXX environment. Whether this occurs depends on the order in which other REXX command lists were started and ended during concurrent processing of the REXX command lists. Storage associated with each REXX environment can increase depending on the needs of the REXX command lists. Since each REXX command list can have different storage needs, REXX environments can grow to meet the needs of the most demanding REXX command list.

You can reduce the number of REXX environments the Tivoli NetView for z/OS program retains, to minimize the storage each task using REXX requires. However, if you set this number to zero, the Tivoli NetView for z/OS program does not save any REXX environments and the initialization performance of every REXX command list is affected.

Consider the storage required to initialize a REXX environment before running any REXX command lists. By default, REXX gets sufficient storage for a REXX command list with about six levels of nested calls. You can change the acquired storage amount with the DEFAULTS or OVERRIDE command.

REXX command lists that use large numbers of REXX variables or that nest more than six levels cause the storage to increase as needed. Each REXX command list requires approximately 12K of storage to start. If you set the amount of initialization storage to zero, storage is acquired as needed, but performance is degraded for the first REXX command list using this REXX environment.

Notes:

1. Two entries in the REXX IRXANCHR table are required for each non-nested NetView or REXX command list to run. If a REXX command list is called from another REXX command list, a new environment is not required. The nested command list uses the environment of the primary command list.
2. A recommended default number of REXX environments slots in IRXANCHR for the Tivoli NetView for z/OS program is twice the maximum number of command lists that can be scheduled to run concurrently under all active NetView tasks, plus one for Data REXX for each active NetView task.
3. The number of REXX environments that are allowed to be initialized by any NetView task can be controlled through the use of the DEFAULTS command REXXENVL keyword and the OVERRIDE command REXXENVL keyword. The number of initialized REXX environments for a NetView task and their relative percentage to all REXX environments can be displayed using the LIST operator command or the LIST STATUS=TASKS,RXENVCNT=YES command. The latter command also displays the total number of REXX environments that are available to NetView and the total number of REXX environments that are initialized followed by the percentage of the total number in parentheses. It also displays the number of REXX environments that are initialized for each NetView task followed by the percentage of the total number in parentheses.

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, 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 MVS. 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 the REXX library for information about 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, the NetView PARSEL2R command, or the PIPE EDIT stage.

PARSEL2R is provided by the Tivoli NetView for z/OS program to make an instruction equivalent to the REXX PARSE instruction available in both the NetView command list language and REXX. The REXX PARSE instruction performs better than PARSEL2R,; use it where possible.

When you use PARSEL2R in a REXX command list, enclose the command in quotation marks to avoid variable substitution. For example:

```
TITLE = 'PROCEDURE/ACTION NOT SUPPORTED: X''087D'''
'PARSEL2R TITLE A1 A2 A3 A4 A5 A6 A7 A8'
```

See the NetView online help for information about the PIPE EDIT and PARSEL2R commands. See the REXX library for information about the REXX PARSE instruction.

Note: Data REXX supports only the REXX PARSE instruction.

Tracing REXX Command Lists

During the creation of a REXX command list for the Tivoli NetView for z/OS 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. The TS command has the following syntax:

TS

After receiving the following message:

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

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

```
GO X=5
```

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

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

If you enter a TS command but decide before the trace begins that you do not want to run the trace, use the TRACE END (TE) command to cancel the trace. You can also use the TE command to end a trace that is not interactive.

Note: The TS and TE commands are not supported in Data REXX.

The TE command has the following syntax:

TE

For more information about TS and TE, see the NetView online help.

Return Codes in REXX Command Lists

The REXX return code variable, RC, is set after completion 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.

The following RC values and meanings are possible:

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 you code SIGNAL ON FAILURE.
–3	The command or nested command list is not authorized for this operator or the REXX ADDRESS environment is not valid. The –3 return code passes control to the FAILURE label if you code SIGNAL ON FAILURE.
–5	The command list is canceled. The –5 return code passes control to the HALT label if you code SIGNAL ON HALT.
Others	Other return codes are set by individual commands, instructions, or nested command lists.

See “Recovering from Errors in REXX Command Lists” for more information about using the SIGNAL instruction with the Tivoli NetView for z/OS program.

Recovering from Errors in REXX Command Lists

When an error occurs in a REXX command list, you can use the SIGNAL instruction to enable processing to continue at a certain point. If the REXX command list calls a command processor that is external to REXX, such as TRAP or WAIT, use the SIGNAL instruction to handle error conditions from that command processor. A command list can encounter an error for the following reasons:

- An error exists in the coding of the command list.

- 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 calling 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 the following 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 the Tivoli NetView for z/OS program 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.
- 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 you do not code SIGNAL ON FAILURE 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 must 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.
- 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 run on the current operator task while your command list is running.
  - A CLOSE IMMED command is run on any task in your Tivoli NetView for z/OS program 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 terminal session is lost for any reason, including the operator entering the LOGOFF command, while the command list is running.

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, the Tivoli NetView for z/OS program 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 next sequential statement of the command list tests the RESULT variable for the –5 cancel condition.
3. If you code SIGNAL ON FAILURE, the Tivoli NetView for z/OS program passes only the halt condition to the calling command list if you code EXIT –1.

For more information about the SIGNAL instruction, see the REXX library.

Chapter 3. REXX Instructions for Command Lists Run in a NetView Environment

Some instructions used in REXX command lists for the NetView program are provided as part of the NetView program so that REXX command lists can perform specific NetView activities. Because these instructions are provided by the program and are not standard REXX instructions, they can be used only in command lists that run in a NetView environment. These instructions do not function in REXX EXECs that are running in non-NetView environments. The REXX instructions provided by the NetView program can be used only in command lists, and are not available for entry at operator consoles. To handle error conditions, code the SIGNAL instruction in any REXX command list that uses one of these NetView instructions.

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.

See Appendix E, "Comparison of REXX and NetView Command List Language," on page 161 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. Note that the REXX instructions MSGREAD and FLUSHQ are provided by the NetView program; however, these instructions are not supported by Data REXX. No commands are support by Data REXX.

Pipelines, called with the PIPE command, provide both extended function and reduced complexity for the automation of message handling. The PIPE command is an alternative to the TRAP and WAIT instructions.

For information about NetView pipelines, see the *IBM Tivoli NetView for z/OS Programming: Pipes*.

For more information about REXX syntax rules and information about other REXX instructions, see the REXX library.

TRAP, WAIT, and MSGREAD monitor the operator station task (OST) for specific messages or wait for a specified period.

Use the TRAP command to define the messages for which the command list must 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 command is issued, the command list stops processing until one or more of the messages specified on the TRAP instruction is received, or when a Persist action provides a message for the TRAP instruction. When a WAIT instruction completes, the value returned by the EVENT() function indicates the reason that the WAIT instruction completed.

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.

The GLOBALV command defines, gets, puts, saves, restores, and purges tasks and common global variables in REXX command lists.

See the NetView online help for more information about these REXX instructions and their syntax.

Using TRAP in Nested REXX Command Lists

You can code a TRAP command 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.

Note: The TRAP instruction cannot be used in Data REXX files.

REXX command lists called as subroutines or functions are considered to be part of the calling command list. Therefore, TRAP commands issued from subroutines or functions operate the same as if they were called in the calling command list.

If you used the REXX CALL instruction to call the nested command list, trapped messages that have not been removed using MSGREAD remain available because the trap message queue is shared with the nested command list. However, if you called the nested command list without using the CALL instruction, the trapped messages are available only to the command list that issued the TRAP instruction.

Note: If a nested command list ends before trapped messages return and these same messages were being trapped by the calling command list, the messages are available to the calling command list and are placed in the message queue. It is possible, therefore, for the message queue to grow large enough for the NetView program to run out of storage.

To prevent that from happening, you can take one of the following actions:
- End the calling command list.
- Issue the instruction TRAP NO MESSAGES.
- Issue the instruction FLUSHQ periodically

Using WAIT in Nested Command Lists

REXX command lists can issue a WAIT command if the command list is called as a regular command, as a subroutine, or as a function.

Notes:

1. The WAIT command cannot be used in Data REXX files.
2. If the command list starts VIEW, you do not need to use the WAIT command to wait for messages or to wait for operator input. For more information about VIEW, see the *IBM Tivoli NetView for z/OS Customization Guide*.
3. The WAIT command cannot be used under the PPT task.

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 finishes processing.
- If you specify the same message number on TRAP commands in both the waiting and nested command lists, the message satisfies the WAIT in the nested command list.
- If you used the REXX CALL instruction to call the nested command list, trapped messages that have not been removed using MSGREAD remain available because the trap message queue is shared with the nested command list. However, if you called the nested command list as a command, the trapped messages are available only to the command list that issued the TRAP command.

Using MSGREAD in Nested Command Lists

You can code MSGREAD in both a nested REXX command list and the initial REXX command list. If you use the REXX CALL instruction to call a nested command list, trapped messages are available to both the initial and nested command lists. If you call a nested command list without using the CALL instruction, trapped messages are available only to the command list that issued the TRAP instruction.

Note: The MSGREAD instruction cannot be used in Data REXX files.

Functions Set by MSGREAD

The MSGREAD instruction affects all REXX functions that refer to the current message.

For example, if MSGREAD is used to read the following message from domain DOM01:

```
DSI008I SPAN1  NOT ACTIVE
```

The following functions are set:

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

For more information about these and other message processing functions, see “Message Processing Information Functions” on page 52.

Notes:

1. Before a MSGREAD instruction is issued, the values of MSGID(), MSGORIGN(), and MSGSTR() are null. The value of MSGCNT() is 0. The MSGVAR(*n*) functions retain any values they are given when the command list is run.

2. If you issue a MSGREAD instruction when the message queue is empty, the values of MSGID(), MSGORIGN(), MSGSTR(), and MSGVAR(*n*) are set to null. The value of MSGCNT() is zero.
3. If the MSGREAD instruction reads a multiline message, the functions are set according to the first line of the message. Refer to the GETM commands in the NetView online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* for information concerning working with multiline messages.
4. The MSGVAR(1) - MSGVAR(31) functions can be given values when a command list is called in the same way as the &1 - &31 NetView command list language parameter variables. If MSGVAR(1) - MSGVAR(31) are given values when the command list is called, save those values in variables before issuing a MSGREAD instruction. This lets you use the values that are modified by the MSGREAD instruction.
5. After using the MSGREAD instruction, save the values of the message functions in variables before issuing another MSGREAD instruction.

Chapter 4. REXX Instructions for NetView REXX Command Lists and Data REXX Files

The NetView program provides a number of REXX functions for use only in NetView REXX command lists and Data REXX files. These functions are provided so that command lists and Data REXX files written in REXX can perform specific NetView activities. Because these functions are provided by the NetView program and are not standard REXX functions, you can use them only in command lists and Data REXX files that run in a NetView environment.

You can improve the performance of your REXX command list by limiting the use of REXX functions provided by the NetView program. 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 returned value of the function. You can then use the local variable in place of the function. If the value returned by the function might change during processing of the command list, you must use the function each time (instead of the local variable) to access its current value.

The functions provided by the NetView program return values 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, the NetView program returns the current value of the related system information of the function.

Use these functions to obtain information about the operating environment, test conditions in a command list, and take actions based on the results.

For more information about REXX syntax rules and other REXX functions, refer to the REXX library.

See Appendix E, “Comparison of REXX and NetView Command List Language,” on page 161 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 the NetView program and functions provided by REXX itself.

The tables in this chapter show the tasks performed by each NetView REXX function and equivalent NetView command list language control variable used in NetView command lists. The tables are listed by NetView functions. REXX functions and equivalent NetView command list language control variables are in alphabetic order, with the REXX function shown first.

In the tables, the function and control variable are followed by the description.

Notes:

1. Where both a NetView command list language control variable and a REXX function exist for a task, descriptions are given *generically* without the NetView command list language ampersand prefix or the REXX open/close parentheses suffix.

2. Where NetView command list language control variables and REXX versions of a function differ operationally, descriptions for each are given separately; the NetView command list language control variable description contains only the differences between the two versions.
3. REXX functions provided for use by the NetView program can be used only with the NetView program. These functions are not supported by the REXX interpreter and cannot be used in REXX execs run in a non-NetView environment.
4. Not all NetView REXX functions can be used in Data REXX files. See the function description to determine if a function can be used in a Data REXX file.
5. REXX functions listed in Table 7 on page 53 and Table 10 on page 68 return a value consistent with no message to process when used in Data REXX files.

Translation Functions

Table 1. Translation Functions

Function or Variable	Description
CODE2TXT(*table,code***)**	Provides translation for various types of code points to national language text. You can use the NetView program with a problem management database to open problem records when NetView alerts are received. The code point translation function is provided in REXX to translate the numeric code points received in the alert into readable text. The CODE2TXT function has the following syntax: **CODE2TXT** ►►—CODE2TXT(*table*,*code*)—————————————————————————————————►◄

Table 1. Translation Functions (continued)

Function or Variable	Description
CODE2TXT(*table,code***)** *(continued)*	where: *code* Indicates the code point to translate. This field is specified as a 1-4 character value representing the hexadecimal code point. The characters can be uppercase or lowercase. Leading zeros are ignored but are counted as characters in the four character limit. Code points in the SNADATA tables are only two characters. To make them the same length as code points in other tables, CODE2TXT adjusts your code by concatenating "00" on the end (for example, "DD" becomes "DD00" and "01" becomes "0100"). Refer to the BNJ82TBL sample and the *IBM Tivoli NetView for z/OS Customization Guide* for more information. *table* Specifies the name of the table to use in the translation. The following tables are valid: **SNAALERT** Systems Network Architecture (SNA) alert description code point **SNACAUSE** SNA probable cause code point **SNADDATA** SNA detailed data code point from subfield X'82' **SNADDAT5** SNA detailed data code point from subfield X'85' **SNADDAT6** SNA actual action code point for resolution major vector **SNAFCAUS** SNA failure cause code point **SNAICAUS** SNA install cause code point **SNAREACT** SNA recommended actions code point **SNAUCAUS** SNA user cause code point An example of using CODE2TXT follows: `CODE2TXT(SNAALERT,362B)` The example translates code point `362B` in the SNAALERT table to "TRANSMITTER FAILURE". **Error Processing:** Error conditions encountered by this function are handled in the following way: • Non-valid operand: If a non-valid operand (such as a non-valid table name) is detected, the NetView program issues message CNM432I (non-valid operand). A REXX syntax condition flag is raised and the REXX interpreter then generates a message. • Non-valid code syntax: If a non-valid syntax is detected, the NetView program issues message CNM423I (non-valid syntax). A REXX syntax condition flag is raised and the REXX interpreter generates a message. • Too many operands: Extraneous operands are ignored. • Code point not found in table: A null string is returned, but no flag is raised.

Table 1. Translation Functions (continued)

Function or Variable	Description
SUBSYM(*symbolic***)**	Returns a literal or variable character string (any character string that has multiple MVS system symbolics or a single MVS system symbolic embedded in it) with the MVS system symbolics replaced within that string. Substitution is always performed on the &DOMAIN symbolic, unless either substitution was disabled when NetView was started or else because you have not defined an MVS system symbolic on your MVS system. The SUBSYM function has the following syntax: **SUBSYM** ►►—SUBSYM(*symbolic*)———————————————————————————————►◄ where: *symbolic* Specifies the name of the MVS system symbolic. An example using SUBSYM to find out the name of the &DOMAIN follows: `SUBSYM('&DOMAIN')`

IP Address Processing

Table 2. IP address Processing

Function or Variable	Description
IPXLATE(*'xltype', 'xlindata'***)**	Used to validate a presentation form IP address and convert it to another format, either a standard format or a compressed format.

The IPXLATE function has the following syntax:

IPXLATE

Where:

xltype
Is a string containing the type of IP address verification or translation to be performed. Following are the valid strings:

COMPRESS
Convert an IP address in presentation form to presentation form compressed, in which one group of multiple, consecutive zero address segments is replaced by a double-colon (::). The compression applies only to IPv4-mapped IPv6, IPv4-compatible IPv6, and IPv6 addresses. The IPv4 addresses are returned in standard presentation form.

STANDARD
Convert an IP address in presentation form to standard presentation form, in which IPv4, IPv4-mapped IPv6, and IPv4-compatible IPv6 addresses are presented in dotted decimal IPv4 address format. The IPv6 addresses are presented in hexadecimal format. In a standard presentation form, all segments of an address are present and leading zeros are removed.

VERIFY
Verify that the input data is an IP address in presentation form.

V42STD
Convert an IP address in presentation form to an IPv6 presentation form. The IPv4 and IPv4-mapped IPv6 addresses are presented as IPv4-mapped IPv6 addresses. All IP addresses are returned in standard presentation form.

xlindata
Is a string containing an IP address in presentation form.

Return Codes: The output from the IPXLATE function is a string containing the return code, and, if the return code is 0 and the type parameter is not VERIFY, the translated IP address string. When the type parameter is VERIFY, no IP address string is returned.

The output string can be parsed as follows:

```
parse var xxxxxx trancode ipaddr
```

Where *xxxxxx* is the variable name which contains the result of the translation, *trancode* is the return code from the translation service, and *ipaddr*, if present, is the translated address string. The values of *trancode* are as follows:

0 The function was successful.

4 - 16 The IP address translation failed because of an internal error in the translation routines. Check the NetView log for the DWO050E message and contact IBM Software Support.

20 or above
The IP address passed to the service was not valid.

Command List Information

Table 3. Command List Information

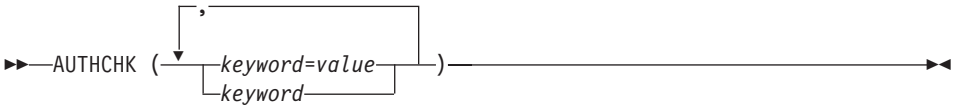
Function or Variable	Description
AUTBYPAS	For information about this function, refer to the *IBM Tivoli NetView for z/OS Security Reference*.
AUTHCHK(*keyword*=*value***)**	This REXX-only function makes a command security check for keywords and values from a REXX program. Use this to check the parameters that are passed to the command list, or any other items you want to check as keywords and values related to this command list. The AUTHCHK function has the following syntax:  where: *keyword* Specifies the keyword to be authority checked. Each *keyword* can contain a maximum of 8 characters. A maximum of 20 keywords with optional values can be passed to the program. Because variable substitution can yield a null keyword, AUTHCHK can accept a null keyword. For example, `AUTHCHK()` is a valid call of the AUTHCHK function. When a null keyword is passed to the AUTHCHK function, authority is assumed to be granted for that particular keyword. *value* Specifies a value for the keyword. Each *value* can contain a maximum of 8 characters. You cannot specify *value* without also specifying the *keyword*= parameter. Because variable substitution can yield a null value, AUTHCHK can accept a null value and strip the "=" (equal sign) to yield an *only* security check. For example, after variable substitution, this is a valid call of the AUTHCHK function: `AUTHCHK(1=,2=value2,keyword3,keyword4=value4)` Keywords and *keyword=value* combinations must be separated by commas. **Usage notes:** 1. If the keyword and value are both null, the null string is returned, which implies that authority is granted. 2. *keyword=value* can be any of the following items: • The value of a single variable. • Two variables with '=' in between. The = sign must be enclosed in single quotation marks. • Two literal strings with '=' in between. The = sign must be enclosed in single quotation marks. • A literal and a variable with '=' in between. The = sign must be enclosed in single quotation marks.

Table 3. Command List Information (continued)

Function or Variable	Description

AUTHCHK() *(continued)*	If all keyword or *keyword=value* combinations in the list pass authority checking, AUTHCHK returns a null string. Otherwise, the first keyword to fail authority checking is returned and any remaining keywords are not checked. If a *value* fails authority checking, the first *keyword=value* combination to fail is returned and any remaining keywords are not checked. If a syntax error occurs, the keyword or the *keyword=value* combination containing the syntax error is returned and the remaining keywords are not checked.
	
	For example, if a REXX program was run by entering `NVRXCMD START,LU=LU200`, authority checking of the keywords START and LU=LU200 can be done by coding the following statements in the NVRXCMD program:
	
	`/* NVRXCMD:`
	`SAMPLE REXX PROGRAM`
	`*/`
	`PARSE ARG P1','P2','.`
	`RESULT=AUTHCHK(P1,P2)`
	`...`
	`IF RESULT¬='' THEN`
	`DO`
	`SAY OPID() 'IS NOT AUTHORIZED TO KEYWORD/VALUE' RESULT`
	`EXIT`
	`END`
	`...`
	
	In this example, if either of the parameters passed in the variables *P1* and *P2* does not pass authority checking, a non-null value is returned by AUTHCHK. If a keyword fails, it is included in a message and the REXX program ends. If a value fails, the keyword and value are included in a message and the REXX program ends. For example, if OPER1 enters `NVRXCMD START,LU=LU200`, but is not authorized to use the START keyword, `OPER1 IS NOT AUTHORIZED TO KEYWORD/VALUE START` is displayed and NVRXCMD ends. If OPER1 enters `NVRXCMD START,LU=LU202`, but is not authorized to use the value LU202, `OPER1 IS NOT AUTHORIZED TO KEYWORD/VALUE LU=LU202` is displayed and NVRXCMD ends.
	
	For information about keyword security, refer to the RACF library and the *IBM Tivoli NetView for z/OS Administration Reference*.

Table 3. Command List Information (continued)

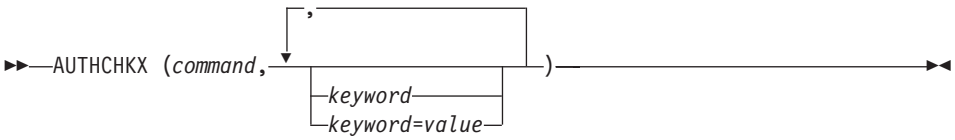
Function or Variable	Description
AUTHCHKX(*command*, *keyword*=*value***)**	This REXX-only function can be used to make a command security check for a command and its associated keywords and values from a REXX program. Use this function to check a command and any keywords, keywords and values, or any other items you want to check as keywords and values related to the command. The AUTHCHKX function has the following syntax:  where: *command* Specifies the command to be used for the authorization checks. This is a required parameter. If a null or blank command is passed, a syntax error occurs. The command parameter can contain a maximum of eight characters. It cannot contain an embedded blank, comma, or equal sign. Command security checking is first performed on the command, which can be a command synonym. If the value is not a valid command, or if the command is not authorized to be issued by the current operator, the command parameter is returned. *keyword* Specifies the keyword, or the keyword portion of the *keyword=value* pair, to be authority checked. The keyword cannot contain an embedded blank or comma. Each keyword can contain a maximum of 246 characters. A maximum of 19 keywords with optional values can be passed to the program, because REXX supports up to 20 parameters on a function call. If the keyword is not a valid keyword, or if the current operator is not allowed to execute the specified command with the specified keyword, the keyword is returned. *value* Specifies a value for the keyword. Each value can contain a maximum of 246 characters. You cannot specify *value* without *keyword* and the = (equal sign). Because variable substitution can yield a null value, AUTHCHKX can accept a null value and strip the = (equal sign) to yield a *keyword*-only security check. For example, after variable substitution, this example is a valid call of the AUTHCHKX function: `AUTHCHKX(command,keyword1=,keyword2=value2` If the current operator is not allowed to execute the specified command with the specified *keyword=value* pair, the *keyword=value* pair is returned.

Table 3. Command List Information (continued)

Function or Variable	Description

AUTHCHKX()*(continued)*	**Usage notes:**
	1. All parameters on the AUTHCHKX invocation must be separated by commas, which is the proper format for REXX function calls.
	2. All security checks are performed using parameters that have been stripped of leading and trailing blanks.
	3. If a keyword or *keyword=value* pair parameter is null or all blanks, or if it is considered valid and passes security checking, processing continues to the next parameter.
	4. If the current command is running with security checking bypassed (such as with **AUTBYPAS(ON)**), all security checking passes.
	5. f the current operator is authorized to issue the specified command with all of the specified keyword and keyword=value pairs, and all parameters are considered to be valid, the null string is returned, indicating that authority is allowed.
	6. Parameters that can be specified on certain commands (such as **READSEC** and **WRITESEC**) do not follow the standard *keyword=value* syntax when the command processor performs authorization checking. To have A**UTHCHKX** perform equivalent authorization checking for these parameters, specify them in the required *keyword=value* syntax instead of the syntax that the command requires. Refer to all available documentation on existing commands before coding the desired AUTHCHKX function invocation.
	7. For information about keyword security, refer to the RACF library and the *IBM Tivoli NetView for z/OS Security Reference*.
	
	Example:
	
	If a NetView command list language program NVCLCMD needs to perform authority checks on two parameters that are passed to it, it can call REXX program NVCHKAUT as follows:
	`NVCHKAUT NVCLCMD,&P1,&P2`
	
	NVCHKAUT can be coded in the following way (after the initial REXX comment line):
	`PARSE ARG cmd','parm1','parm2`
	`res=AUTHCHKX(cmd,parm1,parm2)`
	`Return_Code = 0`
	`IF res <> '' THEN`
	`  DO`
	`    SAY OPID() 'IS NOT AUTHORIZED TO ISSUE' ,`
	`        cmd 'WITH' res`
	`    Return_Code = 8`
	`  END`
	`Return Return_Code`
	
	In this example, if either of the parameters passed in *parm1* or *parm2* does not pass authority checking with the command passed in *cmd*, a non-null value is returned by **AUTHCHKX**. Because *keyword=value* is currently running, that command must be allowed under the current operator. If a keyword fails, it is included in a message along with the command that was passed, and the REXX program sets a return code of 8 and returns. If a value fails, the keyword and value are included in a message along with the command that was passed, and the REXX program sets a return code of 8 and returns. For example, if OPER1 enters NVCLCMD START, LU=LU200, but is not authorized to use the START keyword, the message OPER1 IS NOT AUTHORIZED TO ISSUE NVCLCMD WITH START is displayed, and NVCHKAUT returns a return code of 8 to NVCLCMD. NVCLCMD should then end without further processing.

Table 3. Command List Information (continued)

Function or Variable	Description
CMDNAME()	This REXX-only function returns the name by which the program was called. This name can be the same as the name returned in the third token by the REXX PARSE SOURCE command. This name is the command as it was entered, which is possibly a synonym. For Data REXX, this function returns the member name of the file that is being processed.
IPV6ENV()	Provides access to the configuration value for IPv4 or IPv6 specifications. The following values returned by this function are as follows: **NONE** When this value is returned, NetView services, functions, and components should not attempt to perform IPv6 operations and should use only IPv4 operations, failing even if only IPv6 addresses are available. **MIXED** When this value is returned, NetView services, functions, and components should attempt to perform IPv6 operations first, and if they fail, use IPv4 operations. In many circumstances, it will be possible to use operations that support both IPv4 and IPv6 operations at the same time. For example, it is possible to listen on an AFINET6 socket for either IPv4 datagrams or IPv6 datagrams, either of which can represent SNMP traps. In mixed environments, the explicit specification of the address family for the IP command in process should be respected. The explicit specification of an address family that conflicts with the DEFAULT.IPV6ENV variable specification should be treated as an error. A description of the DEFAULT.IPV6ENV variable specification can be found in the *IBM Tivoli NetView for z/OS Administration Reference*. **ONLY** When this value is returned, NetView services, functions, and components should only attempt to perform IPv6 operations and fail if they do not succeed or if only IPv4 addresses are available.
PARMCNT() **&PARMCNT**	Returns the number of parameter variables that are entered when a command list is initiated. For example, if command list RESC is initiated by entering `RESC ACT,LU200`, then PARMCNT becomes 2. If no parameter variables exist, PARMCNT is zero.
&PARMSTR	Returns the string of parameter values used when the command list is 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 no parameter variables exist, &PARMSTR is null. The maximum length of the string returned by &PARMSTR is 255 characters.

Table 3. Command List Information (continued)

Function or Variable	Description
&RETCODE	Returns the return code set by either the most recent command procedure or the most recently activated or nested command list. &RETCODE is initialized to zero. &RETCODE is set by a command procedure 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 131. On the &EXIT statement, you can set the return code to 0, –1, or a positive integer. The NetView program can set the return code to 0, –1, –2, –3 or –5. You cannot code –2 or –3 on the &EXIT statement, but you can test for them. All other negative return codes are reserved. The following values and meanings of &RETCODE are possible: *n* A positive integer; you define the meaning. If &CONTROL ERR is in effect, the command is echoed on the panel. **0** No error. **–1** An error is 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 authorized for this operator. 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 stopped as the result of a RESET or other failure.

Cross-Domain Information Functions

Table 4. Cross-Domain Information Functions

Function or Variable	Description
NVCNT() **&NCCFCNT**	Returns the number of NetView domains with which the operator can establish a cross-domain session.

Table 4. Cross-Domain Information Functions (continued)

Function or Variable	Description
NVID(*n***)** **&NCCFID** *n*	Returns the NetView domain identifier of a domain with which you can establish a cross-domain session. The domains with which you can establish cross-domain sessions are defined by the DOMAINS statement of your operator profile. However, if your profile specifies AUTH CTL=GLOBAL, you can establish cross-domain sessions with the domains specified by the RRD statement in the CNMSTYLE member. If neither DOMAINS nor CTL=GLOBAL is specified in your operator profile, you receive an error message when using this function. For more information about the DOMAINS and RRD statements, refer to the *IBM Tivoli NetView for z/OS Administration Reference*. The value of *n* is either a number or a variable with a numeric value. The maximum value of *n* is the value of NVCNT. **Notes:** 1. If you specify a value that is not valid in *n* for: • NVID, a null value is returned • &NCCFID, an error message is returned 2. To obtain the local domain identifier, use the APPLID function. APPLID returns the local domain ID appended with a three-character alphanumeric value assigned by the NetView program.
NVMASTER()	Returns the domain name of the master NetView program in the sysplex or a null value if there is no master NetView program.
NVSTAT(*name***)** **&NCCFSTAT** *name*	Indicates whether you have an active session with a domain. The value of *name* is the domain identifier of the domain you are querying. If you have an active session with the domain, NVSTAT(*name*) or &NCCFSTAT return a value of ACT. If you do not have an active session with the domain, INACT is returned. **Note:** If you specify no *name* or a *name* that is not valid for: • NVSTAT, a null is returned. • &NCCFSTAT, an error message is returned.

Data Set Information Functions

Table 5. Data Set Information Functions

Function or Variable	Description
FNDMBR (*DD_name***,***member_name***)**	Tries to find the specified member in the files identified by the DD name. The files must already be allocated by NetView when FNDMBR is processed. The allocated files must be a partitioned data set (PDS). FNDMBR returns two determinant results and various indeterminate results that you can use to debug your REXX program. The arguments of the FNDMBR function are defined in the following way: *DD_name* Specifies the DD name by which the allocated PDS file is known to the NetView program. *member_name* Specifies the name of the member of the allocated PDS file to be located.

Table 5. Data Set Information Functions (continued)

Function or Variable	Description
FNDMBR (*DD_name*,*member_name***)** *(continued)*	The following results can be returned by the FNDMBR function: **0** Indicates that the specified member name was found. **4** Indicates that the specified member name was not found. **100** Indicates that a system error was encountered while trying to process this request. **F** *cccccccc rrrrrrrr* where: **F** Indicates that the MVS FIND macro failed. *cccccccc* Is the FIND macro return code. *rrrrrrrr* Is the FIND macro reason code. **O** *cccccccc rrrrrrrr* where: **O** Indicates that the MVS OPEN macro failed. *cccccccc* Is the OPEN macro system ABEND code or zero. *rrrrrrrr* Is the OPEN macro return code. If *cccccccc* is not zero, this is the same as the return code value in the IEC1nnI message associated with the OPEN macro system abend code. If *cccccccc* is zero, this is the return code from the OPEN macro. For example, a return code of 8 in this case might mean that the DD statement is missing or the file is not allocated. The following example shows the FNDMBR function usage: `IF FNDMBR('DSICLD','MYREXX') = 0 THEN` This REXX statement is evaluated as true if MYREXX exists in DSICLD or as false if MYREXX does not exist in DSICLD. **Notes:** 1. Refer to the MVS/DFP library for the OPEN macro return codes. Refer to the MVS/ESA library for the OPEN macro system abend codes (X'n13' abend codes) and IEC1nnI message explanations. 2. If a REXX variable is used to hold the DD name or the member name for the FNDMBR function, to help insure that the text substituted for the variable does not exceed 8 characters, strip the leading and trailing blanks from the value before calling the FNDMBR function. 3. If the NetView ALLOCATE command was used to allocate the data set, be sure not to use the FREE option on the ALLOCATE when using the FNDMBR function together with another command (for example, EXECIO) to access the allocated file. The FNDMBR function runs an MVS OPEN and CLOSE, which causes the allocated file to be deallocated if the FREE option was coded on the allocate command. This restriction does not apply to files allocated using the DD statements in the NetView startup procedure. 4. The data set can be allocated down to the member level. However, do not do that because an IEC141I message is issued if the member is not found.

Global Variable Information Functions

Table 6. Global Variable Information Functions

Function or Variable	Description
CGLOBAL(*name*)	**REXX** Returns the value of the named common global variable if it exists. If no common global variable with the specified *name* exists, a null value is returned. If you do not specify *name*, or if you specify more than one *name*, a syntax error occurs. **End of REXX** **NetView Command List Language** The NetView command list language control statement &CGLOBAL is operationally different than the NetView REXX function described here. See "Using &TGLOBAL and &CGLOBAL" on page 148. **End of NetView Command List Language**
TGLOBAL(*name*)	**REXX** Returns the value of the named task global variable if it exists. If no task global variable with the specified *name* exists, a null value is returned. If you do not specify *name* or specify more than one *name*, a syntax error occurs. **End of REXX** **NetView Command List Language** The NetView command list language control statement &TGLOBAL is operationally different than the NetView REXX function described here. See "Using &TGLOBAL and &CGLOBAL" on page 148. **End of NetView Command List Language**

Message Processing Information Functions

Table 7 on page 53 lists functions and variables that, unless otherwise noted, are available for use on messages generated by all operating system platforms supported by the NetView program.

The value of the NetView command list language control variable and REXX function is null unless otherwise stated if no message processing information is available.

Some of the functions and variables that are listed in this section contain discussion of a *current message*. "Working with Messages" on page 14 contains additional information about *current message*.

Message Processing Information

Table 7. Message Processing Information

Function or Variable	Description	
ACTIONDL() **&ACTIONDL**	Returns the reason for which the NetView program deleted the associated message, which is one of the following values:	
	(null)	The message is not being deleted.
	LOCAL	The message was deleted by operator overstrike of the CONSOLE DELETE stage.
	NETVIEW	The message was deleted using the NetView DOM NVDELID or CURMSG options, or by the NetView program.
	SMSGID	The message was deleted by MVS DOM using SMSGID.
	TOKEN	The message was deleted by MVS DOM using TOKEN.
	TCB	The message was deleted by MVS DOM for task end.
	ASID	The message was deleted by MVS DOM for address end space.
	INVALID	The message has a combination of control flag settings that are not valid.
ACTIONMG() **&ACTIONMG**	Returns a 1 if the message is an action message. Otherwise it returns a 0.	
AREAID() **&AREAID**	Returns a 1-letter (A - Z) identifier for the area on the multiple console support console panel that displays the message.	
ATTNID() **&ATTNID**	Returns the VSE attention identifier ID. A plus sign (+) indicates that a reply is required for this message *immediately*. A minus sign (–) indicates that a reply is required for this message. This function has a value if the message is from a VSE system, but null for non-VSE messages.	
AUTOTOKE() **&AUTOTOKE**	Returns the 1 - 8 character name of the MVS message processing facility (MPF) automation token. **Note:** If you have specified `AUTO(YES)` or `AUTO(NO)` in the MPF table, the values `YES` and `NO` are not automation tokens.	
DESC() **&DESC**	Returns the MVS DESCriptor codes as a series of 16 on (1) and off (0) EBCDIC characters representing the bits in order. Refer to the MVS library for information about code values.	

Table 7. Message Processing Information (continued)

Function or Variable	Description
EVENT()	The NetView event that satisfied the WAIT instruction is determined by the value of the REXX EVENT() function. The REXX command list can use the EVENT() function to set a variable and take appropriate action based on the set value. The following returned values from the EVENT() function are possible: **M** The message for which the command list is waiting 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** You entered the GO command, and processing is resumed. **E** You did not code the WAIT or TRAP instructions correctly. For example, you entered the operands in the incorrect order or issued a WAIT for messages instruction without a matching TRAP instruction. The command list resumes processing. **X** A PERSIST TRAP for additional data is active and the PERSIST has completed. If you do not issue a WAIT instruction in a command list, the value of the EVENT() function is replaced with a value of null.
HDRMTYPE() **&HDRMTYPE**	Specifies the 1-character NetView buffer type of the received message or MSU. Buffer types are described in *IBM Tivoli NetView for z/OS Programming: Assembler*.
IFRAUGMT() **&IFRAUGMT**	Returns the UTC mean time when the automation internal function request (AIFR) was created. The variable IFRAUGMT is returned as an 8-byte hexadecimal value in store clock format.
IFRAUIND() **&IFRAUIND**	Returns 2 bytes of indicator bits as a series of 16 on (1) and off (0) EBCDIC characters representing the bits in order. This data is mapped in DSIIFR. The following bit positions are valid: **1** MVS system information attached (WQE data). **5** Message from NetView PPT. **6** Message received cross-domain. **11** Message was PRI routed by ASSIGN command. **12** Message was SEC routed by ASSIGN command. **13** Message was COPY routed by ASSIGN command. **14** Message was routed to authorized receiver. **15** The message was revised by the NetView message revision table. **16** Message was unsolicited. **Notes:** 1. Other bits can be tested, but have no defined use. All the bits are defined in the DSIIFR mapping control blocks. For more information, refer to *IBM Tivoli NetView for z/OS Programming: Assembler*. 2. Messages with the unsolicited flag on are eligible for ASSIGN PRI and SEC routing. 3. This field indicates the AIFR indicator fields IFRAUIND and IFRAUIN2. 4. MVS system messages routed to any task except the subsystem router CNMCSSIR are considered solicited messages. 5. For more information about solicited and unsolicited messages, refer to the *IBM Tivoli NetView for z/OS Automation Guide*.

Table 7. Message Processing Information (continued)

Function or Variable	Description
IFRAUIN3() **&IFRAUIN3**	Returns 1 byte of indicator bits as a series of eight on (1) and off (0) EBCDIC characters representing the bits in order. This data is mapped in DSIIFR. The following bit positions and meanings are valid: **1-2** 00 = Default priority 01 = Low priority 10 = High priority 11 = Test the receiver for priority **3** VM PMX
IFRAUI3X() **&IFRAUI3X**	Returns a 32-byte string of '1' and 0' values corresponding to control flags in the IFRAUI3X word of the DSIIFR. The first 8 bits are the same as IFRAUIN3, allowing all 32 bits to be accessed at once.
IFRAUSB2() **&IFRAUSB2**	Returns a 2-byte user field in DSIIFR as a string of 2 characters. **Notes:** 1. This function is null if the field is all blanks or binary zeros in any combination. 2. IFRAUSB2 and IFRAUSRB refer to the same user field, but return the value in different formats.
IFRAUSC2() **&IFRAUSC2**	Returns a 16-byte user field in DSIIFR as a series of 128 on (1) and off (0) EBCDIC characters representing the bits in order. **Note:** IFRAUSC2 and IFRAUSRC refer to the same user field, but return the value in different formats.
IFRAUSDR() **&IFRAUSDR**	Returns the 1 - 8 character name of the originating NetView task.
IFRAUSRB() **&IFRAUSRB**	Returns a 2-byte user field in DSIIFR as a series of 16 on (1) and off (0) EBCDIC characters representing the bits in order. **Note:** IFRAUSRB and IFRAUSB2 refer to the same user field, but return the value in different formats.
IFRAUSRC() **&IFRAUSRC**	Returns a 16-byte user field in DSIIFR as a string of 16 characters. **Notes:** 1. This function is null if the field is all blanks or binary zeros in any combination. 2. IFRAUSRC and IFRAUSC2 refer to the same user field, but return the value in different formats.

Table 7. Message Processing Information (continued)

Function or Variable	Description
IFRAUTA1() **&IFRAUTA1**	Returns 6 bytes of indicator bits as a series of 48 on (1) and off (0) EBCDIC characters representing the bits in order. IFRAUTA1 enables checking of control information. The following bit positions are valid: **1, 2, 25** HOLD action. **5, 6, 26** SYSLOG action. **7, 8, 27** NETLOG action. **9, 10, 28** HCYLOG action. **11, 12, 29** DISPLAY action. **13, 14, 30** BEEP action. **16** Command echo **20** Message from MVS. **23** VSE format message. **24** Action message. **47** Automation vector extensions exist. **48** Presentation vectors exist in data buffers. **Notes:** 1. Other bits can be tested but have no defined use. 2. Refer to the description of DSIIFR fields IFRAUTA1 through IFRAUTA6 in *IBM Tivoli NetView for z/OS Programming: Assembler*.
IFRAUWF1() **&IFRAUWF1**	Returns 4-byte MVS-specific WTO information as a series of 32 on (1) and off (0) EBCDIC characters representing the bits in order. The following specific bit positions with defined uses are valid: **6** Message is a WTOR. **7** Message is suppressed. **8** Broadcast to all. **9** Display JOBNAMES. **10** Display STATUS. **14** DISPLAY SESSION. **Note:** Other bits can be tested but have no defined use. MLWTO flags in this area also have no defined use. MLWTO indicators are moved into the data buffers.
JOBNAME() **&JOBNAME**	Returns the 1 - 8 character MVS job name identifier. Because JOBNAME is the name of the job that originated the message, it might not always be the same as the name of the job to which the message is referring. For example, the job names might be different when MVS issues a message about the NetView job. Also, JOBNAME can contain the name of an initiator (instead of the actual job name) when a job is started or stopped. If the message is issued during startup or stopping, extract the job name from the message text rather than using the JOBNAME function. The same information is available using MSGCOJBN; refer to the information about MSGCOJBN on page 58.
JOBNUM() **&JOBNUM**	Returns the 8-character MVS job number identifier. **Note:** The MVS job identifier might contain embedded blanks.
KEY() **&KEY**	Returns the 8-character retrieval key associated with the message.

Table 7. Message Processing Information (continued)

Function or Variable	Description
LINETYPE() **&LINETYPE**	Returns the following multiline write-to-operator (MLWTO) line type or MSU data buffer type: **C** Message control line. **L** Message label line. **D** Message data line. **DE** Last message data line. **E** The line is the last message line and contains no data. **H** The line is the HIER data buffer type. **M** The line is the MSU data buffer type. **blank** The message is a single-line message. **null** No message or MSU data buffer is associated with this command list.
MCSFLAG() **&MCSFLAG**	Returns the system message flags in a series of eight on (1) and off (0) EBCDIC characters representing the bits in order. The following bit positions and meanings are valid: **1** Send message conditionally to console SYSCONID. **2** Send message unconditionally to console SYSCONID. **3** RESP. **4** REPLY. **5** BRDCST. **6** HRDCPY only. **7** NOTIME. **8** NOCPY. **Notes:** 1. This function does not return the same mapping of multiple console support flags as the automation table compare item. 2. Setting MCSFLAG='00000000' is valid. It overrides MCSFLAG set by an incoming WTO.
MSGASID() **&MSGASID**	Returns the MVS system address space identifier from which the message was issued. The value of MSGASID is a 1–5 digit decimal number. **Note:** This value is null for messages that do not come from an MVS address space.
MSGAUTH() **&MSGAUTH**	Returns the 2-character value indicating whether the message was issued from an authorized program. **00** WTO message is not from MVS. **10** WTO is from an unauthorized program. **11** WTO is from an authorized program.
MSGCATTR() **&MSGCATTR**	Returns the 16-bit MVS message attribute flags as a series of on (1) and off (0) EBCDIC characters representing the bits in order. The following bit positions and meanings are valid: **1** Message is suppressed. **2** Message is a command response. **3** Message issued by authorized program. **4** Message is to be retained by Automation Message Retention Facility (AMRF). **Note:** Other bits can be tested but have no defined use.
MSGCMISC() **&MSGCMISC**	Returns the 8-bit MVS miscellaneous routing flags as a series of on (1) and off (0) EBCDIC characters representing the bits in order. The following bit positions and meanings are valid: **1** Display UD (undeliverable) messages. **2** Display only UD messages. **3** Queue by ID only. **4** Indicates whether the message has been marked in the message processing facility (MPF) table as eligible for NetView automation. **Note:** Other bits can be tested but have no defined use.

Table 7. Message Processing Information (continued)

Function or Variable	Description
MSGCMLVL() **&MSGCMLVL**	Returns the 16-bit MVS message level flags as a series of on (1) and off (0) EBCDIC characters representing the bits in order. The following bit positions and meanings are valid: **1** WTOR **2** Immediate action **3** Critical eventual action **4** Eventual action **5** Informational **6** Broadcast **Note:** Other bits can be tested but have no defined use.
MSGCMSGT() **&MSGCMSGT**	Returns the 16-bit MVS message type flags as a series of on (1) and off (0) EBCDIC characters representing the bits in order. The following bit positions and meanings are valid: **1** Display job names **2** Display status **3** Monitor active **6** Monitor SESS **Note:** Other bits can be tested but have no defined use.
MSGCNT() **&MSGCNT**	**REXX** Returns the number of items in the message string of the current message (see "Working with Messages" on page 14 for information about *current message*). **End of REXX** **NetView Command List Language** Returns the number of words in a message string. See "Control and Parameter Variables Used with &WAIT" on page 139 for more information about using control variables with &WAIT. **End of NetView Command List Language**
MSGCOJBN() **&MSGCOJBN**	Returns the 1 - 8 character originating job name. (The same information is available using JOBNAME, described on page 56.)
MSGCPROD() **&MSGCPROD**	Returns the 16-character MVS product level. The characters are defined in the following way: • The first 4 characters represent an MVS control point object version level. • The next 4 characters represent the control program name (MVS). • The last 8 characters represent the function modification identifier (FMID) of the originating system.
MSGCSPLX() **&MSGCSPLX**	Returns the 1- 8 character name of the MVS SYSPLEX where the received message originated. Available when running under MVS/ESA Version 4 Release 3 or later.
MSGCSYID() **&MSGCSYID**	Returns the 1 - 3 digit decimal number system identification for DOM.

Table 7. Message Processing Information (continued)

Function or Variable	Description
MSGDOMFL() **&MSGDOMFL**	Returns the 8-bit MVS DOM flags as a series of on (1) and off (0) EBCDIC characters representing the bits in order. The following bit positions and meanings are valid: **1** DOM by message ID **2** DOM by system ID **3** DOM by ASID **4** DOM by job step TCB **5** DOM by token

MSGGBGPA()
 	
&MSGGBGPA
	Returns the 4-byte hexadecimal background presentation attributes. The following bytes and descriptions are valid:
	1 Background control field
	2 Background color field
	3 Background highlighting field
	4 Background intensity field.
	Use one of the following forms to check for hexadecimal values:
	REXX
	`IF MSGGBGPA() = '12345678'X THEN ...`
	End of REXX
	NetView Command List Language
	`&IF &MSGGBGPA = X'12345678' &THEN ...`
	End of NetView Command List Language

MSGGDATE() **&MSGGDATE**	Returns the message date in a 7-character format of *yyyyddd*, where *yyyy* is the year and *ddd* indicates a calendar day. **Note:** This is not necessarily the current date. It might be the date with which MVS associates the message as having been issued.

MSGGFGPA()
 	
&MSGGFGPA
	Returns the 4-byte hexadecimal foreground presentation attributes. The following bytes and meanings are valid:
	1 Foreground control field
	2 Foreground color field
	3 Foreground highlighting field
	4 Foreground intensity field
	You can use one of the following forms to check for hexadecimal values:
	REXX
	`IF MSGGFGPA() = '12345678'X THEN ...`
	End of REXX
	NetView Command List Language
	`&IF &MSGGFGPA = X'12345678' &THEN ...`
	End of NetView Command List Language

MSGGMFLG() **&MSGGMFLG**	Returns the 16-bit MVS general message flags. The following bit positions and meanings are valid: **1** DOM (delete operator message)

Table 7. Message Processing Information (continued)

Function or Variable	Description
MSGGMID() **&MSGGMID**	Returns the 4-character hexadecimal value MVS message identifier field.
MSGGSEQ() **&MSGGSEQ**	Returns the 1 - 8 character numeric decimal sequence number. This function represents the last three bytes of MSGGMID.
MSGGSYID() **&MSGGSYID**	Returns the 1 - 3 character numeric decimal system identification. This is the first byte of MSGGMID.
MSGGTIME() **&MSGGTIME**	Returns an 11-character (including periods) time in the form *hh.mm.ss.th*, where *hh* is the hours, *mm* is the minutes, *ss* is the seconds, and *th* is tenths and hundredths of seconds.
MSGID() **&MSGID**	**REXX** Returns the message identifier of the *current message* (see "Working with Messages" on page 14 for a definition of *current message*.) This is the first token, where tokens are determined by blanks. A reply ID, if present, is ignored. If a reply ID is sent with the message, it is not used as the first token. For a multiline write-to-operator (MLWTO) message, MSGID uses the first token of the first line of the message. When an MSU buffer is being processed, MSGID is equal to null (''). **Note:** For messages received over the VM PROP/PMX interface, MSGID cannot be set to the actual message identifier because of information added to the front of the message. **End of REXX** **NetView Command List Language** The difference from REXX is that &MSGID is used in NetView automation, with &WAIT, and with the LINKPD command. **End of NetView Command List Language**
MSGITEM(n) **&MSGITEM**	**REXX** Returns an item from the current message ("Working with Messages" on page 14 contains additional information about *current message*.) The first blank-delimited string after a reply ID (if one exists) is considered the message ID, and also item 0. After the message ID, the following parsing for items 1 through MSGCNT() is performed: • Strings can be delimited by either blanks or commas. • A string beginning with a single quotation mark preceded by a blank or comma, and ending with a single quotation mark followed by a blank or comma, is considered a quoted string, and is treated as a single item, which does not include the surrounding quotation marks. • Every other delimited string that is not in a quoted string is treated as a single item. • Delimiters that are not within quoted strings are not part of any item. MSGITEM can return strings longer than 255. **End of REXX**

Table 7. Message Processing Information (continued)

Function or Variable	Description
MSGORIGN() **&MSGORIGIN**	**REXX** Returns the domain where the last message read by MSGREAD originated. MSGORIGN() is used for NetView automation, with MSGREAD, and with the LINKPD command. Refer to the NetView online help for more information about using functions with MSGREAD. **End of REXX** **NetView Command List Language** Specifies the domain where the message originated. &MSGORIGIN is used in NetView automation, with &WAIT, and with the LINKPD command. See Chapter 5, "Automation Resource Management," on page 91 for more information about NetView automation. See "Control and Parameter Variables Used with &WAIT" on page 139 for more information about using control variables with &WAIT. See "LINKPD Results" on page 157 for more information about the LINKPD command. **End of NetView Command List Language** **Note:** The NetView command list language and REXX versions of this command are spelled slightly differently; be sure to use the correct spelling when writing your command list.
MSGSRCNM() **&MSGSRCNM**	Returns the 1 - 17 character source object name. This source name is an identifier from the source object that was provided by either the DSIMMDBS or CNMPMDB application programming interface (API) call. For more information about DSIMMDBS, refer to *IBM Tivoli NetView for z/OS Programming: Assembler*. For more information about CNMPMDB, refer to *IBM Tivoli NetView for z/OS Programming: PL/I and C*. The source name is selected from the source object by the following rules: • The first nickname, if any • The first network identifier concatenated to a network addressable unit (NAU) name, with a period (.) between, if both exist in sequence • The first NAU name, if it exists • The string "N/A" if none of the other names in this list are specified in the source object • Null, if no source object exists For more information about how the source object is defined, refer to the DSIAIFRO mapping in *IBM Tivoli NetView for z/OS Programming: Assembler*.

Table 7. Message Processing Information (continued)

Function or Variable	Description
MSGSTR() **&MSGSTR**	**REXX** Returns the message text of the current message (see "Working with Messages" on page 14 for information about *current message*). MSGSTR() does not include the message identifier—the token used by the MSGID() function. For a multiline write-to-operator (MLWTO) message, MSGSTR() becomes the message text of the first line of the message. **End of REXX** **NetView Command List Language** 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 139 for more information about using control variables with &WAIT. See "LINKPD Results" on page 157 for more information about the LINKPD command. **End of NetView Command List Language**
MSGTOKEN() **&MSGTOKEN**	Returns a 1 - 10 digit decimal number that indicates the token associated with the message. **Note:** You can use a TOKEN value to group WTOs by setting MSGTOKEN before issuing the WTO command. Later, these messages can be deleted using a single DOM command by specifying the token value in MSGTOKEN.
MSGTSTMP() **&MSGTSTMP**	Returns the message time stamp. The value of this field is the time when the NetView message buffer was created. The field is a 6-character string in the form of *hhmmss*, where: *hh* Hours *mm* Minutes *ss* Seconds

Table 7. Message Processing Information (continued)

Function or Variable	Description
MSGVAR(*n*)	**Note:** The use of MSGVAR is supported for compatibility purposes. Use MSGITEM because the MSGVAR(n) functions do not return data relevant to the *current message* as described in "Working with Messages" on page 14. Use the MSGITEM(n) functions to return current message information. Returns the text of a message. The NetView program changes the values of the MSGVAR(1) through MSGVAR(31) functions to reflect the text of the message. **Note:** MSGVAR(1) through MSGVAR(31) are equivalent to the NetView command list language variables &1–&31. 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 NetView automation, with MSGREAD, and with the LINKPD command. Refer to the NetView online help for more information about using functions with MSGREAD. See "LINKPD Results" on page 157 for more information about the LINKPD command. The MSGVAR(*n*) functions can be given values when a command list is called in the same way as are the &1–&31 NetView command list language parameter variables.
MSGTYP() **&MSGTYP**	Returns the system message type as a series of three on (1) and off (0) EBCDIC characters representing the bits in order. An on character (1) in one of the positions corresponds to the following values: **1** SESS — Corresponds to IFRAUWF1(14) **2** JOBNAMES — Corresponds to IFRAUWF1(9) **3** STATUS — Corresponds to IFRAUWF1(10)
MVSRTAIN() **&MVSRTAIN**	In the automation table, a 3-bit field describing MVS retain characteristics of the message. **Note:** The 3 flags correspond to 3 flags defined in the MVS WQE control block when NetView is using the SSI interface, and corresponds to 3 similar flags in the MDB when running in Extended Console Mode. The exact meaning and use of the flags is a property of the operating system. &MVSRTAIN in NetView command list language is a 3-bit field describing MVS retain characteristics of the message.
NVDELID() **&NVDELID**	Returns the 24-character NetView deletion identifier for a message. You can remove the message from the held queue for all tasks in the NetView program using the NetView DOM NVDELID command. This is the NetView equivalent of the MVS DOM function, but is used for messages that are not MVS WTOs or WTORs.
PRTY() **&PRTY**	Returns the priority of the message as set by the originator. This field is a 1–5 digit decimal number. The NetView program does not use this field when processing the message.
REPLYID() **&REPLYID**	Returns the reply identifier for WTORs. This field has a maximum length of 8 characters. For messages from VSE systems, the REPLYID is the last three characters of the 6-character message prefix. The three returned characters are the message reply ID only if the sending system uses those characters to designate a reply ID for a message.

Table 7. Message Processing Information (continued)

Function or Variable	Description
ROUTCDE() **&ROUTCDE**	Returns the MVS routing code or codes assigned to the message. The value of the field is a series of on (1) and off (0) EBCDIC characters representing the bits in order. The maximum number of ROUTCDEs assigned to a message is 128. **Notes:** 1. After the first 16 bits, the number of characters returned in ROUTCDE is the lowest multiple of 8 that contains one or more on (1) characters. Therefore, compare against a specific substring of ROUTCDE rather than against the entire string. For example, if only bit 17 is turned on, a string of 16 zeros, a 1, and 7 more zeros are returned (000000000000000010000000). One method to test for bit 17 being on is shown in the REXX example in Figure 5 on page 65. The functionally equivalent code written in NetView command list language is shown in Figure 6 on page 65. 2. Another method to check for a specific bit is to use the REXX environment POS (position) function, as shown in Figure 7 on page 65. 3. For details on using the REXX POS function, refer to the REXX library.
SESSID() **&SESSID**	Returns the 1 - 8 character ID of the TAF (terminal access facility) session that sent the message. See Chapter 5, "Automation Resource Management," on page 91 for more information about NetView automation. **Note:** If TAF session is started with a SESSID equal to the domain ID, SESSID is set unpredictably and might give unpredictable results. If the current message originated from a PPI receiver pipe stage, SESSID shows the SAF ID.
SMSGID() **&SMSGID**	Returns a 1 - 10 character decimal number that identifies a particular instance of a message. This function can be used by the DOM command to identify action messages to be removed from the display. Refer to the NetView online help for more information about DOM. This field contains the same information as MSGGMID, except that SMSGID is returned as a decimal number and MSGGMID is returned as a hexadecimal value.
SYSCONID() **&SYSCONID**	Returns the MVS system console name associated with the message. System console names are 2 - 8 characters in length. **Note:** In the command revision environment, returns the system ID under which the command was issued.
SYSID() **&SYSID**	Returns the 1-8 character identifier of the MVS system from which a message arrived. **Note:** In the command revision environment, returns the console name under which the command was issued.
WTO.REPLY **&WTOREPLY**	Returns an operator reply to a WTOR. **REXX** The REXX version is not a function. It is a local variable and therefore does not have parentheses on the end. **End of REXX**

ROUTCDE Examples

```
/* STANDARD COMPARE */
IF ROUTCDE() = '00000000000000000010000000'
   THEN SAY 'ROUTCDE BIT 17 IS SET.'
```

Figure 5. REXX Example to Test for Bit 17

```
&IF &ROUTCDE = 00000000000000000010000000 &THEN
 &WRITE ROUTCDE BIT 17 IS SET
```

Figure 6. NetView Command List Language Example to Test for Bit 17

```
/* POS COMPARE (Using the REXX environment function) */
BIT2CHK = 17
IF POS('1',ROUTCDE(),BIT2CHK) = BIT2CHK
   THEN SAY 'ROUTCDE BIT 17 IS SET'
```

Figure 7. Using the REXX POS Function to Test for Bit 17

Command Processing Information Functions

Table 8 on page 65 lists functions that are available for use with commands originating in the command revision environment.

Table 8. Command Processing Information

Function or Variable	Description
RECEDATA()	Provides information about the origin of a command that was transferred to the NetView environment by using a NETVONLY action in a Command Revision Table. For more information, see Table 9.
SYSCONID()	Returns the console name under which the command was issued.
SYSID()	Returns the console name under which the command was issued.

Table 9 lists the arguments that you can specify and the values that are returned when you use the RECEDATA function:

Table 9. RECEDATA Arguments

Argument	Synonym	Data returned
Null		When no arguments are entered, the following values are returned: **Value** **Description** **0** if the invoking procedure was not driven by a NETVONLY action **asid** a four character hexadecimal representation of the Address Space Identifier for the originating address space

Table 9. RECEDATA Arguments (continued)

Argument	Synonym	Data returned
ASTYPE	T TYPE	Type of address space (job type). A one-character value is returned: **Value** **Description** **D** USS persistent procedure. The address space has a name for initiated programs, appropriate for a JOB. However, the existence of an OpenMVS address space block indicates a special purpose USS persistent procedure. **J** The address space is a JOB. **N** The address space is a system address space started during operating system initialization (NIP) processing. **S** The address space is a Started Task (STC). **T** The address space is a Time-Sharing User (TSO). **U** The address space is a USS forked or created procedure. ***** Error: the address space where the command originated has closed. **?** Error: inconsistent data (might be a transient condition). **!** Error: inconsistent data. **>** Error: the ASID that was specified is greater than the system-generated maximum value for an ASID
AUTH	A	Authority of the console. A one-character value is returned: **Value** **Description** **C** CONSOLE **I** I/O **M** Master **S** SYS
GROUP	G	SAF group
JOBNAME	J	The 1 - 8 character MVS job name identifier
TESTMODE	X	Test mode status at the time the table was loaded: **Value** **Description** **0** Test mode was not requested **1** Test mode was requested
USER	U	SAF username

Usage Note: If the RECEDATA function is started with an argument and the starting procedure is not driven by a NETVONLY action, a REXX error is generated. Consider using the RECEDATA function initially

without an argument to check for a 0 value before performing any actions in the command revision environment.

REXX Management Services Unit Information Functions

Table 10 on page 68 lists REXX functions for management services unit (MSU) processing. MSUs include:

- Control point management services units (CP_MSU)
- Multiple domain support message units (MDS_MU)
- Network management vector transports (NMVT)
- Record maintenance statistics (RECMS)
- Record formatted maintenance statistics (RECFMS)

For more information about MSUs, refer to the *IBM Tivoli NetView for z/OS Automation Guide*.

The following terms are used in Table 10 on page 68:

Generic MSU
All MSUs that contain subvector 92. Generic MSUs include:
- Alerts that contain subvector 92
- Resolutions, which always contain subvector 92

Statistics-only RECMS
Some RECMS records contain only statistical data. The RECMS records that contain only statistical data are those with recording mode (byte 8, 1-offset, into the RECMS) X'81', X'86', and X'87' (for X'87' that represent temporary errors, not permanent errors).

Statistics-only RECFMS
Some RECFMS records contain only statistical data. The RECFMS records that contain only statistical data are those with RECFMS types (byte 8, 1-offset, into the RECFMS) 1, 4, and 5.

MSU Information

Table 10. Management Services Units (MSU) Information Functions

Function	Description
HIER (*n*)	Provides user access to the NetView hardware monitor hierarchy data associated with an MSU. The *n* specifies the index number (1–5) of a specific name/type pair. **Notes:** 1. HIER() (without the *n*) returns a resource hierarchy slightly different from that found in BNJ146I messages. The following example shows name/type pairs: `aaaaaaaa1111bbbbbbbb2222....eeeeeeee5555` The letters represent the resource name and numbers represent the resource type. The hardware monitor defines from one to five name/type pairs. Each name is eight characters long and each type is four characters. The names and types are padded with blanks if necessary. 2. HIER (*n*) returns the name/type pair *aaaaaaaa1111* that corresponds to *n*. If no name/type pair corresponds to *n*, then a null value is returned. 3. HIER(*n*) returns null under the following conditions: • If the command list is not run by the automation table • If the automation table was not driven by an MSU • If the MSU does not have a hardware monitor resource hierarchy 4. Use the HMSECREC function with HIER to determine the resource name of the hierarchy level where secondary recording is performed. For more information, see the description of HMSECREC on page 74. 5. If a complex link exists in a resource hierarchy, there might be resource levels that are not in the information returned by HIER(). You must use a system schematic to determine the complete hierarchy configuration when a complex link is present. Use the HMCPLINK function to check whether a complex link exists. See Table 10 on page 68 for more information about the HMCPLINK function. 6. For information about the NetView built-in function &HIER, see "&HIER" on page 120.
HMASPRID()	Returns a 1 - 9 character alert-sender product ID. This value is identical to the *prodid* value described for the SRFILTER (SRF) command. The ID can be one of the following IDs: • 1–4 character hardware product ID • 1–9 character software product ID Trailing blanks are removed. HMASPRID returns null if: • An MSU is not a generic record. • An MSU is not submitted to automation by the hardware monitor. The maximum length is 9 characters. HMASPRID applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMASPRID" on page 76.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMBLKACT()	Returns a 5-character value consisting of a 3-character block ID and a 2-character action code. This value is identical to the *code* value described for the SRFILTER (SRF) command. HMBLKACT returns null if an MSU is one of the following types or is not submitted to the automation table by the hardware monitor: • A generic alert (X'0000') • A resolution (X'0002') • A PD statistic (X'0025') • A link configuration data (X'1332') • A statistics-only RECMS • A statistics-only RECFMS Otherwise, a value is returned. Examples of MSUs that HMBLKACT returns a value for include nongeneric alerts (X'0000'), RECMSs that are not statistics-only, and RECFMSs that are not statistics-only. The maximum length is 5 characters. HMBLKACT applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMBLKACT" on page 77.
HMCPLINK()	Returns a 0, 1, or null to indicate whether a complex link exists, where: **1** A complex link exists. If a complex link exists, there might be resource levels that are not in the resource hierarchy returned by the HIER function. You must use a system schematic to determine the complete hierarchy configuration when a complex link is present. See the description of HIER on page 68 for more information. Hardware monitor panels, such as Most Recent Events, indicate that a complex link exists by placing an asterisk (*) in the pictorial resource hierarchy at the top of the panel and displaying message BNJ1538I in the message line near the bottom of the panel. **0** A complex link does not exist. **Null** The MSU was not submitted to automation by the hardware monitor. The maximum length is 1 character. HMCPLINK applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMCPLINK" on page 77.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMEPNAU()	HMEPNAU returns the NAU name of the entry point node where the MSU originated for alerts forwarded using the NV-UNIQ/LUC alert forwarding protocol. HMEPNAU returns the local NAU (domain) name for local MSUs. For alerts forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol, HMEPNAU returns the NAU name of the entry point node that contains the MS application that first forwarded the alert to the ALERT_NETOP application. HMEPNAU adds an asterisk (*) to the beginning of the NAU name to indicate that the name returned might not be the entry point node name. For example, if the node name is NETV01 and HMEPNAU cannot determine if the node is an intermediate node or the entry point node, it returns *NETV01. **Note:** Refer to the *IBM Tivoli NetView for z/OS Automation Guide* for more information. The maximum length is 9 characters. HMEPNAU applies only to MSUs submitted to automation by the hardware monitor. HMEPNAU returns null for all other MSUs. See the example in "HMEPNAU, HMEPNET, and HMFWDSNA" on page 77.
HMEPNET()	HMEPNET returns the NETID name of the entry point where the MSU originated. For alerts forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol, HMEPNET returns the NETID name of the entry point node that contains the MS application that first forwarded the alert to the ALERT_NETOP application. HMEPNET adds an asterisk (*) to the beginning of the NETID name to indicate that the name returned might not be the entry point node name. HMEPNET returns the local NETID name for local MSUs. If the hardware monitor cannot determine the NETID name of the entry point, HMEPNET returns an asterisk (*). HMEPNET returns an asterisk (*), indicating that the NETID name cannot be determined by the hardware monitor, for all MSUs forwarded by the NV-UNIQ/LUC alert forwarding protocol. **Note:** Refer to the *IBM Tivoli NetView for z/OS Automation Guide* for more information. The maximum length is 9 characters. HMEPNET applies only to MSUs submitted to automation by the hardware monitor. HMEPNET returns null for all other MSUs. See the example in "HMEPNAU, HMEPNET, and HMFWDSNA" on page 77.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMEPNETV()	Returns a 0, 1, or null to indicate whether the entry point where the MSU originated was a remote node NetView program. This function applies only to MSUs forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol. **1** The entry point was a NetView program. **0** The entry point was not a NetView program. **null** The MSU was not forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol. **Notes:** 1. Refer to the *IBM Tivoli NetView for z/OS Automation Guide* for more information about forwarding mechanisms. 2. The maximum length is 1 character. 3. HMEPNETV applies only to MSUs submitted to automation by the hardware monitor. HMEPNETV returns null for all other MSUs. 4. See the example in "HMEPNETV" on page 77.

HMEVTYPE()	Returns the event type of an MSU. Any trailing blanks in the event type are removed. The following event types are valid:
	`AVAL BYPS CUST DLRC HMV HELD IMPD IMR`
	`INST INTV NTFY PAFF PERF PERM PROC REDL`
	`RSLV RSNT SCUR SNA TEMP USER UNKN`
	For a complete description of all event types, see the NetView online help
	HMEVTYPE returns null if an MSUis one of the following types or is not submitted to automation by the hardware monitor:
	• A PD statistic (X'0025')
	• A link configuration data (X'1332')
	• A statistics-only RECMS
	• A statistics-only RECFMS
	The maximum length is 4 characters.
	HMEVTYPE applies to all MSUs submitted to automation by the hardware monitor.
	See the examples in "HMEVTYPE" on page 78.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMFWDED()	Returns a 0, 1, or null to indicate whether an MSU was forwarded from another NetView node, where: **1** An MSU was forwarded from another NetView program through the NV-UNIQ/LUC alert forwarding protocol. **0** An MSU was not forwarded from another NetView program, or was forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol. Examples of when a 0 is returned include: • Local MSUs received over the CNM interface • Local MSUs received from the operating system • MSUs received over the PPI • MSUs received using the SNA-MDS/LU 6.2 alert forwarding protocol **Null** The MSU was not submitted to automation by the hardware monitor RECMS and RECFMS records forwarded from an entry point NetView program to a focal point NetView program by the LUC forwarding method are not submitted to automation by the hardware monitor of the receiving focal point. These RECMS and RECFMS records can be automated only by the sending entry point NetView program. Refer to the *IBM Tivoli NetView for z/OS Automation Guide* for more information about forwarding mechanisms. The maximum length is 1 character. HMFWDED applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMFWDED" on page 78.
HMFWDSNA()	Returns a 0 or 1 to indicate if an MSU was forwarded from a remote entry point node using the SNA-MDS/LU 6.2 alert forwarding protocol. **1** An MSU was forwarded from a remote entry point node using SNA-MDS/LU 6.2 alert forwarding protocol. **0** An MSU was not forwarded from a remote entry point node using SNA-MDS/LU 6.2 alert forwarding protocol. **null** An MSU was not submitted to automation by the hardware monitor. Refer to the *IBM Tivoli NetView for z/OS Automation Guide* for more information about forwarding mechanisms. The maximum length is 1 character. The HMFWDSNA function applies only to MSUs submitted to automation by the hardware monitor. The HMFWDSNA function returns null for all other MSUs. See the example in "HMEPNAU, HMEPNET, and HMFWDSNA" on page 77.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMGENCAU()	Returns the 1-character hexadecimal general cause code of an MSU. The general cause code indicates both the general classification and exception condition that caused the MSU to be created. For more details about general cause codes, refer to the information about basic alert (X'91') alert MS subvectors in the SNA library. HMGENCAU returns null if an MSU is one of the following types or is not submitted to the automation table by the hardware monitor: • A generic alert (X'0000') • A link event (X'0001') • A resolution (X'0002') • A PD statistic (X'0025') • A link configuration data (X'1332') • A statistics-only RECMS • A statistics-only RECFMS Otherwise, a general cause code is returned. Examples of MSUs that HMGENCAU returns a value for include nongeneric alerts (X'0000'), RECMS records that are not statistics-only, and RECFMS records that are not statistics-only. The maximum length is 1 hexadecimal character. HMGENCAU applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMGENCAU" on page 78.
HMONMSU()	Returns 0 or 1 to indicate whether an MSU was submitted to automation by the hardware monitor, where: **1** Indicates that an MSU was submitted to automation by the hardware monitor. **0** Indicates that an MSU was not submitted to automation by the hardware monitor (for example, it was submitted to automation by the generic receiver MS application). The maximum length is 1 character. HMONMSU applies to all MSUs. See the examples in "HMONMSU" on page 78.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMORIGIN()	Returns the name of the resource sending the MSU. Any trailing blanks are removed from the value returned. The resource name returned by HMORIGIN is the same name displayed on the hardware monitor Alerts Dynamic, Alerts Static, and Alerts History panels when `ALT_ALERT ORIGIN` is specified in BNJMBDST. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about the statements used in BNJMBDST. If a complex link does not exist in a resource hierarchy, the resource name returned with HMORIGIN matches one of the resource names returned with the HIER function. If a complex link does exist, the resource name might not be one of the names returned with HIER. Use the HMCPLINK function to determine whether a complex link exists. For more information, see the description of HMCPLINK, on page 69, and the description of HIER, on page 68. HMORGIN returns null if an MSU is not submitted to automation by the hardware monitor. The maximum length is 8 characters. HMORIGIN applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMORIGIN" on page 79.
HMSECREC()	Returns 0, 1, or null to indicate whether the hardware monitor performs secondary recording for an MSU, where: **1** Secondary recording is performed for an MSU at the resource level returned by the HIER function. See the description of HIER, on page 68, for more information. **0** Secondary recording is not performed for an MSU. HMSECREC always returns a 0 for PD statistics (X'0025') and frame relays (X'1332') because the hardware monitor never performs secondary recording for these MSUs. **Null** The MSU was not submitted to automation by the hardware monitor. The maximum length is 1 character. HMSECREC applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMSECREC" on page 79.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMSPECAU()	Returns the 2-character hexadecimal-specific component code of an MSU. The specific component code indicates the generic type of component, subcomponent, or logical resource that is most closely related to the exception condition that caused the MSU to be created. For more details about specific component codes, refer to the information about Basic Alert (X'91') Alert MS subvector in the SNA library. Note that these codes are valid for RECMS and RECFMS records. HMSPECAU returns null if an MSUis one of the following types or is not submitted to the automation table by the hardware monitor: • A generic alert (X'0000') • A link event (X'0001') • A resolution (X'0002') • A PD statistic (X'0025') • A link configuration data (X'1332') • A statistics-only RECMS • A statistics-only RECFMS Otherwise, a general cause code is returned. Examples of MSUs that HMSPECAU returns a value for include nongeneric alerts (X'0000'), RECMS records that are not statistics-only, and RECFMS records that are not statistics-only. The maximum length is 2 hexadecimal characters. HMSPECAU applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMSPECAU" on page 79.

Table 10. Management Services Units (MSU) Information Functions (continued)

Function	Description
HMUSRDAT()	Returns 1 to 5 characters of user-specified data from subvector 33 of an MSU. Trailing blanks are removed from the value returned. This data can be used with hardware monitor filtering. The hardware monitor translates any unprintable data in subvector 33 to underscores (_) and translates lowercase characters to uppercase. The characters returned with HMUSRDAT reflect any translation done by the hardware monitor and therefore might not be the same characters in subvector 33. You can use HMUSRDAT to determine whether the hardware monitor has translated any data in subvector 33 to underscores or uppercase. Although translated data and subvector 33 data are often identical, hardware monitor filtering is performed against the translated data, not against the subvector 33 data. You can use MSUSEG to retrieve untranslated user-specified data from subvector 33 in an MSU. For more information about subvector 33 data, see the UDAT option of the GENALERT command and the U option of the SRFILTER command. HMUSRDAT returns a null if an MSU has the following characteristics: • Does not contain subvector 33. Note that subvector 33 is never present in RECMS or RECFMS records. According to the SNA architecture, only generic major vectors can contain subvector 33. However, the hardware monitor accepts and processes subvector 33 information in any of the major vectors submitted to automation. • Is a frame relay (key X'1332'). • Is not submitted to automation by the hardware monitor. The maximum length is 5 characters. HMUSRDAT applies to all MSUs submitted to automation by the hardware monitor. See the examples in "HMUSRDAT" on page 79.
MSUSEG(*operands***)**	Provides the parsing capability needed to extract information from a management services unit (MSU) or other similarly designed pieces of data. Use this function in a command list that is called by the NetView automation table or an LU6.2 application. For complete MSUSEG syntax and some examples of usage, see "MSUSEG Syntax and Examples" on page 79. For information about the built-in function &MSUSEG, see "&MSUSEG" on page 123.
NPDABA(*operands***)**	Returns the associated hardware monitor probable-cause and error-description text. For complete NPDABA syntax and some examples of usage, see "Probable Cause Syntax and Examples" on page 81.

Hardware Monitor (HMxxxxxx) Examples

HMASPRID

```
/* Example A:  The following example checks for a generic */
/* hardware monitor MSU.                                  */
IF HMASPRID() ¬= '' THEN ....
```

Figure 8. HMASPRID Example A

```
/* Example B:  The following example checks for a generic */
/* MSU from a 3745 device.                                */
IF HMASPRID() = '3745' THEN ....
```

Figure 9. HMASPRID Example B

HMBLKACT

```
/* Example A:  The following example checks for a block id  */
/* and action code that is not null.                       */
IF HMBLKACT() ¬= '' THEN ....
```

Figure 10. HMBLKACT Example A

```
/* Example B:  The following example checks for a block id  */
/* of 'FFD' and action code of '03'.                       */
IF HMBLKACT() = 'FFD03' THEN ....
```

Figure 11. HMBLKACT Example B

```
/* Example C:  The following example checks for a block id   */
/* of 'FFD'.  It does not check for a specific action code.  */
IF SUBSTR(HMBLKACT(),1,3) = 'FFD' THEN ....
```

Figure 12. HMBLKACT Example C

HMCPLINK

```
/* Example A:  The following example checks for an MSU */
/* with a complex link.                               */
IF HMCPLINK() = 1 THEN ....
```

Figure 13. HMCPLINK Example A

```
/* Example B:  The following example checks for an MSU */
/* that has no complex link.                          */
IF HMCPLINK() = 0 THEN ....
```

Figure 14. HMCPLINK Example B

HMEPNAU, HMEPNET, and HMFWDSNA

```
/*====================================================================*/
/* Example A:  Was the MSU was forwarded from node NETA.CNM01         */
/* over LU 6.2?                                                       */
/*====================================================================*/
IF (HMFWDSNA() = '1') &   ,       /* MSU forwarded over LU 6.2?      */
   (HMEPNET() = 'NETA') & ,       /* From network NETA?              */
   (HMEPNAU() = 'CNM01') THEN ... /* And nau CNM01?  Then do ...     */
```

Figure 15. HMEPNAU, HMEPNET, and HMFWDSNA Example

HMEPNETV

```
/*====================================================================*/
/* Example A:  Was the MSU was forwarded from a remote node           */
/* entry point NetView over LU 6.2?                                   */
/*====================================================================*/
IF HMEPNETV() = '1' THEN ...
```

Figure 16. HMEPNETV Example

HMEVTYPE

```
/* Example A:  The following example checks for hardware */
/* monitor MSUs with an event type of PERM.             */
IF HMEVTYPE() = 'PERM' THEN ....
```

Figure 17. HMEVTYPE Example A

```
/* Example B:  The following example checks for hardware */
/* monitor MSUs that do not have an event type of null.  */
IF HMEVTYPE() ¬= '' THEN ....
```

Figure 18. HMEVTYPE Example B

HMFWDED

```
/* Example A:  The following example checks for hardware */
/* monitor MSUs forwarded from another NetView program   */
/* using the NV-UNIQ/LUC..      */
IF HMFWDED() = 1 THEN ....
```

Figure 19. HMFWDED Example A

```
/* Example B:  The following example checks for hardware    */
/* monitor MSUs not forwarded from another NetView program  */
/* using the NV-UNIQ/LUC..      */
IF HMFWDED() = 0 THEN ....
```

Figure 20. HMFWDED Example B

HMGENCAU

```
/* Example A:  The following example checks for a general */
/* cause code that is not null.                          */
IF HMGENCAU() ¬= '' THEN ....
```

Figure 21. HMGENCAU Example A

```
/* Example B:  The following example checks for a general */
/* cause code of '01'X.                                  */
IF HMGENCAU() = '01'X THEN ....
```

Figure 22. HMGENCAU Example B

HMONMSU

Example A shows one way to check for MSUs that have been submitted by the hardware monitor.

```
/* Example A                                  */
IF HMONMSU() = 1 THEN ...
⋮
```

Figure 23. HMONMSU Example A

Example B shows one way to check for MSUs that have not been submitted by the hardware monitor.

```
/* Example B                                  */
IF HMONMSU() = 0 THEN ...
⋮
```

Figure 24. HMONMSU Example B

HMORIGIN

```
/* Example:    The following example checks for hardware    */
/* monitor MSUs sent from a resource named GENALERT.        */
IF HMORIGIN() = 'GENALERT' THEN ....
```

Figure 25. HMORIGIN Example

HMSECREC

```
/* Example:    The following example checks for secondary   */
/* recording on an MSU and displays the resource hierarchy. */
IF HMSECREC() = 1 THEN
   DO
     SAY 'Secondary recording is being done for an MSU at'
     SAY 'resource level: ' HIER()
     SAY 'The name and type pair displayed last is most likely'
     SAY 'involved with the error.'
   END
```

Figure 26. HMSECREC Example

HMSPECAU

```
/* Example A:  The following example checks for a specific  */
/* component code that is not null.                         */
IF HMSPECAU() ¬= '' THEN ....
```

Figure 27. HMSPECAU Example A

```
/* Example B:  The following example checks for a specific  */
/* component code of '0001'X.                               */
IF HMSPECAU() = '0001'X THEN ....
```

Figure 28. HMSPECAU Example B

HMUSRDAT

```
/* Example:    The following example checks for hardware */
/* monitor MSUs with user specified data of MYDAT in     */
/* subvector 33.                                         */
IF HMUSRDAT() = 'MYDAT' THEN ....
```

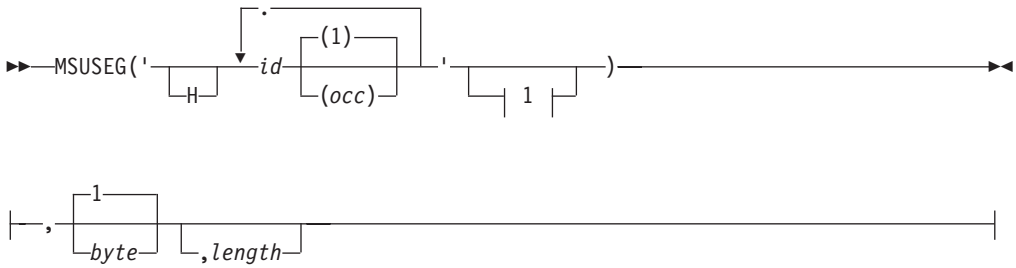
Figure 29. HMUSRDAT Example

MSUSEG Syntax and Examples

Syntax

The MSUSEG(*operands*) has the following syntax:

MSUSEG



where:

byte
Is the byte position into the lowest ID specified in *id*, counting from 1. Position 1 is the first length byte in the header of the lowest ID. The header is composed of one or two length bytes followed by the 1- or 2-byte ID. This entry is optional. The default is 1.

H Is inserted if the first ID is to be obtained from the next higher level multiple-domain support message unit (MDS-MU) as opposed to the NMVT/control point management services unit (CP-MSU) level. You can code the H in uppercase or lowercase. You can place H inside or outside of the single quotation marks when single quotation marks are coded.

id Is the 2- or 4-character representation of the 1- or 2-byte hexadecimal ID of GDS, major vector (MV), subvector, subfield, or sub-subfield. The hexadecimal characters (0 - 9, A - F, a - f) can be mixed case. The first ID is required; additional IDs are optional.

length
Is the number of bytes in decimal to be returned from the lowest ID specified in *id* and starting at the *byte* position. This entry is optional. The default is equal to the remainder of the lowest *id* specified, and starting at the *byte* position.

occ
Is the occurrence number, counting from one (1) in decimal. You can use an asterisk (*) to specify the first occurrence found. This entry is optional at every level. The default is 1.

The single quotation marks shown in the REXX syntax diagram are required only when an *occ* is specified. If you do not explicitly code an *occ*, the quotation marks are optional.

. The period (.) establishes a hierarchy of IDs. Thus, the vector ID specified on the right side of the period is contained within the vector specified on the left side.

Notes:

1. With MSUSEG(*operands*), as with other REXX function operands, if operands are specified, they must be delimited by commas. Two successive commas indicate an omitted operand.
2. If the location is not found, or if the command list containing the MSUSEG(*operands*) was not run by an automation table statement because of an MSU, or if the function was not driven by an MSU, then the value of the MSUSEG(*operands*) is null.
3. If you do not specify a *byte* position, the data returned includes the 1- or 2-byte length followed by the 1-or 2-byte ID of the lowest ID specified in *id*.
4. If the *byte* position is beyond the end of the location, a null value is returned.
5. If the specified length is longer than what remains at the location specified, whatever remains at the location is returned.
6. Examples of using MSUSEG(*operands*) are shown in the figures in "MSUSEG Syntax and Examples" on page 79.
7. For more information about the automation table, refer to the *IBM Tivoli NetView for z/OS Automation Guide*. For more information about vector definitions, refer to the SNA library. For more LU6.2 and MSU information, refer to the *IBM Tivoli NetView for z/OS Application Programmer's Guide*.

8. For information about using the built-in function &MSUSEG in NetView command list language CLISTs, see "&MSUSEG" on page 123.

Examples

The following examples show MSUSEG() function usage.

In Figure 30, the third byte of subvector A0 within the Alert major vector (0000) starts with 'OPEN'. The Alert can be in any of the supported envelopes.

```
IF MSUSEG('0000.A0',3,4) = 'OPEN' .....
```

Figure 30. MSUSEG() Example 1

In Figure 31, Alert subvector A0 has 'LINE' followed by 'DOWN' anywhere in it. Literals can be in hex and EBCDIC.

```
INTERPRET 'PARSE VALUE "'MSUSEG('0000.A0')'" WITH 'LINE'X +4
'DOWN'' Y +4 .'
IF X ¬= '' & Y ¬= '' .....
```

Figure 31. MSUSEG() Example 2

In Figure 32, Alert subvector A1 has bits '01X01X00XX11XXXX', including unimportant bits, starting from the first bit of the fourth byte.

```
IF BITAND(MSUSEG('0000.A1',4,2),'DB30'X) = '4830'X .....
```

Figure 32. MSUSEG() Example 3

Figure 33 shows an MDS-MU whose first 1212 (CP-MSU) contains a 1323, the first of which contains any 1326s, the second of which contains 132Bs, the third of which contains a subvector 01.

```
IF MSUSEG('H1212.1323.1326(2).132B(3).01') ¬= '' .....
```

Figure 33. MSUSEG() Example 4

Probable Cause Syntax and Examples

Syntax

NPDABA has the following syntax:

NPDABA

where:

bbbaa

The 5-character string that defines the pre-generic alert:

- *bbb* is a hardware monitor block ID.

  The block ID is the code used to identify the IBM hardware or software associated with the record.
- *aa* is an associated hardware monitor action-code.

  The action code is the specific alert ID within a block ID.

Note: The NPDABA function returns the associated hardware monitor probable-cause and error-description text (maximum length of 48 characters).

Examples

Table 11. NPDABA Examples

Example	Probable Cause
`NPDABA('FFF03')` The NPDA block ID is FFF and the associated action code is 03.	`'ADAPTER FEEDBK CHK:COMMUN CTRLR PGM/COMMUN CTRLR'`
`NPDABA('123456')` The input is 6 characters, which is not a valid length.	`'INCORRECT INPUT'` The input is not valid (not a valid hexadecimal number or not 5 characters in length).
`NPDABA('12345')` The input is 5 characters, but is not a valid block ID and action code.	`'NOT AVAILABLE'` The hardware monitor does not recognize this combination of block ID (123) and action code (45).

Operator Information Functions

You can use the following operator information function in REXX command lists or Data REXX files for the NetView program.

Table 12. Operator Information Functions

Function or Variable	Description
getpw()	This is intended primarily for use in datarexx. This function can also be used when a REXX procedure is driven from automation. The value is not available when a member is browsed with the BROWSE command or by some other means.
OPID() **&OPID**	Returns the operator or task ID the same as OPID ('O'). OPID is a 1 - 8 character identifier.
OPID('*x*')	Returns the operator or task ID as a 1 - 8 character identifier where *x* is one of the following values: **O** Returns the identity of the owner. On a regular OST, it is the same as OPID(), but on a VOST, it returns the operator ID of the owning OST. **R** Returns the operator ID of a remote task controlling the distributed autotask. If the task is not a distributed autotask, it returns a null. **S** Returns the source ID of the operator that originated the command that is running. The following special values, other than the operator ID, might be returned: **automation** The command originated in the automation table processing. **(null)** The command originated at an optional task or otherwise in assembler code that specified that the source is ignored. **Note:** Currently no case exists in which the NetView program calls REXX in this way. Customer-written code or code from other venders might. **T** Returns the target identity, the identity of the task on which the REXX program is running.

Session Information Functions

You can use the following session information functions in REXX command lists and Data REXX files for NetView.

Table 13. Session Information Functions

Function or Variable	Description
APPLID() **&APPLID**	Returns the application program identifier for the task under which the command list is running. APPLID is the NetView domain ID appended with a 3-character hexadecimal suffix assigned by the NetView program. For example, if your domain ID is PARIS, APPLID might be PARIS001. The NetView program attempts to use an APPLID that is both defined and available. If successful in this attempt, each APPLID is unique. If no defined APPLID is available, an APPLID of `notInit!` is used until a defined APPLID is available. In this case, the `notInit!` APPLID is not guaranteed to be unique as multiple tasks might be in this situation.
ASID() **&ASID**	Returns the current NetView address space identifier. The value of ASID is a 1 - 5 digit decimal number.
ATTENDED() **&ATTENDED**	Returns a single-character value of either 1 or 0. The following values are defined: **1** Indicates that the task is one of the following types: • An OST with a display • An NNT with a corresponding OST • An autotask with an associated MVS console assigned using the AUTOTASK command • A distributed autotask **0** Indicates that the task is one of the following types: • An autotask without an associated MVS console assigned using the AUTOTASK command • Another type of task, such as a DST or an OPT task **Notes:** 1. If the associated operator is an AUTOTASK, the presentation data is not eligible for display unless the AUTOTASK is associated with an active MVS console. 2. ATTENDED can be used with DISTAUTO and AUTOTASK variables to further determine the characteristics of the task. For example, if ATTENDED is 1, DISTAUTO is `0`, and AUTOTASK is 1, the task is an AUTOTASK with an associated MVS console.
AUTCONID() **&AUTCONID**	Returns the MVS console identifier associated with this autotask. This association was made using the AUTOTASK command with the CONSOLE keyword. The value of AUTCONID is the console name of the MVS console where NetView commands can be entered to run under this autotask.
AUTOTASK() **&AUTOTASK**	Returns a single-character value of either 1 or 0 indicating whether the task is an autotask. The following values are valid: **1** An autotask **0** Not an autotask
CGI()	Returns a single-character value of either 1 or 0. The following values are valid: **1** The procedure was called by the NetView web server. **0** The procedure was not called by the NetView web server.
CLOSING()	Returns a value of 1 during post CLOSE command processing; otherwise a value of 0 is returned. Use this function to identify commands that are scheduled to run using the ENDCMD option for a pipe KEEP stage with the GLOBAL option.

Table 13. Session Information Functions (continued)

Function or Variable	Description
CURCONID() **&CURCONID**	Returns the MVS console identifier obtained by a NetView task. This console was obtained with the GETCONID command or by issuing an MVS command. The value of CURCONID is the console name of the MVS console that this task uses to enter MVS commands.
CURSYS() **&CURSYS**	Returns the 1 - 8 character current system name.
DISC()	Returns a single-character value of either 1 or 0 that indicates whether the task is disconnected. The following values can be returned **1** Autotask is disconnected. **0** Autotask is not disconnected.
DISTAUTO() **&DISTAUTO**	Returns a single-character value of either 1 or 0 that indicates whether a task is a distributed autotask started with the RMTCMD command. The following values are valid: **1** A distributed autotask **0** Not a distributed autotask **Note:** This corresponds to the value of TVBDAUT.
DOMAIN() **&DOMAIN**	Returns the 1 - 5 character name of the current NetView domain.
DOMAIN('*x*')	Returns the 1 - 5 character name of a NetView domain, where *x* is the following value **R** Returns the domain name of a remote task controlling the distributed autotask. If the task is not a distributed autotask, it returns a null.
ECVTPSEQ()	Is the z/OS product sequence number found in the MVS IHAECVT data area. For example, if you are running on z/OS v1.6, ECVTPSEQ returns a value of `01010600`.
ENVDATA('*x*')	Returns a numeric value or character string, where *x* is one of the following values **C** Returns the screen color count. **D** Returns the screen depth (number of rows on the screen). **W** Returns the screen width (number of columns on the screen). **G** Returns a list of blank delimited entries representing the REXX, PL/I, and C procedures in the calling sequence or procedure group that was active when ENVDATA was called. Each entry consists of two names separated by a slash (/), in the format *command/name*. *command* is the command verb or synonym used to call the procedure. *name* is one of the following names: • The module name if the procedure is PL/I or C. • The member name in DSICLD if the procedure is REXX. Multiple entries show the calling sequence in reverse order. The command the operator entered is the last entry listed.
JOBNAME(*)	Returns the 1 - 8 character name of the name of the job under which the NetView program is running. An asterisk (*) is the only parameter value allowed.
MVSLEVEL() **&MVSLEVEL**	Returns the version of MVS currently running. For example, if you are running MVS/ESA 4.2.2, MVSLEVEL returns `SP4.2.2`.
NETID() **&NETID**	Returns the VTAM network identifier. This field has a maximum length of 8 characters. If VTAM has never been active when the NetView program is active, the value of NETID is null.

Table 13. Session Information Functions (continued)

Function or Variable	Description
NETVIEW() **&NETVIEW**	Returns the version and release of the currently running NetView program. The value of NETVIEW is a 4-character string in the form of NV*vr*, where: *v* Indicates the version number of the NetView program *r* Indicates the release number of the NetView program
NETVIEW('*x*')	Returns a text string, where *x* has the following value: **T** Returns the text string containing the official NetView name.
OPSYSTEM() **&OPSYSTEM**	Returns the type of operating system for which the NetView program was compiled.
PANEL()	Returns a single-character value of either 1 or 0. The following values are valid: **1** Panel commands can be issued. **0** Panel commands are not allowed.
PARTID() **&PARTID**	Returns the first two characters of the six-character prefix for VSE messages. The two returned characters are the message partition ID only if the sending system uses those characters to designate a partition ID for a message.
STCKGMT() **&STCKGMT**	Returns the current UTC mean time in store-clock format. This field is returned as an 8-byte hexadecimal value.
SUPPCHAR() **&SUPPCHAR**	Returns the suppression character for your installation. (The suppression character prevents the NetView program from writing the command out to the terminal, hardcopy log, and network log.) SUPPCHAR is a single character that you define in the CNMSTUSR or C*xx*STGEN member that is included in the CNMSTYLE member. The default suppression character that is defined in these members is the question mark (X'6F'). If you do not define a suppression character in the CNMSTUSR or C*xx*STGEN member that is included in the CNMSTYLE member, SUPPCHAR defaults to X'3F'. **Note:** The SUPPCHAR default character of X'3F' cannot be typed at the operator console. Therefore, if you do not define a suppression character, the operator is prevented from using one.
SYSPLEX() **&SYSPLEX**	Returns the 1 - 8 character name of the MVS SYSPLEX where the command list is running. Available for MVS/ESA Version 4 Release 2.2 or later. **Note:** This function returns a value only if the command list is running on an MVS SYSPLEX.
TASK() **&TASK**	Returns the 3-character string indicating the type of task under which the command list is running. The following values are possible: **PPT** Primary POI task **OST** Operator station task **NNT** NetView-to-NetView task For Data REXX, in addition to PPT, OST, and NNT, any of these values can be returned: **DST** Data services task **HCT** Hardcopy task **MNT** Main task **OPT** Optional task **UNK** Unknown task **Note:** This value indicates that an error has occurred. Contact IBM Software Support for more information. TASK enables 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, some restrictions apply to command lists running under the PPT. See "Primary POI Task Restrictions" on page 12.

Table 13. Session Information Functions (continued)

Function or Variable	Description
TOWER(*string***)**	Returns either a binary value that indicates whether a tower or subtower is enabled, or the name of the towers and subtowers that are enabled. If a string does not end with an asterisk (*), a single-character of either 1 or 0 is returned. The following values are valid: **1** The tower or subtower is enabled. **0** The tower or subtower is not enabled. For example, assume that the AON tower and the SNA subtower are enabled, but the TCP subtower is not, `SAY TOWER(AON.SNA)` returns 1 and `SAY TOWER('aon.TCP')` returns `0`. Strings that end with an asterisk (*) return the names of the towers and subtowers that are enabled. Note that asterisks can be either used alone or used together with a tower name to determine the subtowers that are enabled. For example, if the AON tower and the SNA and TCP subtowers are enabled, `SAY TOWER('*')` returns `AON SNA TCP` and `SAY TOWER('aon.*')` returns `SNA TCP`. Input strings are not case-sensitive and mixed case strings can be returned. Tower and subtower combinations must be concatenated with a period (.). Towers and subtowers are enabled in the CNMSTUSR or C*xx*STGEN member that is included in the CNMSTYLE member. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information.
TRAP()	Returns a binary value that indicates whether a TRAP command will fail because of a conflicting correlation environment. Values are: **1** TRAP command is acceptable. **0** TRAP command will fail with message DWO373E.
TYPE()	Returns a 3-character string that indicates the level of the NetView program that is installed. The following values are possible: **ENT** Enterprise option **SYS** NetView System Services
VTAM() **&VTAM**	Returns the version and release of VTAM as a 4-character string in the form of either VT*vr* or V*vrm*, where: *v* Is the version number *r* Is the release number *m* Is the modification number **Note:** The value of VTAM is null if the VTAM program is not active.
VTCOMPID() **&VTCOMPID**	Returns the 14-character VTAM component identifier. The following list shows the VTAM component identifiers: **MVS/ESA** 5685-08501-*xxx* (for VTAM Version 3) **z/OS** 5695-11701-*xxx* (for VTAM Version 4) and later **MVS/XA** 5665-28901-*xxx* **VSE/ESA** 5666-36301-*xxx* **VM/SP** 5664-28001-*xxx* **VM/9370** 5684-05201-*xxx* **VM/ESA®** 5684-09501-*xxx* where *xxx* is the release number. Additional VTAM component identifiers might be added in future updates to VTAM. The value of VTCOMPID is null if VTAM is not active.

Table 13. Session Information Functions (continued)

Function or Variable	Description
WEEKDAYN() **&WEEKDAYN**	Returns a numeric value in the range of 1 -7 indicating the day of week (from Monday through Sunday), as shown here: **1** Monday **2** Tuesday **3** Wednesday **4** Thursday **5** Friday **6** Saturday **7** Sunday

REXX Environment Information Functions

You can use the following REXX environment functions in REXX command lists for the NetView program.

Refer to the *IBM Tivoli NetView for z/OS Tuning Guide* for the rationale on the use of these functions.

Note: Refer to the DEFAULTS command and the OVERRIDE command in the NetView online help for more information about the meaning of the following REXX values. These functions return a null value for operating systems other than MVS/XA, MVS/ESA, and VSE/ESA.

Table 14. REXX Environment Information Functions

Function or Variable	Description
RXDEFENV() **&RXDEFENV**	Returns the default number of NetView REXX environments set by the REXXENV parameter of the DEFAULTS command.
RXDEFSTR() **&RXDEFSTR**	Returns the default NetView REXX environment initial storage size set by the REXXSTOR parameter of the DEFAULTS command. This value can be -1 if REXXSTOR was set to the default or was never set.
RXNUMENV() **&RXNUMENV**	Returns the current number of REXX environments initialized for this task. For RXNUMENV(), this number is always at least 1, representing the REXX environment currently running. For &RXNUMENV, this number can be zero (0).
RXOVRENV() **&RXOVRENV**	Returns the override number of NetView REXX environments set by the REXXENV parameter of the OVERRIDE command. If the number of REXX environments has not been overridden or is set to the default value, a null value is returned.
RXOVRSTR() **&RXOVRSTR**	Returns the override NetView REXX environment initial storage size set by the REXXSTOR parameter of the DEFAULTS command. If the REXX initial storage size has not been overridden or is set to the default value, a null value is returned.

Terminal Information Functions

You can use the following terminal information functions in command lists for NetView.

Table 15. Terminal Information Functions

Function or Variable	Description
HCOPY() **&HCOPY**	Returns the name of device defined as the hardcopy log printer started by the operator. If no device is defined as the hardcopy printer for this operator, HCOPY is null.

Table 15. Terminal Information Functions (continued)

Function or Variable	Description
LU() **&LU**	Returns the logical unit name for this operator terminal.

Time and Date Variables

You can use the following time and date control variables in the NetView command list language:

Table 16. Date and Time Variables

Function or Variable	Description
&DATE	Returns the current date in the form of *mm/dd/yy*, where *mm* is the month, *dd* is the day, and *yy* is the year.
&TIME	Returns the processor 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 3:00 p.m. is shown as 15:00.

Note: Because &TIME and &DATE are separate variables, you might need extra coding to determine the correctly matched time and date. For example, if you get &DATE first, midnight can occur before you get &TIME, so you have the wrong date for the current time. If you get &TIME first, midnight can occur before you get &DATE, and then you have the wrong time for the current date.

The following example shows some NetView command list language code that you can use to determine if you have the correct date and time:

```
-RETRY
    &TDATE = &DATE
    &TTIME = &TIME
    &IF &TDATE NE &DATE &THEN &GOTO RETRY
    &WRITE &TDATE &TTIME
```

REXX provides equivalent but more comprehensive time and date functions. For more information, refer to the REXX library.

Nulls and Blanks Stripping

The stripping (removal) of trailing nulls and blanks is automatically performed by the NetView program on some of the NetView command list language control variables and NetView REXX functions that have character values. Notice that some control variables and REXX functions have different levels of trailing character removal.

Function or Variable	Stripping Provided
ACTIONDL(), &ACTIONDL	Nulls and blanks
ACTIONMG(), &ACTIONMG	Nulls and blanks
APPLID(), &APPLID	None
AREAID(), &AREAID	None
AUTCONID(), &AUTCONID	Nulls and blanks
AUTOTOKE(), &AUTOTOKE	Nulls and blanks
CURCONID(), &CURCONID	Nulls and blanks
CURSYS(), &CURSYS	Nulls and blanks
CMDNAME()	Blanks
DCO(), &DCO	None
DOMAIN(), &DOMAIN	Nulls and blanks
HCOPY(), &HCOPY	Blanks
HMASPRID()	Blanks
HMEVTYPE()	Blanks
HMORIGIN()	Blanks
HMUSRDAT()	Blanks
IFRAUI3X(), &IFRAUI3X	Nulls and blanks
IFRAUSB2(), &IFRAUSB2	Nulls and blanks
IFRAUSDR(), &IFRAUSDR	Nulls and blanks
IFRAUSRC(), &IFRAUSRC	Nulls and blanks
JOBNAME(), &JOBNAME	Blanks
JOBNUM(), &JOBNUM	None
LU(), &LU	Blanks
MSGCOJBN(), &MSGCOJBN	Nulls and blanks
MSGCPROD(), &MSGCPROD	Nulls and blanks
MSGCSPLX(), &MSGCSPLX	Nulls and blanks
MVSLEVEL(), &MVSLEVEL	Nulls and blanks
NVDELID(), &NVDELID	None
OPID(), &OPID	Blanks
SESSID(), &SESSID	Blanks
SYSCONID(), &SYSCONID	Nulls and blanks
SYSID(), &SYSID	None
SYSPLEX(), &SYSPLEX	Nulls and blanks

Chapter 5. Automation Resource Management

This chapter is intended to help you perform NetView automation using command lists.

Defining NetView Automation Table Command Lists

The automation table identifies which messages or MSUs are automated. It consists of statements filed in a member of DSIPARM. The statements identify which messages or MSUs are to be automated based on almost any attribute of the message or MSU, such as:

- Message number
- Specific MSU field values
- Origin of message or MSU

As a result, the automation table can change display, logging, routing, or almost any other disposition of the message. Commands or command lists can be called to analyze the message or MSU either before or in addition to any action taken.

To define an automation table, code automation statements in a member of DSIPARM and then issue the AUTOTBL command using the name of that specific NetView automation table. You can enter the AUTOTBL command anywhere a regular command can be issued, and you can activate a table from the CNMSTYLE member.

Notes:

1. A regular command is a command or command list defined with TYPE=H or TYPE=R. See *IBM Tivoli NetView for z/OS Administration Reference* for further explanation of a regular command.
2. You cannot run the AUTOTBL command under a DST.

For a complete definition of the syntax of the NetView automation statement, refer to the *IBM Tivoli NetView for z/OS Automation Guide*. For the syntax of the AUTOTBL command, refer to the NetView online help.

Routing Messages from Automation-Table-Driven Command Lists

You might have difficulty deciding where to route a message from the NetView automation table. To decide, cause a command list to be driven and use the MSGROUTE command or the ROUTE stage to route the message to operators or groups of operators.

For more information about the MSGROUTE command, refer to the NetView online help.

Implementing NetView Automation

This section provides suggestions to help you implement NetView automation. For more information, refer to the *IBM Tivoli NetView for z/OS Automation Guide*.

Suppressing Messages

You can suppress some messages so that no operator receives them. To suppress messages with NetView automation, make an entry in the NetView automation member. Assume, for example, that you do not want the message `IST4001 TERMINATION IN PROGRESS FOR APPLID applnm` to be displayed. The following example shows the NetView automation statement:

```
IF MSGID='IST400I' THEN DISPLAY(N);
```

Determining the Environment for a Command List

In REXX, if you are not sure of the type of task or condition under which a command list is to run, have the command list check the TASK() function, the PANEL function, or the AUTOTASK function. You can then use conditional processing to make the command list flexible enough to run differently under different tasks.

In NetView command list language, if you are not sure of the type of task or condition under which a command list is to run, have the command list check the &TASK control variable or AUTOTASK function 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 Appendix B, “NetView Command List Language Branching,” on page 129 for more information about conditional processing.

Testing Automation Command Lists

An automation command list can be tested in several ways to ensure that it is called correctly from the NetView automation table and processes correctly after being called.

Verifying Proper Operation of Automation Command Lists

To thoroughly test your REXX automation, you might want to verify proper automation of your NetView CLIST before you put it in the AUTOTABLE. The NetView program provides a means to simulate the current message that is present when your REXX program is driven from the automation table. Sometimes it is sufficient to use the LITERAL stage to create your simulated message. However, LITERAL cannot set message attributes, such as job name or automation token, that might be important to your automation. If these message attributes are relevant to your testing, you can obtain an exact copy of the message for testing purposes by temporarily coding the HOLD(Y) automation action in place of the planned action.

After this temporary automation is called, with the subject message held on your NCCF screen, you can use the HELDMSG stage to create accurate copies for testing. You can test this by issuing the TS command to cause tracing and then issue a command such as this:

```
PIPE HELDMSG | NETV CMD (HIGH) yourcmd yourarguments
```

If more than one held message is on the NCCF screen, you might also add a LOCATE or TAKE stage. After the PIPE HELDMSG command completes, `yourcmd` is running outside of pipelines, and functions such as JOBNAME() and PIPE SAFE * in your procedure return accurate results from the current message provided by the pipeline.

You can stop this command and then reissued the TS and the pipe command to test repeatedly.

Verifying NetView Automation Table Entries

You can verify that an automation command list is driven correctly by the NetView automation table by issuing the message from an operator station or command list. From a NetView operator console, enter the message ID and message text from the command line.

From a REXX command list, use the SAY instruction with the message ID and message text in quotation marks.

From NetView command list language, use the &WRITE statement with the message ID and message text in quotation marks.

If the message you create matches an entry in the NetView automation table, the table processes any actions specified for that entry. Through this process, you can test NetView automation table entries. This method works only if limited information, such as the message identifier and message text, is checked in the NetView automation table entry.

By using the AUTOCNT command with the `STATS=DETAIL` option, detailed information, including the number of automated comparisons and matches, are shown for each automation table statement. When your created message is automated, the count of the number of comparisons and matches is incremented if the message matches the intended automation statement.

Keeping a Record of Automation Command Lists Processed

Command lists can use the NetView MSG command to place information in the network log. This transfer of information might be necessary because not all command lists are run directly from the NetView automation table. By having automation command lists send a message to the network log using the NetView PIPE LOGTO statement, you can track which automation command lists are driven by which tasks and at what time.

Testing Automation Command List Processing

To test REXX automation command lists by tracing their processing, use the TRACE command. If your command list is being run by a NetView automated operator (autotask), the result of SAY or TRACE is not displayed unless the autotask is assigned an MVS console. The results are displayed in the network log regardless of whether the autotask is assigned to a console.

To test NetView command list language automation command lists by tracing their processing, use the &CONTROL statement. If your command list is being run by a NetView automated operator (autotask), the results of &WRITE, &BEGWRITE, or &CONTROL are not displayed unless the autotask is assigned an MVS console. The results are displayed in the network log regardless of whether the autotask is assigned to a console. Refer to the AUTOTASK command in the NetView online help for more information about assigning an autotask to an MVS console.

Looping and Automation

Messages issued from command lists are subject to automation unless you take measures to prevent it. If a message issued by a command list causes the same command list to be driven, then a looping situation occurs. In some cases, such looping can involve multiple messages and multiple tasks. If you know which operators or autotasks are involved in the loop, you might be able to end the loop using the STOP FORCE command or the RESET command. In extreme cases, you must disable the automation that is creating the loop; see the AUTOTBL command.

You can prevent a displayed message from causing a loop in either of two ways:

- Display from a pipeline using CONSOLE ONLY
- Alter the message so that it does not match the automation statement.

Another way to avoid a loop is to display the original message with its attributes intact. One of those attributes is already automated. An example is

```
PIPE SAFE * | CONSOLE
```

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 instruction requiring a GO command is not appropriate.

For example, in NetView command list language, using either the &PAUSE or &WAIT statement and requiring a GO command is not appropriate.

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 operators enter requests.

Common Automation Problems

Because NetView automation is called after the command facility exit routines (for example, DSIEX02A, DSIEX06, and DSIEX11) are called, changes made to messages in these routines affect NetView automation. For example, if a message is deleted by DSIEX02A, Tivoli NetView for z/OS does not call automation for that message. If a message is assigned to SYSOP or LOG as the primary receiver, Tivoli NetView for z/OS does not call automation for that message. Because NetView automation does not occur in the preceding instances, the DISPLAY keyword in the NetView automation member has no effect.

If the MVS 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. For REXX, you must change the setting of the MCSFLAG variable for the WTO to be displayed. For command list, you must change the setting of the &MCSFLAG control variable for the WTO to be displayed. Refer to the PARSEL2R command in the NetView online help for an example of how to change a function or control variable.

If automating a message that was marked non-displayable using a NetView Message Revision table or the MVS message processing facility, then a message you create using the WTO will, by default, inherit that non-displayable characteristic. You can control this behavior using the MCSFLAG variable. For example, code MCSFLAG = '00000000'.

You can discover whether a message has been subject to Message Revision Table (MRT) action by using the MRT edit order (see help for PIPE EDIT). When automating a message subject to the MRT, a message you issue using the WTO command is not submitted to the MRT. However, you can control this behavior using the MRT edit order as an output order.

If a multiline write-to-operator (MLWTO) message is used to drive a command list and a WTO is issued from the command list, whether the WTO is displayed depends on the setting of the MLWTO line type variable. If the setting of WTO is a single-line message, change the setting to a blank.

The REXX MLWTO line type variable is LINETYPE.

The NetView command list language MLWTO line type variable is &LINETYPE.

Appendix A. Writing Simple Command Lists in the NetView Command List Language

This chapter explains the basics of writing command lists for the Tivoli NetView for z/OS 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.

What the NetView Command List Language Includes

The NetView command list language consists of six types of statements:
- Command
- Comment
- Control
- Assignment
- Label
- Null

Within command list statements, you can use the following variables and functions:
- Parameter variables
- Control variables
- User variables
- Global variables
- Built-in functions

All except global variables are described in detail in later sections of this chapter. Global variables and descriptions of passing parameter values are described in Appendix C, "NetView Command List Language Global Variables," on page 147.

Use the NetView command list language to write application code to perform repetitive or alternate processing (loop or if-then structures). These features are implemented with the following control statements:
- &IF
- &GOTO
- &EXIT
- &WAIT

Control statements are described in Appendix B, "NetView Command List Language Branching," on page 129.

Note: Command lists can interrupt the processing of other command lists. This is done using the CMDDEF statement in CNMCMD.

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 Tivoli NetView for z/OS are divided into sections describing the conventions for:
- General coding
- Continuing a statement
- Double-byte character sets

- Suppression characters

Conventions for General Coding

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

- Code a CLIST statement as the first line of your command list; the CLIST statement is optional. Code CLIST statements in the following way:
  - Optionally, code a label. The label must begin in column 1. You cannot branch to this label; the Tivoli NetView for z/OS program ignores the label.

    label CLIST
  - Code the word CLIST beginning in column 2 or later. The word CLIST must be preceded by at least one blank.
- Do not code the name of the command list on the first line unless accompanied by the word CLIST.
- 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 one exists) 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 alphabetic characters A-Z.
- Code statements so that the maximum length is 32 000 characters after variable substitution.

  Note: To familiarize yourself with how variable substitution works, see "Variable Substitution Order" on page 101.
- Code comment lines with an asterisk (*) as the first non-blank character of the command list line. Place the comment after the asterisk. Comment lines cannot be coded 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 Continuing a Statement

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 nonblank 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 nonblank 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 Tivoli NetView for z/OS 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 is replaced by a blank. When filling a line with characters from the beginning of the continuation line, Tivoli NetView for z/OS 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 are kept. The first line writes 64 characters to the output screen (43 characters of text plus 21 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 causes the first four characters of the second line to be placed at the end of the first line. In the example, this is two blanks and the letters BL. However, because the Tivoli NetView for z/OS program does not split a word across lines of the output screen, the following message is displayed:

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

Conventions for Double-Byte Character Set Text

In a double-byte character set (DBCS), each symbol is represented by a 2-byte code rather than a 1-byte code.

- Use A - Z, 0 - 9, @, $, and # characters to code Tivoli NetView for z/OS commands and command lists used as commands. The command list name must begin with a nonnumeric character.
- DBCS data input is not supported.
- Enclose all DBCS strings within shift-out (X'0E') and shift-in (X'0F') control characters. Be sure that each DBCS string has an even number of bytes. (If you are using an editor and terminal that supports double-byte characters, this is done automatically.)
- You can code label names, variable names, and variable values in DBCS characters. Restrict variable names and label names to a length of 11 bytes. These include shift-out (X'0E') and shift-in (X'0F') control characters.
- When DBCS labels and variables are displayed on a DBCS terminal, the shift-out and shift-in control characters are displayed as blanks.
- DBCS text can be split across multiple lines, using an EBCDIC plus sign (+) or hyphen (-) as a continuation character. To split a string, end the string with a shift-in (X'0F') control character followed by the continuation character. Start the next line with a shift-out (X'0E') control character to resume the string.
- When writing DBCS text in a &BEGWRITE statement, the SUB option is required.
- Comments can contain DBCS strings enclosed by shift-out (X'0E') and shift-in (X'0F') control characters.
- &WRITE, &CONCAT, and &SUBSTR are enabled for DBCS.

Conventions for Suppression Characters

The following rules apply when coding suppression characters:

- The first nonblank character before a command is the suppression character.
- When you browse or list a file, you can see every line, even suppressed lines.

- In general, do not use suppression characters preceding a NetView CLIST language label. The suppression character prevents you from branching to the label unless the command list line containing that label has already been processed.

In Figure 34, the control variable &SUPPCHAR is replaced with the character defined as a suppression character. The last line of the command list in the example is suppressed.

```
&CONTROL CMD
* COMMAND LIST UPDATED 2/5/95 BY OPERATOR CARL
START DOMAIN=&1
&WRITE ENTER GO WHEN MESSAGE DSI809I ARRIVES FROM &1
&PAUSE
&SUPPCHAR ROUTE &1,OPER1,123456
```

Figure 34. Example of Using 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 use the following code in a command list, no synchronous output from the command list is displayed to the operator:

```
&DOUBLESUPP = &CONCAT &SUPPCHAR &SUPPCHAR
&DOUBLESUPP SET PF24 IMMED RETRIEVE
```

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

Labels

Labels identify command list statements for control of flow, for internal documentation, or to indicate the target statement for a transfer of control. Transferring control is explained in Appendix B, "NetView Command List Language Branching," on page 129.

You can code labels on any command list statement except a comment statement. You can code labels on commands, control statements, assignment statements, and null statements. If Tivoli NetView for z/OS cannot find the label, processing stops, and an error message is issued.

A label must be the first nonblank word on a command list line. A label consists of an EBCDIC hyphen (-) followed by 1 to 11 characters. A - Z, 0 - 9, #, @, and $ are valid characters. You do not have to code a command list statement after a label. If you do, however, start the command list statement after the label, leaving at least one blank between the label and a keyword.

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, Tivoli NetView for z/OS 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.

The following examples are labeled command list statements:

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

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 Appendix B, "NetView Command List Language Branching," on page 129.

Variables

Use variables to accept from an operator, or define for yourself, different values for the statements within a command list. With the following variables, you can write a command list that operates correctly in many different situations:

- Parameter
- Control
- User
- Global

This section describes 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 Appendix C, "NetView Command List Language Global Variables," on page 147 for a description of global variables.

Code the variable as the first nonblank word in the command list.

A variable consists of an EBCDIC ampersand (&) followed by 1 - 11 characters. A - Z, 0 - 9, #, @, and $ are valid characters.

Variable Substitution Order

Variable substitution is performed when the Tivoli NetView for z/OS program scans each statement from right to left and substitutes values for each variable in the following way:

1. Each element is scanned from right to left for an ampersand (&).
   - 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. The Tivoli NetView for z/OS program then scans to the right and takes any following numbers as part of the parameter variable. When a blank or a letter is found, the search stops. If a special character (non-alphanumeric) is found, the variable name is delimited.

     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, the value of &XYZ if substituted, and then &A is replaced with the value substituted for &XYZ.

     Note: The value of X'50' (ampersand in the EBCDIC character set) is ignored within double-byte character sets. If you want to use an ampersand, end the string using a shift-in (X'0F') control character and enter the variable. To resume the string, begin the string using a shift-out (X'0E') control character.

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 the variable value.

   Note: 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 is to be assigned a value. For example, the statements in the following example set the value of user variable &A1 to 2.

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

Variable substitution is not done on the following items:

Control keywords
For more information, see "&CONTROL Statement" on page 110.

&PAUSE statement
The variables are assigned values when you enter a GO command. For more information, see "&PAUSE Control Statement" on page 114.

&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 is coded separately. For more information, see "&IF Control Statement" on page 129.

Any statements in an &BEGWRITE NOSUB series of messages
For more information, see "&BEGWRITE Control Statement" on page 113.

Built-in functions
For more information, see "NetView Built-in Functions" on page 117.

Parameter Variables

A parameter variable is a positional variable that is defined at the time a command list is run. 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 numeric or character values.
- When used in an arithmetic expression (for example, addition or subtraction), a parameter variable can have a numeric value between -2147483647 and 2147483647. When used in a non-arithmetic expression (for example, assignment statements, &IF statements, &CONCAT, or &SUBSTR statements), a parameter variable can have a value up to 238 digits long, including the sign.

Note: When Tivoli NetView for z/OS receives a message coded in an &WAIT statement, some control variables are set (for example, &MSGORIGIN,

&MSGID, &MSGCNT, and &MSGSTR) and the values of the parameter variables (&1 – &31) are changed to reflect the information in the received message.

See "Control Variables" on page 106 for information about these variables. LINKPD sets the same control and parameter variables. See "LINKPD Results" on page 157 for more information about the LINKPD command.

Passing Parameter 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. The following example shows the format for an operator passing up to 31 parameter variables to a command list:

```
cmdlistname  _____,_____,_____,. . .,_____

              &1    &2    &3          &31
```

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 is 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 the following example.

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

If you want the command list to use parameter variables, you can change it to activate or deactivate any resource. The following example shows how the command list looks with parameter variables:

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

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

When the command list runs, &1 and &2 are replaced with the following positional parameters:

- &1 with ACT
- &2 with LU100.

The command list takes the values for &1 and &2 from the entered operands in the order in which the operands are 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.

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 5, "Automation Resource Management," on page 91.

Using Parameter Variables in a Command List

No set order is required for placing the parameter variables in the command list. The following example shows that you can use &2 before &1.

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

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

If a command list statement has two or more parameter variables, the rightmost variable is changed first; the scan then continues right to left and the next variable is replaced. You can use this method to change the meaning of some of your parameter variables. If you must 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 106.

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. In addition, you can pass either actual values or other variables as parameter variables. If you pass other variables, make sure that these variables are known to the next activated command list.

The following examples show passing parameters.

Command list CALLER contains a line of code such as:

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

Command list CALLEE uses the following variables:

&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
```

Command list MINOR uses the following variables:

&1 55

&2 ALPHA

&3 BETA

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

If you must pass a nested command list a variable containing a quoted string, enclose the variable in single quotation marks on the nested command list call. In the following example, CLIST1 calls CLIST2:

```
CLIST1 CLIST
&STR = &1
CLIST2 '&STR'
&EXIT
```

The parameter variable on the nested command list call must be surrounded by quotation marks.

Using Quoted Strings or Special Characters in Parameter Variables

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 quotation marks. 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 NetView command list language quoted string is any text that meets one of the following requirements:

- Text preceded by a delimiter and a single quotation mark, followed by either a single quotation mark and a delimiter or a single quotation mark that is the rightmost nonblank character.
- Text preceded by a single quotation mark that is the leftmost nonblank character, followed by a single quotation mark and a delimiter.
- Text preceded by a single quotation mark that is the leftmost nonblank character, followed by a single quotation mark that is the rightmost nonblank character.

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

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

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

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

Suppose you activated the RESC command list by entering this command:

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

The parameter variables in this case contain these values:

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

Null Parameter Values

Use two commas (,,) 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 that you wrote a command list called CONN that contained the following statement:

```
BGNSESS FLSCN,APPLID=&1,SRCLU=&2,LOGMODE=&3,INT=&4,D=&5
```

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

```
CONN TSO,TAF01F00,,,PF12
```

In this example, &1 is TSO, &2 is TAF01F00, &3 and &4 are null, and &5 is PF12. The extra commas between TAF01F00 and PF12 represent positional place holders for &3 and &4, and tells the command list that they are null. If you use only one comma, the command list takes PF12 as &3 and incorrectly uses PF12 as the LOGMODE.

Test for null parameter variables in your command list and provide default values to avoid possible syntax errors.

Control Variables

The following sections describe the control variables as used in NetView command list language.

Control variables are set by 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, the correct values are assigned to each control variable. Use the LISTVAR command to view the values of some of the control variables.

For more information about control variables used with the SPCS commands LINKDATA and LINKTEST, see "LINKDATA and LINKTEST Results" on page 156.

Note: A command list can create a user variable that has already been defined as a control statement, control variable, or built-in function. However, if such a user variable is created, you cannot use the provided control statement, control variable, or built-in function in that command list anymore.

User Variables

User variables are variables you create and set within the command list. You can set user variables with an assignment statement or an &PAUSE control statement.

Assignment statements are explained in "Assignment Statements" on page 108.

The &PAUSE control statement halts the command list so that the operator can 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 114.

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, it is treated as a parameter variable.
- The ampersand must be followed by 1 to 11 characters. A - Z, 0 - 9, #, @, and $ are valid characters.
- The value of the user variable can be 255 characters long. The maximum number of double-byte characters between the shift-out (X'0E') and shift-in (X'0F') control characters is 126.
- A user variable can have a numeric value that is 255 digits long, including the sign. However, if the value of the user variable is obtained using an arithmetic expression (for example, addition or subtraction), or if the user variable is used in an arithmetic expression, the user variable can have a numeric value between -2147483647 and 2147483647. The only characters you can use in a numeric value are 0-9. The numeric value can be immediately preceded by a character indicating whether the value is positive (+) or negative (-).

Note: A command list can create a user variable that has already been defined as a control statement, control variable, or built-in function. However, you cannot use the provided control statement, control variable, or built-in function anymore in the command list.

Table 17 on page 107 shows some examples of user variable names.

Table 17. User Variable Names

Valid	Non-valid	Reason
&A	&2A	Is read as &2, a parameter variable
&USERNAME	&INVALIDUSERNAME	Too long
&@23456	&A%	% is not a valid character

The following example 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
```

Hexadecimal Notation

The NetView command list language provides a hexadecimal notation capability to process hexadecimal data. You can use hexadecimal notation anywhere you can use a command list variable, except as the receiver in an assignment statement.

The following syntax describes the hexadecimal notation for the NetView command list language:

where:

n Is an even or odd number of hexadecimal digits (0-9 or A-F in uppercase) with no embedded blanks. If *n* is an odd number of digits, then the number is prefixed with a zero (for example, X'2C6' is converted to X'02C6'). The maximum length of *n* is 255 hexadecimal digits.

The following examples show the use of hexadecimal notation:

```
&A = X'3B9' &IF &A = X'3B9' &THEN....
```

Comments

It can be helpful to code comments in a command list. Command lists with comments are easier to maintain and expand than command lists without comments.

You can use comments to show the following information:

- 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 the command list depends on other programs or on other command lists.

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

Null Statements

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 100 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, the statement is ignored when the command list is run.

Assignment Statements

Assignment statements give values to variables and do arithmetic operations within a command list. An assignment statement has the following syntax:

assignment

Figure 35. Assignment Statement

A blank must be 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, the assignment statement &A = 5 sets the &A to 5. The assignment statement &B = &1 sets the &B to the value of &1, and &1 keeps its value.

An expression is one of the following items:

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 that contains a blank, comma, apostrophe, or hyphen, use single quotation marks. For example:

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

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 numeric value are 0-9. The numeric value can be immediately preceded by a character indicating whether the value is positive (+) or negative (-).

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 statement:

```
&USERVAR = &OPID
```

Note: Using a control statement as a variable is not valid, even if the control statement is enclosed in single quotation marks. For example, the following assignment statements are not valid:

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

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 is to be set to a number. The only characters you can use in a numeric value are 0-9. The numeric 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 does not work.

The result 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 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 statement:

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

The value of &C becomes 2, not 02.

Note: To avoid an error condition in an arithmetic operation, code a zero before a potential null variable.

Built-in function

You can use a built-in function in an assignment statement. The result of the operation is placed in the user variable. See “NetView Built-in Functions” on page 117 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
```

Control Statements

Control statements are unique command list statements that control the way the Tivoli NetView for z/OS acts on other statements in the command list. You can use the control statements in this chapter either for straight-line coding or with the statements described in Appendix B, "NetView Command List Language Branching," on page 129 for conditional processing.

You can use control statements to change the sequential order of processing. Using command list control statements, you can take the following actions:
- 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.

Reading the descriptions of the control statements provides a general idea of the capability of these basic statements. Read the sections that follow for details concerning each control statement.

The control statements follow:

&BEGWRITE
Writes a message or series of messages to the operator.

&CONTROL
Indicates the command list statements that are shown on the operator screen while the command list is running.

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

&WRITE
Writes a message to the designated operator.

&CONTROL Statement

The &CONTROL statement lets you indicate which command list statements are displayed at the operator 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. You can use the display of the command list statements from &CONTROL ALL or &CONTROL CMD 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 example, 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 an operand. The &CONTROL control statement has the following syntax:

&CONTROL

where:

ALL
The ALL control statement displays all command list statements at the operator terminal. Each statement is displayed just before it is processed. &CONTROL ALL is a good choice when you first write the command list and want to test it. After your command list is tested, the &CONTROL CMD control statement or the &CONTROL ERR control statement is a better choice. When processing for this command list is complete, the following message is displayed:

```
DSI013I COMMAND LIST clistname COMPLETE
```

If you code the &CONTROL control statement without operands, or if you do not code the &CONTROL control statement, the default control statement is &CONTROL ALL.

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

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

Writing to the Operator

The &WRITE and &BEGWRITE statements send messages to the operator terminal. The &WRITE statement sends a one-line message, and the &BEGWRITE statement sends multiline messages. These statements are used to give the operator information, such as 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 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. The &WRITE statement and the &BEGWRITE statement 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 use the NetView VIEW command instead of using the &WRITE statement and the &BEGWRITE statement

&WRITE Control Statement

The &WRITE statement sends one line of text to the operator. Tivoli NetView for z/OS 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, a blank line is sent to the operator. The &WRITE statement has the following syntax:

&WRITE

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

In the following line:

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

the period causes the line to print like this:

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

Otherwise, the line shifts left until the first nonblank 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
```

The following example shows 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
```

Activating this command list by entering PATH HD3790N1 causes Figure 36 to be displayed.

```
*** 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 36. 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 37 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 ***
```

Figure 37. Sending One-line Messages to the Operator

&BEGWRITE Control Statement

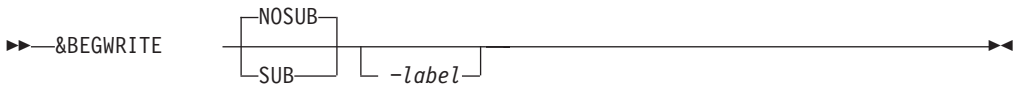
You can use &BEGWRITE to write a series of lines to the operator terminal. You can also control whether variables are replaced before sending the messages.

Code the &BEGWRITE statement on a line by itself, one line above the first operator message you want 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 100 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.

The &BEGWRITE statement has the following syntax:

&BEGWRITE



where:

-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 the label cannot be found, the rest of the command list statements are sent to the operator as comments and the command list is ended. If &BEGWRITE has no label, only the first command list statement after &BEGWRITE is sent to the operator.

  You can code a variable for your label on &BEGWRITE. Replace the variable with a valid value.

NOSUB
: 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, NOSUB is assumed.

SUB

Causes Tivoli NetView for z/OS to carry out substitution on the message text before sending the messages to the operator. See "Variable Substitution Order" on page 101 for information about how the variable substitution is managed.

If blanks precede 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 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.

Figure 38 is an example of a &BEGWRITE statement 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 38. &BEGWRITE with Variable Substitution

In some cases, you might not want variable substitution. In the following example, the &BEGWRITE statement 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
```

The ENDIT command list is called by entering ENDIT. Figure 39 shows the messages that the operator sees when ENDIT is used.

```
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 39. 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

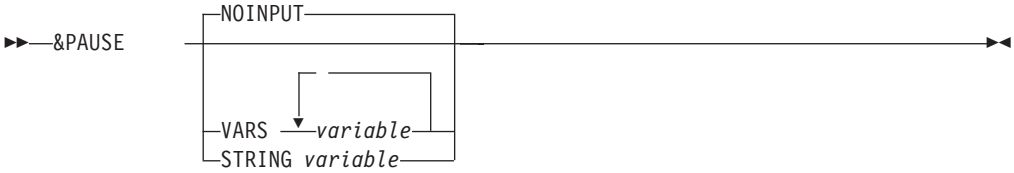
command list pauses until the operator enters the GO command to continue processing, or the RESET command to end the command list. You can code the &PAUSE command to enable the command list to pick up the operands following the GO commands and take them as user variables. See "User Variables" on page 106 for more information.

Notes:

1. Using &PAUSE in an automation task command list or a command list that runs under the PPT is not valid.
2. The VIEW command obtains operator input without requiring the use of the GO command.

The &PAUSE statement has the following syntax:

&PAUSE



where:

NOINPUT
Pauses until the operator enters the GO or RESET command. Operands cannot be specified with the GO command. If the operator enters operands, an error message is returned. NOINPUT is the default.

STRING *variable*
Pauses until the operator enters the GO command with or without a string, or the RESET command. A previous &WRITE or &BEGWRITE statement notifies the operator 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.

VARS *variable*
Pauses until the operator enters the GO command with or without the correct number of operands, or the RESET command. A previous &WRITE or &BEGWRITE statement notifies the operator 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.

When the command list interprets an &PAUSE control statement, the letter `P` is displayed in the upper right corner of the panel to alert 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 panel.

Using NetView Commands with &PAUSE

The operator can enter the NetView commands GO, RESET, STACK, and UNSTACK during a pause.

STACK and UNSTACK are used to suspend and then resume command list processing during an &PAUSE. After the STACK is issued, the operator can enter any network command.

Note: While an &PAUSE is suspended with the STACK command, the P is removed from the upper right corner of the panel. The P is displayed again after UNSTACK is issued. After UNSTACK is issued, the operator enters GO, 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 that 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 quotation marks.

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.

Precede the pauses for operator input with messages that supply the information to enter. Use the &WRITE or &BEGWRITE statements to send this information.

Note: The operator can call your command list from any NetView component. If you expect the command list to run from components other than the command facility, use NCCF in the command lists to present the operator with the command facility panel and command panel input area. (Do this before issuing any messages.) If the command list is running in the command facility, the NCCF command has no effect. Refer to the online help for more information about NetView commands.

An Example Using &PAUSE

The following example contains a portion of a command list that illustrates how to request 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
```

The example writes a message to the operator prompting for the last name, first name, and middle initial of the operator. 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 the following command:

```
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.

NetView 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 129 for information about the &IF control statement.) In an assignment statement, the value of the user variable is set to the result of the operation of the built-in function. Two of the NetView built-in functions, &HIER and &MSUSEG, have REXX-format functions, HIER() and MSUSEG(), for use in NetView REXX-only command lists.

Do not confuse built-in functions with variables of the same name. (All NetView command list language variables are described in Chapter 4, "REXX Instructions for NetView REXX Command Lists and Data REXX Files," on page 39.) Although they look similar, they are not the same. Except for the functions HIER() and MSUSEG(), both built-in functions and NetView command list language variables start with an ampersand (&). The following list describes the difference:

- A variable is replaced by its value when the command list runs. The variable is just a placeholder for the value.
- A built-in function is never replaced by a value. A built-in function is an action indicator rather than a placeholder.

These are the built-in functions that you can use:

- &BITAND
- &BITOR
- &BITXOR
- &CONCAT
- &HIER
- &LENGTH
- &MSUSEG
- &NCCFID
- &NCCFSTAT
- &SUBSTR

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 129.

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.

&BITAND

The &BITAND function returns a string composed of the two input strings logically ANDed together, bit by bit. The length of the result is the length of the longer of the two strings. If the AND operation ends when the shorter of the two strings is exhausted, the unprocessed portion of the longer string is appended to the partial result. If the value of both strings is null, the result is a null string.

The &BITAND function has the following syntax:

&BITAND

where:

string1
Can be either a constant or a command list variable.

string2
Can be either a constant or a command list variable.

The following two examples show the &BITAND operation:

```
&BITAND X'73' X'27' results in X'23'

&BITAND X'13' X'5555' results in X'1155'
```

Usage Notes:

1. If *string2* is null, the result is *string1* unchanged.
2. If you specify more than two strings, message DSI186I is issued and the command list ends. This is consistent with the equivalent REXX function.
3. If you do not specify *string1*, message DSI187I is issued and the command list ends. This is consistent with the equivalent REXX function.

&BITOR

The &BITOR function returns a string composed of the two input strings logically ORed together, bit by bit. The length of the result is the length of the longer of the two strings. If the OR operation ends when the shorter of the two strings is exhausted, the unprocessed portion of the longer string is appended to the partial result. If the value of both strings is null, the result is a null string.

The &BITOR function has the following syntax:

&BITOR

where:

string1
Can be either a constant or a command list variable.

string2
Can be either a constant or a command list variable.

The following two examples show the &BITOR operation:

```
&BITOR X'15' X'24' results in X'35'

&BITOR X'15' X'2456' results in X'3556'
```

Usage Notes:

1. If *string2* is null, the result is *string1* unchanged.

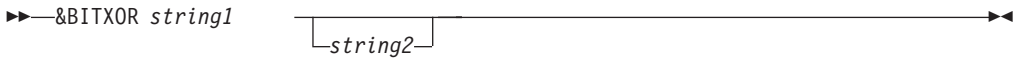
2. If you specify more than one string, message DSI186I is issued and the command list ends. This is consistent with the equivalent REXX function.
3. If you do not specify *string1*, message DSI187I is issued and the command list ends. This is consistent with the equivalent REXX function.

&BITXOR

The &BITXOR function returns a string composed of the two input strings logically exclusive ORed together, bit by bit. The length of the result is the length of the longer of the two strings. If the XOR operation ends when the shorter of the two strings is exhausted, the unprocessed portion of the longer string is appended to the partial result. If the value of both strings is null, the result is a null string.

The &BITXOR function has the following syntax:

&BITXOR



where:

string1
Can be either a constant or a command list variable.

string2
Can be either a constant or a command list variable.

The following two examples show the &BITXOR operation:

```
&BITXOR X'12' X'22' results in X'30'

&BITXOR X'1211' X'22' results in X'3011'
```

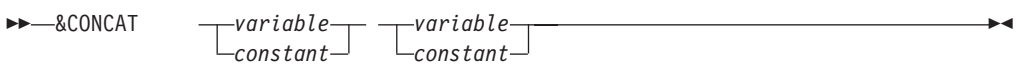
Usage Notes:

1. If *string2* is null, the result is *string1* unchanged.
2. If you specify more than one string, message DSI186I is issued and the command list ends. This is consistent with the equivalent REXX function.
3. If you do not specify *string1*, message DSI187I is issued and the command list ends. This is consistent with the equivalent REXX function.

&CONCAT

The &CONCAT function concatenates the values of two variables, two constants, or a variable and a constant to form a new value. The &CONCAT built-in function has the following syntax:

&CONCAT



Ensure that when the two items are joined, the resulting value does not exceed the maximum of 255 characters; higher values are truncated. If the value of both items being joined is null, the result is null.

For example, suppose you had the following statement:

```
&PREFIX = SN/
&ID     = 5497
&SERIAL = &CONCAT &PREFIX &ID
```

After processing, the user variables are set in the following way:
&PREFIX SN/
&ID 5497
&SERIAL SN/5497

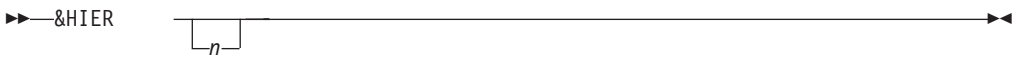
Note: When &CONCAT is used to concatenate two double-byte character set (DBCS) strings, it removes adjacent shift-in (SI) and shift-out (SO) characters.

&HIER

The &HIER function provides user access to the NetView hardware monitor hierarchy data associated with an MSU.

&HIER has the following syntax:

HIER



where:
n Specifies the index number (1 - 5) of a specific name/type pair.

Notes:

1. &HIER without *n* returns a resource hierarchy slightly different from that found in BNJ146I messages. The name/type pairs look like:

   aaaaaaaa1111bbbbbbbb2222....eeeeeeee5555

   The letters represent the resource name and numbers represent the resource type.

   The hardware monitor defines from one to five name/type pairs. Each name is eight characters long and each type is four characters. The names and types are padded with blanks if necessary.
2. &HIER with *n* returns the name/type pair, such as *aaaaaaaa1111* that corresponds to *n*. If no name/type pair corresponds to *n*, then a null value is returned.
3. &HIER returns null under the following conditions:
   - If the command list is not run by the automation table
   - If the automation table was not driven by an MSU
   - If the MSU does not have a hardware monitor resource hierarchy
4. You can test whether a resource is present in a resource hierarchy by using the example NetView command list language parsing template shown in Figure 40 on page 121.
5. If a complex link exists in a resource hierarchy, resource levels that are not displayed in the information returned by the &HIER function might exist. You must use a system schematic to determine the complete hierarchy configuration when a complex link is present.

```
*
* Set up variables for search
*
&RESNAME = AAAA
&RESLN   = &LENGTH &RESNAME
&SOURCE = &HIER
&SOURCLN = &LENGTH &SOURCE
*
* Check for existence of Hierarchy
*
&IF &SOURCLN = 0 &THEN -
  &GOTO -NOTFOUND
*
* Parse out desired resource name with PARSEL2R
*
  PARSEL2R SOURCE FIRSTSEG /&RESNAME/ LASTSEG
*
* If the last segment is non null, we found the resource name
*   imbedded in the hierarchy.
*
&IF &LASTSEG = '' &THEN -
  &GOTO -CKLAST
&GOTO -FOUNDMSG
*
* Check last segment of the hierarchy for desired resource name.
*   (If the desired resource name is the last entry in the hierarchy,
*   PARSEL2R will not detect it.  We need to make a special check for
*   the last entry.)
*
-CKLAST
*
* Trim any trailing blanks
*
-TRIMBLANK
&LASTCHAR = &SUBSTR &SOURCE &SOURCLN 1
&IF &LASTCHAR ¬= ' ' &THEN -
  &GOTO -OUTTRIM
&SOURCLN = &SOURCLN - 1
&IF &SOURCLN > 0 &THEN -
  &GOTO -TRIMBLANK
-OUTTRIM
*
```

Figure 40. Example of a &HIER Parsing Template (Part 1 of 2)

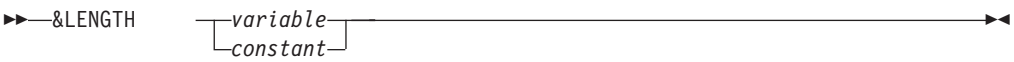
```
&IF &SOURCLN < &RESLN &THEN -
  &GOTO -NOTFOUND
&INDEX  =  &SOURCLN - &RESLN
&INDEX =   &INDEX + 1
&LASTENT = &SUBSTR &SOURCE &INDEX &RESLN
&IF &LASTENT = &RESNAME &THEN -
  &GOTO -FOUNDMSG
&GOTO -NOTFOUND
*
* Issue found message
*
-FOUNDMSG
&WRITE THE RESOURCE &RESNAME EXISTS IN THE HIERARCHY
&GOTO -LAST
*
* Issue not found message
*
-NOTFOUND
&WRITE THE RESOURCE &RESNAME DOES NOT EXIST IN THE HIERARCHY
*
* Exit
*
-LAST
&EXIT
```

Figure 40. Example of a &HIER Parsing Template (Part 2 of 2)

&LENGTH

The &LENGTH function returns the length of a variable or a constant. &LENGTH has the following syntax:

&LENGTH



The length of the variable value or constant is returned. If the variable is null or the constant is a null string, the value returned is 0.

The following example shows how to use &LENGTH. Suppose you called command list SAMP by entering SAMP LU2525. Assume the name of the hardcopy printer (&HCOPY) control variable is HC55.

```
SAMP CLIST
&HCLENGTH = &LENGTH &HCOPY
&RESLEN = &LENGTH &1
```

After processing, the variables are set in the following way:
&HCOPY HC55
&HCLENGTH
4
&1 LU25257
&RESLEN 6

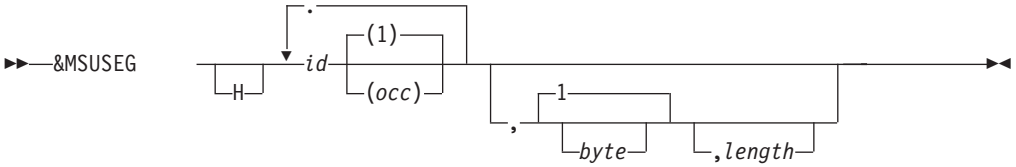
User variable &HCLENGTH is set to the length of the hardcopy device name. The hardcopy device is HC55. HC55 has four characters, so &HCLENGTH becomes 4. &RESLEN becomes the length of the first parameter variable. The first parameter variable is LU2525, so &RESLEN becomes 6.

&MSUSEG

The &MSUSEG function provides the parsing capability needed to extract information from a management services unit (MSU) or other similarly designed pieces of data. Use this function in a command list that is called by the NetView automation table or an LU6.2 application.

The &MSUSEG function has the following syntax:

&MSUSEG



where:

byte
The byte position into the lowest ID specified in *id*, counting from 1. Position 1 is the first length byte in the header of the lowest ID. The header is composed of one or two length bytes followed by the 1- or 2-byte ID. This entry is optional. The default is 1.

H Is inserted if the first ID is to be obtained from the next higher level multiple-domain support message unit (MDS-MU) as opposed to the NMVT/control point management services unit (CP-MSU) level. You can code the H in uppercase or lowercase. You can place H inside or outside of the single quotation marks when quotation marks are coded.

id Is the 2- or 4-character representation of 1- or 2-byte hexadecimal ID of GDS, major vector (MV), subvector, subfield, or sub-subfield. The hexadecimal characters (0 - 9, A - F, a - f) can be mixed case. The first ID is required; additional IDs are optional.

length
Is the number of bytes in decimal to be returned from the lowest ID specified in *id* and starting at the *byte* position. This entry is optional. The default is equal to the remainder of the lowest *id* specified, and starting at the *byte* position.

occ
The occurrence number, counting from one (1). You can use an asterisk (*) to specify the first occurrence found. This entry is optional at every level. The default is 1.

. The period establishes a hierarchy of IDs. Thus, the vector ID specified on the right side of the period is contained within the vector that is specified on the left side.

You can use blanks as delimiters between operands, but blanks do not act as place holders. For example, if you code a variable for the *byte* and the value of the variable is null and you used a blank as a delimiter, the *length* is considered to be the *byte* operand.

If the location is not found, or if the command list containing the &MSUSEG function was not processed by an automation table statement because of an MSU, or if the function was not driven by an MSU, then the value of the &MSUSEG function is null.

If you do not specify a *byte* position, the data returned includes the 1- or 2-byte length followed by the 1-or 2-byte ID of the lowest ID specified in *id*.

If the *byte* position is beyond the end of the location, a null value is returned.

If the specified length is longer than what remains at the location specified, whatever remains at the location is returned.

For more information about the automation table, refer to the *IBM Tivoli NetView for z/OS Automation Guide*. For more information about vector definitions, refer to the SNA library. For more LU6.2 and MSU information, refer to the *IBM Tivoli NetView for z/OS Application Programmer's Guide*.

&NCCFID

The &NCCFID function returns the NetView domain identifier of a domain with which you can establish a cross-domain session. The domains with which you can establish cross-domain sessions are defined by the DOMAINS statement of your operator profile. However, if your profile specifies AUTH CTL=GLOBAL, you can establish cross-domain sessions with the domains specified by the RRD statements in the CNMSTUSR or C*xx*STGEN that is included in the CNMSTYLE member. If you do not specify DOMAINS or CTL=GLOBAL in the operator profile, you receive an error message when using this function.

For more information about the domains and RRD statements, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

NCCFID has the following syntax:

&NCCFID

where:

number
Is either a number or a variable that becomes a number. 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.

The command list can use &NCCFID to automatically start or stop a cross-domain session.

The following example shows how to use &NCCFID:

```
&DOM1 = &NCCFID 1
&DOM2 = &NCCFID 2
&DOM3 = &NCCFID 3
START DOMAIN=&DOM1
START DOMAIN=&DOM2
START DOMAIN=&DOM3
```

Assume that your operator profile defines three domains with which you can establish cross-domain sessions:
1 ALPHA
2 BETA
3 GAMMA

After processing, the user variables are set in the following way:
&DOM1 ALPHA
&DOM2 BETA
&DOM3 GAMMA

The three domains are then started with the START command.

In this example, the operator must know that three domains can be started. You can also use the &IF control statement to test &NCCFCNT to find the number of domains and start them.

&NCCFSTAT

The &NCCFSTAT function returns a value indicating whether you have an active cross-domain session with the specified domain. &NCCFSTAT has the following syntax:

&NCCFSTAT

where:

domain
Is either a domain name or a variable that becomes a domain name.

The function call is replaced by the characters ACT if the operator has an active cross-domain session with the domain. The function call 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 the following example 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
```

After processing, the variables are set in the following way:
&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 129).

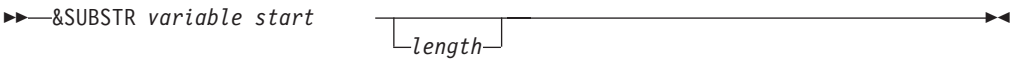
If NCCFA is inactive, the command list starts it. If NCCFA is active, you receive the following message:

```
DOMAIN NCCFA IS ALREADY ACTIVE
```

&SUBSTR

The &SUBSTR function returns the specified portion of an input variable by parsing the variable, starting at position *start* for *length* characters. &SUBSTR has the following syntax:

&SUBSTR



where:

length
The number of characters to parse, beginning with the specified *start* position. If no *length* is specified, the parsing is from the *start* to the end of the variable.

start
Is the starting position of the parsing operation within the *variable*.

variable
Is the variable to be parsed.

For example, suppose you have the following statements:

```
&HOLD = ACF/VTAM
&FIRST = &SUBSTR &HOLD 1 3
&SECOND = &SUBSTR &HOLD 5 4
&THIRD = &SUBSTR &HOLD 6
```

After processing, the user variables are set in the following way:

&HOLD	ACF/VTAM
&FIRST	ACF
&SECOND	VTAM
&THIRD	TAM

The first line of the previous example 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 next 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. In the last line, &SUBSTR starts at the sixth character of &HOLD (the character T) but does not specify a length. &THIRD is therefore TAM, the value of &HOLD from the letter T through the end of the variable (M). 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 not valid 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. Substrings are never padded with blanks. If you specify a length that is too long, no length is assumed and the entire string beginning at the starting position is used. If the length is 0, or the starting position is beyond the variable length, the result of &SUBSTR is null.

Figure 41 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 41. Using &APPLID to Determine the Domain Name

When using double-byte characters along with Roman characters (A-Z, a-z), the &SUBSTR function adjusts the variable in the following way:

Start byte = shift-out character
No adjustment

Start byte = shift-in character
Replace with blank

Start byte = first half of double-byte
Replace with blank and shift-out character

Start byte = second half of double-byte
Replace with shift-out character

Last byte = shift-out character
Replace with blank

Last byte = shift-in character
No adjustment

Last byte = first half of double-byte
Replace with shift-in character

Last byte = second half of double-byte
Replace with shift-in character and blank.

The following example shows the &SUBSTR statement used on a double-byte character and Latin character string:

```
&DBCS = 'AB<D1D2D3>EFG'
```

where:
- A, B, E, F, G are Latin characters.
- < (X'0E') represents the shift-out control character.
- > (X'0F') is the shift-in control character.
- D1, D2, D3 are double-byte characters.

Using this value, &SUBSTR works in the following way:

```
&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.

Appendix B. NetView Command List Language Branching

This chapter describes the conditional and unconditional branching statements in the NetView command list language.

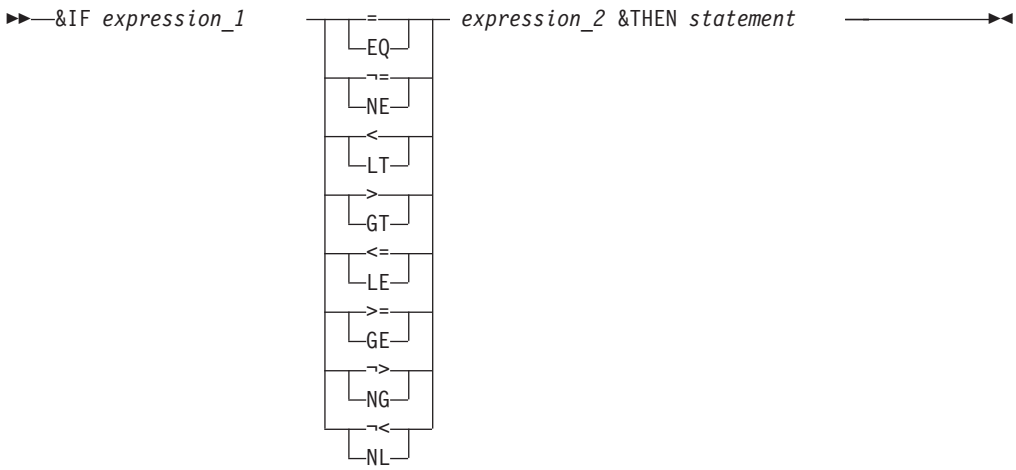
- The &IF statement causes a conditional branch based on logical or arithmetical comparisons. The result of a test or comparison in an &IF statement determines the alternative to perform. Conditional processing statements give you the flexibility to code if-then and loop structures.
- The &GOTO statement causes unconditional branching.
- The &EXIT statement lets you code logical exit points within a command list.
- The &WAIT statement suspends processing and waits for the completion of an event.

&IF Control Statement

The &IF control statement tests a condition and performs processing based on the results of the test. The condition consists of two expressions and a logical or arithmetical operator.

If the condition is true, the &THEN clause is processed. If the condition is false, processing continues at the statement following the &IF control statement. The &IF control statement has the following syntax:

&IF&THEN



where:

= or EQ
Equal

expression_1
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, see "Assignment Statements" on page 108.

expression_2
Is the second term of comparison. It follows the same rules as *expression_1*.

> or **GT**
Greater than

>= or **GE**
Greater than or equal

< or **LT**
Less than

<= or **LE**
Less than or equal

¬= or **NE**
Not 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 have the same meaning.

&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: Coding the ampersand (&) with THEN identifies the word as part of the control statement.

statement
Is the command list statement that is processed if the comparison is true, otherwise it is ignored. The statement can be any NetView command list language statement.

Variables coded in the comparison expressions are replaced by their values before the comparison is checked. You can use two single quotation marks with no space ('') to test whether a variable is null. For example, the comparison `&1 = ''` is true when &1 is null.

The following example shows comparisons:

```
5 = &A
&1 = ''
2 + 2 NE &ANSWER
&PARMCNT LE 5
```

If a variable used in an arithmetic expression can be equal to null, then use the following syntax:

```
7 > 3 + 0&1
```

In this example, the zero (0)&1 is evaluated as zero (0) because &1 is null. Therefore, the expression 3 + 0 is compared to 7. If &1 is equal to 9, the expression 3 + 09 is compared to 7.

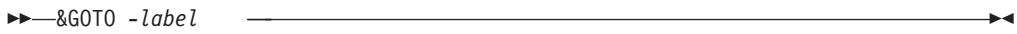
The following five examples use the &IF control statement:

```
&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 = '01/01' &THEN &WRITE         HAPPY HOLIDAY

&IF &A = X'41' &THEN &GOTO -PROC1
```

&GOTO Control Statement

The &GOTO statement 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. The &GOTO control statement has the following syntax:

&GOTO



where:

-label
: Identifies the target statement in this command list where processing continues.

When the NetView program interprets the &GOTO statement, it searches the command list for a statement starting with this same label. The NetView program 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 the NetView program processes the &GOTO statement. See “Labels” on page 100 for more 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 an error occurs. You can use &EXIT with &GOTO to control the flow of the command list. The &EXIT control statement has the following syntax:

&EXIT

where:

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 act based on the return code. See "Command List Information" on page 44 for more information about &RETCODE.

You can define 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 panel.

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 no &EXIT statement exists, a zero return code is set.

Figure 42 shows 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, 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 42. Example of a CLIST to Stop TAF Sessions

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 want a command list to wait for a specific event or message. With the &WAIT control statement, you define what event causes the command list to resume processing. The command list can wait for any message with a 1- to 10-character message identifier.

Notes:

1. You cannot use &WAIT when operating under the primary POI task (PPT), or when using common operations services (COS) commands. See "Primary POI Task Restrictions" on page 12 for more information using &WAIT under the PPT. For additional information about using &WAIT with common operations services commands, see Appendix D, "Common Operations Services Commands," on page 155.
2. NetView pipelines, called with the PIPE command, provide both extended function and reduced complexity for the automation of message handling. The PIPE command is an alternative to the &WAIT control statement. For information about NetView pipelines, refer to the *IBM Tivoli NetView for z/OS Programming: Pipes* book.

If you use &WAIT in an automation task command list, be sure to specify a reasonable timeout value. For instructions about coding a time-out event, see "The Event=-Label Pair" on page 135.

If the trapped message satisfies the wait condition, processing of the waiting command procedure resumes. If you do not suppress the message, it continues with the message flow. If you suppress the message, however, the NetView program marks it for deletion. In this case, automation-table processing does not occur and the NetView program does not display or log the message.

&WAIT performs the following actions in a command list:

- It causes the NetView program 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, &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. For example, for your installation, you might 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 the NetView program to monitor the OST 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 used to resume the command list only if the event never occurs. When a command list is in a wait

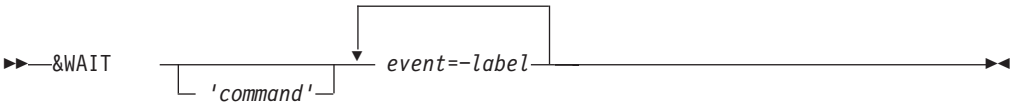
state, the NetView program ignores operands on the GO command. RESET, STACK, and UNSTACK work the same way for &WAIT and &PAUSE.

Coding an &WAIT Control Statement

You can code an &WAIT statement in several ways. This section describes the basic format. "Customizing the &WAIT Statement" on page 141 describes ways to customize &WAIT.

When the command list begins processing a &WAIT control statement, NetView displays the letter W in the upper right corner of the panel if the panel is refreshed because a message is received or the ENTER key is pressed. This W notifies the operator that a command list process is in a wait state. Wait state means that 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. The &WAIT control statement has the following syntax:

&WAIT



where:

'command'
Is any command or command list that you can issue from the NetView program. 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 quotation marks. 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.

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 the NetView program 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 about the types of events that can satisfy an &WAIT, see "The Event=-Label Pair" on page 135.

When the NetView program 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 141.

The only messages checked are those that are intended for the operator screen. If you code the DSIEX02A exit routine (output to the operator), the &WAIT control statement might not set the message for matching. For example, if the DSIEX02A exit routine deletes the message, the &WAIT control statement does not get the message so a match is not made. Because the operator does not receive the message, neither does the waiting command list. Therefore, wait only for messages that are displayed on the NetView console.

When coding the &WAIT command control statement, it is important to code an *event=-label* pair for the DSI210I message and the *NN event. The DSI210I message is returned when the command found in the command list is not authorized for this operator, and the *NN event prevents waiting indefinitely for operator intervention. The statements following labels need to notify the operator of the error and exit the command list.

The W that signifies wait state, if present, remains in the upper right corner of the panel while this initial &WAIT command is processed. The W indicates 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 the DSI016I NOT IN PAUSE OR WAIT STATUS message. 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 is no longer displayed in the upper right corner of the panel.

You can code several *event=-label* pairs, but 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 141.

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 100. The label coded on the &WAIT statement can be a variable, but do not use parameter variables.

You can pass control to the label on an &WAIT statement by specifying an *event=-label* pair. The following events can be used:

- *token*
- *ERROR
- **nn*
- *ENDWAIT

The following list describes the previously mentioned events:

token This event occurs when the NetView program receives a message matching *token*. The *token* variable can be 1 - 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.

The following examples show some of the ways you can specify the messages for which you want the command list to wait:

domainid.token
: The event occurs when the NetView program receives any message whose domain identifier matches the 1–5 character *domainid* and whose first token matches *token*.

dom.token*
: The event occurs when the NetView program 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 the NetView program receives any message whose first token matches *token*. The message can be from any domain.

token
: The event occurs when the NetView program receives any message whose first token matches *token*. The message can be from any domain.

*tok**
: The event occurs when the NetView program receives any message whose first token matches the partial token specified by *tok**. For example, DSI* means that the event occurs when the NetView program receives any message whose first token begins with DSI (such as DSI463I or DSI386I).

*
: The event occurs when the NetView program receives any message or other output. For example, if you code &CONTROL ALL in the command list, every line of the command list is echoed on the panel. These echoes satisfy the * condition, and depending on the code in the command list, can cause a loop or other unwanted results. Therefore, use the *=-label condition with caution.

If you specify a token that contains a special character such as a comma, period, asterisk, or most other non-numeric and non-alphabetic characters, use the DOMAIN.TOKEN format. The NetView program does not accept single quotation marks ('), commas (,), or blanks when you specify a token because these characters are reserved as NetView default delimiters. If the token contains the ampersand (&) then &CONCAT must be used to concatenate the ampersand with the rest of the token.

Figure 43 on page 136 shows examples of coding tokens that contain special characters:

```
&WAIT DOMAIN1.*HASP=-MSG1
&WAIT DOMAIN1.=HASP=-MSG1
&X = &CONCAT & HASP
&WAIT DOMAIN1.&X=-MSG1
```

Figure 43. Examples of Coding Tokens with Special Characters

Multiline messages such as multiline write-to-operators (MLWTOs) are treated as one message. Therefore, only the message identifier of the first message in a multiline message is available to the &WAIT, and the &WAIT statement can be satisfied only by that message identifier. Use GETMSIZE, GETMTYPE, GETMLINE, GETMPRES, and GETMTFLG to access the other lines of a multiline message. Refer to these commands in the NetView

online help for more information about multiline messages and an example of using &WAIT with multiline messages.

Notes:

1. When using a *token* event, messages not related to the command issued by the &WAIT statement can be matched to the event and, depending on the options on the &WAIT statement, can be suppressed. However, use caution when coding * or *.* with SUPPRESS when specifying a domain identifier or token. If the command list is suspended and the SUPPRESS option is in effect on the &WAIT statement, any messages the task receives are suppressed before the command list is resumed.
2. Because NetView-NetView tasks (NNTs), PPT, OSTs, and autotasks do not process any commands or messages queued to the low priority queue of a task that is running any command (assembler command or HLL, REXX or NetView command list language command procedure), only messages that are queued to the high or normal priority queue of a waiting task are checked for matches to satisfy a wait condition.
3. Usually, messages queued to tasks through assign...copy= processing can satisfy an outstanding &WAIT. However, a message is not sent to the waiting task through assign...copy= processing if the message contains a message automation table entry specifying DISPLAY(N). The assign...copy= processing requires a displayed message, but DISPLAY(N) specifies no display, which prevents that processing. The message is not passed on; therefore, it cannot satisfy the &WAIT condition.

   For more information about message flow, refer to the *IBM Tivoli NetView for z/OS Automation Guide* book.

***ERROR**

This event occurs when the command specified on the &WAIT statement returns a nonzero return code. If you do not code *ERROR, the NetView program continues to wait for the messages associated with this command even if the command ends with an error. If the NetView program is waiting for a message that says the command was successful, the operators running this command list are delayed until someone issues GO or RESET. If *ERROR is satisfied, the message control variables are set in the following way:

&MSGID	*ERROR
&MSGORIGIN	Name of domain where the command list is running
&MSGSTR	Null
&MSGCNT	0

Note: Messages associated with the command can be received before the command returns a nonzero 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 example, if you code the name of the &WAIT command on a MSGID=-*label* pair, and you also code an *ERROR=-*label* pair, the NetView program 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 1 - 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.

***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, the NetView program must 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. The NetView program scans the pairs in the order you code them, from left to right.

For example, assume that you code the following statement:

```
&WAIT IST*=-ALL,IST123I=-SPECIAL
```

When the NetView program receives IST123I, it goes to the label -ALL, not -SPECIAL. 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. The NetView program scans the list from left to right until a match is found.

Ending an &WAIT

An &WAIT statement can end in one of the following ways:

- The operator enters the GO command. Processing continues with the next statement, unless *ENDWAIT is specified on the &WAIT statement. If *ENDWAIT is specified, processing continues with the statement marked by the label.
- The operator enters the RESET command. The command list and all of its nested command lists end.
- 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.
- 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.
- 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.

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

GO Ends the wait.

If *ENDWAIT is coded, processing continues with the labeled statement.

STACK
Suspends command list processing and causes any commands that are 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 command list using &WAIT does not process messages as they occur, but, instead, after the command list is given control again. The W does not remain in the upper right corner of the NetView panel. 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.

RESET
Ends a command list that is in a wait state, and all command lists related to it by nesting.

Note: 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, and the GO, STACK, UNSTACK, and RESET commands, refer to the NetView online help.

Control and Parameter Variables Used with &WAIT

The NetView program sets the values of the control variables. The following variables are based on the receipt of a message coded on an &WAIT control statement:

&ACTIONDL
&ACTIONMG
&AREAID
&ATTNID (VSE only)
&AUTOTOKE
&DESC
&HDRMTYPE
&IFRAUGMT
&IFRAUIND
&IFRAUIN3
&IFRAUI3X
&IFRAUSB2
&IFRAUSC2
&IFRAUSDR
&IFRAUSRB
&IFRAUSRC
&IFRAUTA1
&IFRAUWF1
&JOBNAME
&JOBNUM
&KEY
&LINETYPE
&MCSFLAG
&MSGASID
&MSGAUTH
&MSGCATTR
&MSGCMISC
&MSGCMLVL
&MSGCMSGT
&MSGCNT
&MSGCOJBN
&MSGCPROD
&MSGCSPLX
&MSGCSYID
&MSGDOMFL
&MSGGBGPA
&MSGGDATE
&MSGGFGPA
&MSGGMFLG
&MSGGMID
&MSGGSEQ
&MSGGSYID
&MSGGTIME
&MSGID
&MSGORIGN
&MSGSRCNM
&MSGSTR
&MSGTOKEN
&MSGTSTMP
&MSGTYP
&MVSRTAIN()
&NVDELID
&PARTID (VSE only)
&PRTY
&REPLYID
&ROUTCDE
&SESSID
&SMSGID
&SYSCONID
&SYSID
&1–&31

The NetView program 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.

For more information, see "Message Processing Information Functions" on page 52.

The following example shows how the variables are set when the message `DSI008I SPAN1 NOT ACTIVE` from domain DOM01 is intercepted by an &WAIT statement:

&MSGORIGIN	DOM01
&MSGID	DSI008I
&MSGSTR	SPAN1 NOT ACTIVE
&MSGCNT	3
&1	SPAN1
&2	NOT
&3	ACTIVE
&4–&31	NULL

Notes:

1. If the NetView program receives a multiline message, the control variables and parameter variables are set according to the first nonblank line of the message. Refer to the GETM commands in the NetView online help for information about multiline messages.
2. If &1–&31 are given values when the command list runs, save the parameter variables in user variables before calling the &WAIT control statement. This procedure 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 procedure 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 &MSGID is the first character string on an &WRITE or &BEGWRITE, the output might 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 screen. If an &WAIT CONTINUE statement is encountered after a MSGID=-*label* is matched, and no other statement ends the command list or the &WAIT, the command list loops.

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. Note 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 the nested command list ends before the message satisfies the &WAIT, the message is queued for the waiting command list. Without the ending of the &WAIT or the waiting command list, the message queue continues to grow and the NetView program can run out of storage.

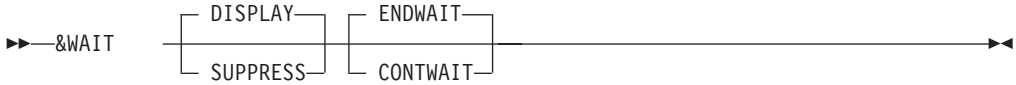
Customizing the &WAIT Statement

The previous sections described the simplest form of the &WAIT command, in which 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 more flexibility.

To customize your &WAIT statements, use the following syntax.

&WAIT



or

&WAIT

where:

DISPLAY
Indicates that the message the command list is waiting for is to be displayed at the operator's terminal upon arrival to the NetView program. DISPLAY is the default value.

SUPPRESS
Indicates that any messages that satisfy a &WAIT statement are not displayed, logged, or automated.

CONTWAIT
Indicates that the next &WAIT *event=-label* statement encountered waits for additional events until the wait is ended. CONTWAIT enables one &WAIT statement to process more than one event. This operand is useful when you want to retrieve more than one message from a single command, such as a LIST command.

ENDWAIT
Sets up processing for the next *event=-label* pair to be processed. ENDWAIT is the default value, and indicates that the wait ends after the first event that satisfies the &WAIT. Although the ENDWAIT specification 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.

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. It is similar to branching back into the &WAIT statement.

Notes:

1. If neither DISPLAY nor SUPPRESS is specified, then either ENDWAIT or CONTWAIT must be specified.

2. DISPLAY is the default *only if* ENDWAIT or CONTWAIT is specified and SUPPRESS is *not* specified. See Table 18 for valid option combinations.
3. The DISPLAY and SUPPRESS options can be changed at any point in a command list. After messages are suppressed, you must code another &WAIT statement with the DISPLAY operand to begin displaying messages again.
4. &WAIT SUPPRESS overrides DISPLAY because the command list has been given the message and does not issue an echo.
5. When SUPPRESS is in effect, you do not know whether messages are received. Therefore, all of the messages might not be processed when an operator issues a GO or RESET command to end an &WAIT.
6. If neither ENDWAIT nor CONTWAIT is specified, then either DISPLAY or SUPPRESS must be specified.
7. ENDWAIT is the default *only if* DISPLAY or SUPPRESS is specified and CONTWAIT is *not* specified. See Table 18 on page 142 for valid option combinations.
8. The ENDWAIT and CONTWAIT options can be changed at any point in a command list. After CONTWAIT starts, you must code another &WAIT statement with the ENDWAIT operand to return to the default value.

Table 18. &WAIT Customization Options Matrix.
S = Specified operand
• = Default called

DISPLAY	SUPPRESS	ENDWAIT	CONTWAIT
S		S	
S			S
	S	S	
	S		S
•		S	
•			S
S		•	
	S	•	
Note: At least one option must be specified. Defaults are not called if no option is specified.			

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 138 describes 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 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 an error occurs in the command list and you do not have *ERROR coded, the wait might not end without intervention.
- 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.

  The command list ends.
- By entering the RESET command.

  The command list, including the command list that initiated it, ends.

Because &WAIT CONTWAIT queues NetView messages, also code &WAIT CONTINUE to receive these queued messages. If you code &WAIT CONTWAIT with SUPPRESS and end the wait, you might lose some messages.

Suggestions for Coding &WAIT

For the best performance, 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 you specify &WAIT ENDWAIT before the &WAIT *event=-label* statement, or if &WAIT ENDWAIT is in effect by default, the first matched event ends the wait, and command list processing continues. See "Ending an &WAIT" on page 138.
   - If you specify &WAIT CONTWAIT, the receipt of the first event does not end the &WAIT unless this event is specified as shown in "Ending &WAIT If CONTWAIT Is in Effect" on page 143. The command list goes to the label specified for the event and continues processing.

To complete this section of the command list, take one of the following actions:

- Continue the wait by coding &WAIT CONTINUE.
- Specify that the next event is the last event 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.

Sample Using &WAIT

Figure 44 shows 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, reformats the messages, and writes them to the operator screen.

```
&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
-REFORM
&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 44. 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
```

Activate the command list 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) causes 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 message text shown in Figure 45 is displayed on the operator terminal:

```
ACTONE NODE1
IST097I  VARY      ACCEPTED
VTAM MESSAGE IST093I WAS RECEIVED
NODE1 IS NOW ACTIVE
COMMAND LIST ACTONE COMPLETE
```

Figure 45. ACTONE NODE1 Message Text

Appendix C. NetView Command List Language Global Variables

This chapter describes the use of global variables in the NetView command list language. Global variables enable values to be defined, referenced, and updated by different operators. Values are passed to a command list for updates, and the updated values can then be referenced by other command lists. 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.

The two types of global variables are task and common.

Task global variables can be defined, referenced, and updated by any command list running under a particular task. Task global variables can be referenced only by command lists running under the task in which the variable was defined.

Common global variables enable definition of user variables that can be referenced by command lists running under any NetView task that supports command list processing.

The NetView program provides two methods to access global variables:

- You can use &TGLOBAL, &CGLOBAL, and GLOBALV DEF to provide direct reference to global variables.
- You can use the GLOBALV GET and PUT commands to copy and replace global variable values.

Refer to the NetView online help for more information about the GLOBALV command.

Notes:

1. Use caution when mixing &TGLOBAL, &CGLOBAL, or GLOBALV DEF with the GLOBALV GET or PUT command. Using both methods to access global variables of the same name within a single NetView command list language command list is not recommended.

   A direct set affects how subsequent copying and replacing are performed. GLOBALV GETs and PUTs copy the value from one dictionary to the other. &TGLOBAL and &CGLOBAL and GLOBALV DEFT or DEFC let you reference the global variable and set it directly from that statement forward in the command list. While each command provides function, use only one or the other within a single NetView command list.
2. When you create global variables, the variable can be 1 - 11 characters in length. A - Z, 0 - 9, #, @, and $ are valid characters.
3. The value of the global variable can be 255 characters long. The maximum number of double-byte characters between the shift-out (X'0E') and shift-in (X'0F') control characters is 126.
4. You can give global variables a numeric value between -2147483647 and 2147483647. Numeric values outside these limits are treated as character strings.

Using &TGLOBAL and &CGLOBAL

You can specify more than one global variable using the &TGLOBAL and &CGLOBAL control statements. The variable names must be delimited by a comma or blank.

On the definition statement, do not code an & 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 & to the variable name, just as you do for user variables.

You need two &s when referencing a global variable indirectly. See "Using Parameter Variables in a Command List" on page 103 and "Variable Substitution Order" on page 101 for more information about indirect referencing of variables.

&TGLOBAL

A task global variable can be referenced only by command lists that run under the same task.

Use the &TGLOBAL control statement to define any variable as a task global variable. The syntax of the &TGLOBAL control statement follows:

&TGLOBAL

This statement defines the listed variables as task global variables. 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 assigned, the value is undefined or null, and any attempt to retrieve the value causes a null value to be returned. If you do not use the &TGLOBAL statement in each command list before a variable is referenced, that variable defaults to a local user variable.

The following example shows using the &TGLOBAL control statement:

```
&NAME = JOHN
&TGLOBAL ABC,&NAME
```

The first line consists of a local user variable set to the value JOHN. The second line defines two task global variables in the following way:

- ABC becomes task global variable &ABC. The value of &ABC is null because a value was not defined.
- The value of &JOHN is null because a value has not been defined. This is an example of indirect referencing of variables.

See "Extent of Variables When Using &TGLOBAL and &CGLOBAL" on page 151 for information about the interaction of task global variables with user variables and common global variables.

If you specify more than one variable name on the &TGLOBAL statement, the variable names must be delimited by commas or blanks.

Note the following 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 that 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 improves nested command list communication because task global variables can return information from a nested command list.

&CGLOBAL

Use the &CGLOBAL control statement to define any variable as a common global variable. The syntax of the &CGLOBAL control statement follows:

&CGLOBAL

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 was assigned, the value is undefined or null, and any attempt to retrieve the value causes a null value to be returned. If you do not use the &CGLOBAL statement in each command list before a variable is referenced, that variable defaults to a local user variable.

An example using the &CGLOBAL control statement follows:

```
&NAME = JOHN
&CGLOBAL ABC,&NAME
```

The first line consists of a local user variable set to the value JOHN. The second line defines two common global variables:

- ABC becomes common global variable &ABC. The value of &ABC is null because a value is not defined.
- &NAME becomes common global variable &JOHN. Because &NAME has a value of JOHN, the &NAME on this line gets substituted as JOHN. This defines the common global variable &JOHN. The value of &JOHN is null because a value is not defined.

If you have more than one command list running under different tasks accessing the same global variable, the last value that the variable is set to is the value that is set by any command list changing 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 prevent a common global variable from being updated by different command lists at the same time, you can have all command lists that update the variable run

under the same task. See "Extent of Variables When Using &TGLOBAL and &CGLOBAL" on page 151 for information about the interaction of common global variables with user variables and task global variables.

If you specify more than one variable name on the &CGLOBAL statement, the variable names must be delimited by commas or blanks.

You can use the command lists UPDCGLOB and SETCGLOB command lists supplied with the NetView product to update and set common global variables. Refer to the NetView online help for information.

You can use common global variables to maintain accurate information about the network regardless of operators logging on and off.

You can use common global variables to keep cumulative information from unsolicited access method messages. For example, you can use notification of a failing resource to recover the resource. With a global variable, you can maintain a count of the number of retries to prevent a loop.

Updating Task Global Variables Using &TGLOBAL

The following two examples show command lists. The first command list, CLIST1, calls the nested command list UPDT1. The CLIST1 and UPDT1 command lists show how to define, reference, and update a task global variable.

```
* THIS STATEMENT DEFINES SYSVAR1 AS A TASK GLOBAL VARIABLE.
  &TGLOBAL SYSVAR1
* THIS ASSIGNMENT STATEMENT GIVES THE TASK GLOBAL
*   VARIABLE, "SYSVAR1", A VALUE OF 5.
  &SYSVAR1 = 5
* THIS STATEMENT CALLS A NESTED COMMAND LIST NAMED UPDT1.
*   SYSVAR1 IS A PARAMETER THAT IS PASSED TO COMMAND LIST UPDT1.
  UPDT1 SYSVAR1
* THIS STATEMENT WILL WRITE VALUE OF SYSVAR1.
  &WRITE SYSVAR1 = &SYSVAR1
  &EXIT
```

Figure 46. CLIST1 Command List to Define, Update, and Reference Task Global Variables

CLIST1 in Figure 46 defines a task global variable, SYSVAR1. The value of the task global variable SYSVAR1 returns a null value until a value is assigned using the assignment statement, &SYSVAR1 = 5. CLIST1 activates a nested command list named UPDT1.

```
* THIS STATEMENT DEFINES &1 AS A TASK GLOBAL VARIABLE.
*   &1 IS SET TO THE VALUE OF THE POSITIONAL PARAMETER
*   SYSVAR1, 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 RETURNED WAS NULL.
*   THE TASK GLOBAL VARIABLE PASSED AS &1 IS REFERENCED
*   AS &&1.  THE VALUE OF &&1 IS EQUAL TO THE VALUE OF SYSVAR1,
*   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 &SYSVAR1.
  &&1 = &&1 + 1
  &EXIT
```

Figure 47. UPDT1 Command List to Update Task Global Variables

UPDT1 in Figure 47 on page 151 redefines the value stored in task global variable &1. Task global variable &1 gets its original value from SYSVAR1, which was the first (and only) variable passed to UPDT1 when it was called by CLIST1. Because the NetView program scans variables from right to left, the &1 part of &&1 is evaluated first, and the value of &1 is equal to the value of SYSVAR1. The value of the task global variable is referenced as &SYSVAR1. The initial value of &SYSVAR1 is 5, and then &SYSVAR1 is incremented by 1 using the &&1 = &&1 + 1 statement. (&SYSVAR1 = &SYSVAR1 + 1 after it is evaluated by the NetView program.)

The updated value is available as a task global variable &SYSVAR1 in CLIST1. The &WRITE SYSVAR1 = &SYSVAR1 statement displays the updated value of the &SYSVAR1 task global variable.

Extent of Variables When Using &TGLOBAL and &CGLOBAL

If you define a global variable 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 command list defines a common global variable after the task global variable is defined and has the same name as a task global variable, the value of the task global variable remains unchanged. However, this command list can no longer access the value of the task global variable unless you redefine the variable using &TGLOBAL.

If a command list defines a task global variable after the common global variable is defined with the same name as a common global variable, the value of the common global variable remains unchanged. However, this command list can no longer access the value of the common global variable unless you redefine the variable using &CGLOBAL.

GLOBVAR1, Figure 48 on page 152, illustrates the extent of user variables, task global variables, and common global variables within individual command lists and command lists running under different tasks. This command list gives 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.

The following example shows a command list containing global variables. In the example, the values of the variables are shown in parentheses.

```
&CONTROL ERR
*** CLIST NAME: GLOBVAR1
*** ASSIGN VALUES TO SEVERAL USER VARIABLES AND PRINT THEIR VALUES
*********************************************************************
    &OPER = OPER1
    &VTLV = VT33
    &DOM1 = CNM01002
    CLEAR
    &BEGWRITE SUB -ENDLOCAL
       FROM GLOBVAR1: AFTER LOCAL VARIABLES ASSIGNED
             VARIABLE       VARIABLE       VARIABLE
               TYPE           NAME           VALUE
             ========       ========       ========
               LOCAL          OPER           &OPER (OPER1)
               LOCAL          VTLV           &VTLV (VT33)
               LOCAL          DOM1           &DOM1 (CNM01002)
    -ENDLOCAL
*
*** DEFINE TASK GLOBAL VARIABLES
*******************************
    &TGLOBAL OPER VTLV CNT
    &BEGWRITE SUB -ENDTG1
       FROM GLOBVAR1: AFTER TGLOBAL VARIABLES DEFINED
             VARIABLE       VARIABLE       VARIABLE
               TYPE           NAME           VALUE
             ========       ========       ========
               LOCAL          DOM1           &DOM1 (CNM01002)
               TASK           OPER           &OPER (NULL)
               TASK           VTLV           &VTLV (NULL)
               TASK           CNT            &CNT  (NULL)

           NOTE THAT THE VALUES ASSIGNED TO OPER AND VTLV
           HAVE BEEN LOST AS THEY ARE NO LONGER LOCAL
           VARIABLES AND THE TASK GLOBAL VARIABLES HAVE NOT
           BEEN ASSIGNED YET.
    -ENDTG1
```

Figure 48. GLOBVAR1 Example Showing Extent of Global Variables (Part 1 of 3)

```
*
*** ASSIGN VALUES TO THE TASK GLOBAL VARIABLES
*********************************************
    &OPER = OPER2
    &VTLV = VT33
    &CNT  = 3
    &BEGWRITE SUB -ENDTG2
       FROM GLOBVAR1: AFTER VALUES ASSIGNED TO TGLOBAL VARIABLES
             VARIABLE       VARIABLE       VARIABLE
               TYPE           NAME           VALUE
             ========       ========       ========
               LOCAL          DOM1           &DOM1 (CNM01002)
               TASK           OPER           &OPER (OPER2)
               TASK           VTLV           &VTLV (VT33)
               TASK           CNT            &CNT  (3)

    -ENDTG2
*
*** DEFINE COMMON GLOBAL VARIABLES
*********************************
    &CGLOBAL OPER VTLV VAL
    &BEGWRITE SUB -ENDTG3
       FROM GLOBVAR1: AFTER CGLOBAL VARIABLES DEFINED
             VARIABLE       VARIABLE       VARIABLE
               TYPE           NAME           VALUE
             ========       ========       ========
               LOCAL          DOM1           &DOM1 (CNM01002)
               TASK           CNT            &CNT  (3)
               COMMON         OPER           &OPER (NULL)
               COMMON         VTLV           &VTLV (NULL)
               COMMON         VAL            &VAL  (NULL)

           NOTE THAT THE VALUES ASSIGNED TO TASK GLOBAL
           VARIABLES OPER AND VTLV HAVE BEEN REPLACED BY
           COMMON GLOBAL VARIABLES OPER AND VTLV.  THESE ARE
           NULL AS NO VALUE HAS BEEN ASSIGNED TO THEM YET.
    -ENDTG3
```

Figure 48. GLOBVAR1 Example Showing Extent of Global Variables (Part 2 of 3)

```
*
*** ASSIGN VALUES TO COMMON GLOBAL VARIABLES
*****************************************
    &OPER = OPER3
    &VTLV = VT32
    &VAL  = HEX
    &BEGWRITE SUB -ENDTG4
       FROM GLOBVAR1: AFTER CGLOBAL VARIABLES ASSIGNED
             VARIABLE       VARIABLE       VARIABLE
               TYPE           NAME           VALUE
             ========       ========       ========
               LOCAL          DOM1           &DOM1 (CNM01002)
               TASK           CNT            &CNT  (3)
               COMMON         OPER           &OPER (OPER3)
               COMMON         VTLV           &VTLV (VT32)
               COMMON         VAL            &VAL  (HEX)
    -ENDTG3
*** CHANGE ONE COMMON GLOBAL VARIABLE BACK TO A TASK GLOBAL VARIABLE
*********************************************************************
    &TGLOBAL OPER
    &BEGWRITE SUB -ENDTG5
       FROM GLOBVAR1: AFTER FINAL TGLOBAL STATEMENT
             VARIABLE       VARIABLE       VARIABLE
               TYPE           NAME           VALUE
             ========       ========       ========
               LOCAL          DOM1           &DOM1 (CNM01002)
               TASK           CNT            &CNT  (3)
               TASK           OPER           &OPER (OPER2)
               COMMON         VTLV           &VTLV (VT32)
               COMMON         VAL            &VAL  (HEX)
           NOTE THAT THE OPER NOW HAS THE VALUE OF THE TASK
           GLOBAL VARIABLE OPER AGAIN AS THE MOST RECENT
           DECLARATION STATEMENT DEFINED IT AS TASK GLOBAL.
    -ENDTG5
```

Figure 48. GLOBVAR1 Example Showing Extent of Global Variables (Part 3 of 3)

GLOBALV Command

The GLOBALV command is used to define, put, and get global variables in NetView command list language command lists. The GLOBALV command also saves global variables in a VSAM database. You can restore saved global variables if the NetView program is stopped and restarted, or erase (purge) saved global variables from external storage. Global variables enable multiple command procedures, regardless of their language, to share a common set of values.

Refer to the NetView online help for more information about the GLOBALV command.

Appendix D. Common Operations Services Commands

This chapter describes how to use the common operations services (COS) commands.

Common Operations Services

The common operations services is a set of commands that supports and enhances NetView control of common operations, for example, NetView/PC. A COS application manages nonsystem network architecture devices, such as front-end line switches and multiplexers. You can send commands to the COS application to do problem determination for these devices.

For detailed information about the vector formats used by COS, refer to *NetView/PC: Application Programming*.

Four NetView COS commands are used with NetView/PC for problem determination:

LINKTEST
Requests that the COS test a given link or link segment

LINKDATA
Requests that the COS return device data for a given link or link segment

LINKPD
Requests problem determination analysis from the COS on a given link or link segment

RUNCMD
Sends COS application commands to the COS applications from the NetView program

Refer to the NetView online help for the syntax of the LINKTEST, LINKDATA, LINKPD, and RUNCMD commands.

The COS commands are long-running commands that suspend the command list when they are processed. The command list resumes when the COS command is complete. While the command list is suspended, no other commands waiting on the low-priority queue of the task running the command list are processed until the command list completes. This ensures that commands on the low-priority queue can be processed in order.

You cannot use the NetView command list language &WAIT control statement or the REXX TRAP and WAIT functions with the COS commands. Use automation table-driven command lists to trap messages generated from the COS commands, except for:

- LINKPD messages
  - DSI533I
  - DSI534I
  - DSI535I
  - DSI536I
  - DSI582I

These five messages are set to values you can use in the form of control and parameter variables. See "LINKPD Results" on page 157 for more information about LINKPD results.

- Responses to RUNCMD with the CLISTVAR keyword. The CLISTVAR keyword causes the responses to be stored in variables. See "RUNCMD Results" on page 158 for more information about RUNCMD results.

Common Operations Services Return Codes

After the command is completed, the 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	Description
0	The command succeeded.
4	The command failed, CLISTVAR was specified with RUNCMD but no response was returned, or the task is not authorized to issue the command.
16	The command was canceled by the CANCMD.
24	Some command list data was truncated.
28	The COS application returned more than 132 responses for the RUNCMD with the CLISTVAR keyword.
32	The COS application did not respond within the amount of time specified by the COS command timeout value in the NetView constants module.

LINKDATA and LINKTEST Results

You can use LINKDATA and LINKTEST in command lists to manage COS, for example, NetView/PC. Refer to the NetView online help for the formats of these commands.

Use the variable name without the ampersand for REXX command lists and the variable name with the ampersand for NetView command list language command lists.

Note: The path number is 01 for LINKTEST and LINKDATA.

LINKDATA and LINKTEST Variables

You can use the following LINKDATA and LINKTEST variable names in command lists:

DSIPATHCNT or **&DSIPATHCNT**
Specifies the number of paths returned. It is always 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**
Specifies the 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**
Specifies the 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**
Specifies the name of the resource *rr* on path *pp*.

DSI*pprr***RT** or **&DSI***pprr***RT**
Specifies the type of the resource *rr* on path *pp*.

DSI*pprree***DN** or **&DSI***pprree***DN**
Specifies the name of the data item *ee* for resource *rr* on path *pp*.

DSI*pprree***DT** or **&DSI***pprree***DT**
Specifies the type of data item *ee* for resource *rr* on path *pp*. The following values are possible:
- BIT STRING
- CHARACTER
- DECIMAL
- HEXADECIMAL

DSI*pprree***DV** or **&DSI***pprree***DV**
Specifies the value of data item *ee* for resource *rr* on path *pp*.

The italicized letters in the variable names are replaced with the following values:

pp	Path number (01)
rr	Resource number (01-99)
ee	Entry number (01-99)

LINKTEST Additional Variables

In addition, LINKTEST uses the following variables:

DSIREQUEST or **&DSIREQUEST**
Specifies the number of tests requested.

DSIACTUAL or **&DSIACTUAL**
Specifies the actual number of tests processed.

DSITESTTYPE or **&DSITESTTYPE**
Indicates the type of test data reported. The following values are possible:
- BACKGROUND
- REQUESTED

DSIRESULT or **&DSIRESULT**
Indication of the overall results of the test processing. The following values are possible:
- PASSED
- FAILED
- INDETERMINATE

LINKPD Results

Results from the LINKPD command are returned in messages that you can use in a command list to automate the recovery of resources controlled by a COS, for example, NetView/PC.

LINK results can be accessed by the MSGCNT(), MSGID(), MSGORIGN(), MSGSTR(), MSGTYP(), and MSGVAR(1) through MSGVAR(31) functions.

For more information about the REXX functions MSGORIGN(), MSGID(), MSGCNT(), MSGSTR(), and MSGTYP(), see “Message Processing Information Functions” on page 52. For more information about MSGVAR(1) through MSGVAR(31), see “Functions Set by MSGREAD” on page 37.

LINKPD results can be accessed by the &MSGCNT, &MSGID, &MSGORIGIN, &MSGSTR, &MSGTYP control variables, and the parameter variables &1-&31.

For more information about the NetView command list language control variables &MSGCNT, &MSGID, &MSGORIGIN, &MSGSTR, and &MSGTYP see “Message Processing Information Functions” on page 52. For more information about parameter variables used in command lists written in the NetView command list language, see “Parameter Variables” on page 102.

RUNCMD Results

If you use RUNCMD without the CLISTVAR keyword, responses from the COS application that performed the RUNCMD are sent to the network operator terminal, and a return code is set. See “Common Operations Services Return Codes” on page 156 for a description of the return codes.

In REXX, the return code is returned in the RC variable.

In NetView command list language, the return code is returned in the &RETCODE variable.

If you use RUNCMD with the CLISTVAR keyword, the command has the following results:

- A return code is set (RC or &RETCODE); see “Common Operations Services Return Codes” on page 156 for a description of the return codes.
- If the command completes with a return code of 0, 24, or 28, the following variables are set.

  Note: 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 COS application. The variable has a value in the range 001–998.

  DSIRUN*xxx* or **&DSIRUN***xxx*
  Contains the different responses from the COS application. The responses are numbered 001–998.

  Note: The responses from the COS must be character data and cannot be longer than 255 characters.

If you use RUNCMD with CLISTVAR=YES in a PIPE command, you receive message BNH074I. This occurs even if the PIPE command is issued from a NetView command list. You must use CLISTVAR=NO (the default) in the PIPE command.

Using RUNCMD in a Pipeline

An alternative to using CLISTVAR=YES is to run your RUNCMDs in a pipeline. Using pipelines has the following advantages:

- You can run multiple calls of RUNCMD in parallel.
- You have direct control over the timeout value.
- You can filter the data in the pipeline before setting any variables.
- You can choose the names of variables. REXX-style stem names can be used.

For example, if you used the following segment of REXX code to set up commands for service points:

```
RCMD.1 = 'RUNCMD SP=NT000001,APPL=APPLNAME,QUERY ABC'
RCMD.2 = 'RUNCMD SP=NT000002,APPL=APPLNAME,QUERY ABC'
RCMD.3 = 'RUNCMD SP=NT000003,APPL=APPLNAME,QUERY ABC'
:
RCMD.100 = 'RUNCMD SP=NT000100,APPL=APPLNAME,QUERY ABC'
RCMD.0 = 100
```

You can then use the following command to issue all 100 commands as fast as the requests can be scheduled, without waiting for results. The NetView program then waits up to 120 seconds (between messages) for results and places the messages in stem RESULT.

```
'PIPE STEM RCMD. | NETVIEW | CORRWAIT 120 | STEM RESULT.'
```

Notice that this command does not wait 120 seconds after the last result (assuming all 100 commands have completed). The NetView program counts the commands processed and notes when the last response to each command has arrived. The results in stem RESULT. are not necessarily in the same order as the commands in stem RCMD. because some service points respond faster than others. The number of lines stored in RESULT. is found in RESULT.0.

For more information about using NetView pipelines, refer to *IBM Tivoli NetView for z/OS Programming: Pipes*.

Appendix E. Comparison of REXX and NetView Command List Language

This appendix provides a brief comparison between REXX and the NetView command list language.

Comparison of REXX Instructions and NetView Command List Language Control Statements

Table 19 shows each control statement used in the NetView command list language and provides the equivalent REXX instruction. The table is in alphabetic order 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 an instruction provided by the NetView program.

Instructions provided by the NetView program can be used only with the Tivoli NetView for z/OS program. These instructions are not supported by the REXX interpreter and cannot be used in REXX execs run in a non-NetView environment.

Table 19. Comparison of REXX Instructions and NetView Command List Language Control Statements

REXX Instruction	Described on	NetView Control Statement	Described on	REXX Instruction Provided By
None	N/A	&BEGWRITE	113	N/A
CGLOBAL(*name*)	52	&CGLOBAL	149	NetView
TRACE	32	&CONTROL	110	REXX
EXIT	131	&EXIT	131	REXX
SIGNAL	33	&GOTO	131	REXX
IF	129	&IF	129	REXX
PARSE EXTERNAL	26	&PAUSE	114	REXX
PARSE PULL	26	&PAUSE	114	REXX
TGLOBAL(*name*)	52	&TGLOBAL	148	NetView
TRAP	*	&WAIT	133	NetView
WAIT	*	&WAIT	133	NetView
MSGREAD	*	&WAIT	133	NetView
FLUSHQ	*	&WAIT	133	NetView
SAY	27	&WRITE	112	REXX
* See the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)*.				

Comparison of REXX Functions and NetView Command List Language Control Variables and Functions

Table 20 shows the various control variables and functions used in the NetView command list language and the equivalent REXX functions.

If the function is provided by the NetView program, it can be used only with the NetView program and is not supported by SAA REXX.

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions

REXX Function or Variable	Described on	NetView Control Variable	REXX Function Provided By
ACTIONDL()	53	&ACTIONDL	NetView
ACTIONMG()	53	&ACTIONMG	NetView
APPLID()	83	&APPLID	NetView
AREAID()	53	&AREAID	NetView
ASID()	83	&ASID	NetView
ATTENDED()	83	&ATTENDED	NetView
ATTNID()	53	&ATTNID	NetView
AUTCONID()	83	&AUTCONID	NetView
AUTHCHK(...)	44	None	NetView
AUTHCHKX(...)	46	None	NetView
AUTOTASK()	83	&AUTOTASK	NetView
AUTOTOKE()	53	&AUTOTOKE	NetView
BITAND(...)	117	&BITAND	REXX
BITOR(...)	118	&BITOR	REXX
BITXOR(...)	119	&BITXOR	REXX
CGI()	83	None	NetView
CGLOBAL(*name*)	52, 149	&CGLOBAL	NetView
CMDNAME()	48	None	NetView
CODE2TXT(...)	40	None	NetView
||	119	&CONCAT	REXX
CURCONID()	84	&CURCONID	NetView
CURSYS()	84	&CURSYS	NetView
DATE()	88	&DATE	REXX
DESC()	53	&DESC	NetView
DISC()	84	&DISC	NetView
DISTAUTO()	84	&DISTAUTO	NetView
DOMAIN()	84	&DOMAIN	NetView
DOMAIN('x')	84	None	NetView
ECVTPSEQ()	84	&ECVTPSEQ	NetView
ENVDATA('x')	84	None	NetView
EVENT()	54	None	NetView

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions (continued)

REXX Function or Variable	Described on	NetView Control Variable	REXX Function Provided By
FNDMBR(...)	50	None	NetView
HCOPY()	87	&HCOPY	NetView
HDRMTYPE()	54	&HDRMTYPE	NetView
HIER(*n*)	68, 120	&HIER	NetView
HMASPRID()	68	None	NetView
HMBLKACT()	69	None	NetView
HMCPLINK()	69	None	NetView
HMEPNAU()	70	None	NetView
HMEPNET()	70	None	NetView
HMEPNETV()	71	None	NetView
HMEVTYPE()	71	None	NetView
HMFWDED()	72	None	NetView
HMFWDSNA()	72	None	NetView
HMGENCAU()	73	None	NetView
HMONMSU()	73	None	NetView
HMORIGIN()	74	None	NetView
HMSECREC()	74	None	NetView
HMSPECAU()	75	None	NetView
HMUSRDAT()	76	None	NetView
IFRAUGMT()	54	&IFRAUGMT	NetView
IFRAUIND()	54	&IFRAUIND	NetView
IFRAUIN3()	55	&IFRAUIN3	NetView
IFRAUI3X()	55	&IFRAUI3X	NetView
IFRAUSDR()	55	&IFRAUSDR	NetView
IFRAUSRB()	55	&IFRAUSRB	NetView
IFRAUSB2()	55	&IFRAUSB2	NetView
IFRAUSRC()	55	&IFRAUSRC	NetView
IFRAUSC2()	55	&IFRAUSC2	NetView
IFRAUTA1()	56	&IFRAUTA1	NetView
IFRAUWF1()	56	&IFRAUWF1	NetView
JOBNAME()	56	&JOBNAME	NetView
JOBNUM()	56	&JOBNUM	NetView
KEY()	56	&KEY	NetView
LENGTH(...)	122	&LENGTH	REXX
LINETYPE()	57	&LINETYPE	NetView
LU()	88	&LU	NetView
MCSFLAG()	57	&MCSFLAG	NetView
MSGASID()	57	&MSGASID	NetView

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions (continued)

REXX Function or Variable	Described on	NetView Control Variable	REXX Function Provided By
MSGAUTH()	57	&MSGAUTH	NetView
MSGCATTR()	57	&MSGCATTR	NetView
MSGCMISC()	57	&MSGCMISC	NetView
MSGCMLVL()	58	&MSGCMLVL	NetView
MSGCMSGT()	58	&MSGCMSGT	NetView
MSGCNT()	58	&MSGCNT	NetView
MSGCOJBN()	58	&MSGCOJBN	NetView
MSGCPROD()	58	&MSGCPROD	NetView
MSGCSPLX()	58	&MSGCSPLX	NetView
MSGCSYID()	58	&MSGCSYID	NetView
MSGDOMFL()	59	&MSGDOMFL	NetView
MSGGBGPA()	59	&MSGGBGPA	NetView
MSGGDATE()	59	&MSGGDATE	NetView
MSGGFGPA()	59	&MSGGFGPA	NetView
MSGGMFLG()	59	&MSGGMFLG	NetView
MSGGMID()	60	&MSGGMID	NetView
MSGGSEQ()	60	&MSGGSEQ	NetView
MSGGSYID()	60	&MSGGSYID	NetView
MSGGTIME()	60	&MSGGTIME	NetView
MSGID()	60	&MSGID	NetView
MSGITEM()	60	None	NetView
MSGORIGN()	61	&MSGORIGIN	NetView
MSGSRCNM()	61	&MSGSRCNM	NetView
MSGSTR()	62	&MSGSTR	NetView
MSGTOKEN()	62	&MSGTOKEN	NetView
MSGTSTMP()	62	&MSGTSTMP	NetView
MSGTYP()	63	&MSGTYP	NetView
MSGVAR()	63	None	NetView
MSGVAR(*number*)	63	&1 - &31	NetView
MSUSEG(...)	76, 123	&MSUSEG	NetView
MVSLEVEL()	84	&MVSLEVEL	NetView
NVCNT()	49	&NCCFCNT	NetView
NVDELID()	63	&NVDELID	NetView
NVID(*n*)	50	&NCCFID	NetView
NVSTAT(*name*)	50	&NCCFSTAT	NetView
NETID()	84	&NETID	NetView
NETVIEW()	85	&NETVIEW	NetView
NETVIEW('x')	85	None	NetView

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions (continued)

REXX Function or Variable	Described on	NetView Control Variable	REXX Function Provided By
OPID()	82	&OPID	NetView
OPID('x')	82	None	NetView
OPSYSTEM()	85	&OPSYSTEM	NetView
PANEL()	85	None	NetView
PARMCNT()	48	&PARMCNT	NetView
ARG(1)	48	&PARMSTR	REXX
PARTID()	85	&PARTID	NetView
PRTY()	63	&PRTY	NetView
REPLYID()	63	&REPLYID	NetView
RC	49	&RETCODE	REXX
ROUTCDE()	64	&ROUTCDE	NetView
RXDEFENV()	87	&RXDEFENV	NetView
RXDEFSTR()	87	&RXDEFSTR	NetView
RXNUMENV()	87	&RXNUMENV	NetView
RXOVRENV()	87	&RXOVRENV	NetView
RXOVRSTR()	87	&RXOVRSTR	NetView
SESSID()	64	&SESSID	NetView
SMSGID()	64	&SMSGID	NetView
STCKGMT()	85	&STCKGMT	NetView
SUBSTR(...)	126	&SUBSTR	REXX
SUBSYM(...)	42	None	NetView
SUPPCHAR()	85	&SUPPCHAR	NetView
SYSCONID()	64	&SYSCONID	NetView
SYSID()	64	&SYSID	NetView
SYSPLEX()	85	&SYSPLEX	NetView
TASK()	85	&TASK	NetView
TGLOBAL(*name*)	52, 148	&TGLOBAL	NetView
TIME()	88	&TIME	REXX
TOWER(...)	86	None	NetView
TRAP()	86	None	NetView
TYPE()	86	None	NetView
VTAM()	86	&VTAM	NetView
VTCOMPID()	86	&VTCOMPID	NetView
WEEKDAYN()	87	&WEEKDAYN	NetView
WTO.REPLY	64	&WTOREPLY	NetView

Commands Used in Command Lists

These NetView commands that are described in the*IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* and the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* are for use in command lists. Except for the FLUSHQ, MSGREAD, TRAP, and WAIT commands, you can use these commands in command lists written in REXX or in the NetView command list language.

- DOM
- FLUSHQ
- GETMPRES
- GETMSIZE
- GETMTFLG
- GETMTYPE
- GLOBALV
- MSGREAD
- MSGROUTE
- PARSEL2R
- SDOMAIN (with QUIET option)
- TRAP
- WAIT
- WTO
- WTOR

Note: The FLUSHQ, MSGREAD, TRAP, and WAIT commands can be used only in REXX command lists.

When using the commands in a REXX command list, enclose in single quotation marks the parts of the command on which you do not want variable substitution to take place.

Appendix F. Command List Examples Index

This appendix contains reference tables for the REXX and NetView command list examples contained in this book. Entries in the tables are listed in alphabetic order.

The tables show the name of the command list example, a brief description of its function, and where to find the example in this book.

REXX Command List Examples

Table 21 lists the REXX command list examples shown in this book.

Table 21. REXX Command List Examples Reference

Command List Example	Description	Location
ACTAPPLS	This command list displays active applications.	Figure 49 on page 169
ACTLU	This command list activates a VTAM node.	Figure 50 on page 171
CHKOPNUM	This command list shows how basic REXX functions and NetView-specific functions can be used in command lists. CHKOPNUM illustrates the use of such things as the REXX PARSE instruction, and the NetView MSGTRAP, WAIT, MSGREAD, and GLOBALV commands.	Figure 51 on page 172
CHKRSTAT	This command list shows how more complex REXX functions and NetView-specific functions can be used in command lists. CHKRSTAT illustrates the use of the REXX INTERPRET instruction, and the NetView WAIT and GETMLINE commands.	Figure 52 on page 174
CNMS1101	This command list is an example of PPI communication and of full screen automation.	"CNMS1101" on page 175
CNME1080	This is an example of updating a common global variable re-entrantly.	"CNME1080" on page 187
CNMESRVAR	This is an example of updating a single Revision Variable.	"CNMSRVAR Example" on page 188
CNMSRVMC	This is an example of Command Revision's NETVONLY action.	"CNMSRVMC Example" on page 191
DSPRSTAT	This command list can be used by an operator station task (OST) operator to display the results of several runs of the CHKRSTAT command list for a specific resource. Use DSPRSTAT as an aid when you must determine how often a resource is active, based on the intervals in which it was checked by the CHKRSTAT command list.	Figure 57 on page 194
GETCG	The GETCG command list gets the value of a common global variable and displays it to the requesting task.	Figure 58 on page 195
GREETING	This command list shows an example of waiting and trapping using the DATE command.	Figure 59 on page 195
LISTVAR	Refer to the NetView online help for a functional description of this command list.	Figure 60 on page 196
PRINT	This command list prints members of a data set to a system print file.	Figure 61 on page 198

Table 21. REXX Command List Examples Reference (continued)

Command List Example	Description	Location
TYPE	This command list displays members of a data set one line at a time at the terminal of the user who issued it.	Figure 62 on page 199
TYPEIT	This command list displays members of a data set one line at a time at the terminal of the user who issued it.	Figure 63 on page 200

NetView Command List Language Examples

Table 22 lists the NetView command list language command list examples shown in this book.

Table 22. NetView Command List Examples Reference

Command List Example	Description	Location
ACTONE	This command list issues a VTAM command to activate a logical unit (LU). The ACTONE command list shows the use of &WAIT to wait for one message.	Figure 44 on page 144
CLIST1	The CLIST1 command list contains the nested command list UPDT1. CLIST1 and UPDT1 show how to define, reference, and update a task global variable.	Figure 46 on page 150
GLOBVAR1	The GLOBVAR1 command list illustrates the scope of user variables, task global variables, and common global variables within individual command lists.	Figure 48 on page 152
PATH	This command list uses the &WRITE control statement and a VTAM command.	"&WRITE Control Statement" on page 112
UPDT1	The CLIST1 command list contains the nested command list UPDT1. CLIST1 and UPDT1 show how to define, reference, and update a task global variable.	Figure 47 on page 151

Appendix G. Examples of REXX Command Lists for NetView

This section contains examples of REXX command lists written for the NetView program. These examples show how you can use the instructions and functions provided by NetView and the standard REXX instructions and functions together in REXX command lists running in a NetView environment.

ACTAPPLS Example

```
/* ********************************************************************/
/*                                                                    */
/*  ACTAPPLS - REXX VERSION                                           */
/*                                                                    */
/*  DISPLAY ONLY THE ACTIVE APPLS                                     */
/*                                                                    */
/* ********************************************************************/
TRACE E
SAY 'ACTIVE APPLICATIONS:'          /* Write the header              */
SAY '====================='
'TRAP SUPPRESS MESSAGES  IST350I IST097I'   /* Wait on the display  */
'D NET,APPLS'
'WAIT 60 SECONDS FOR MESSAGES'
DO WHILE EVENT() = 'M'
  SELECT                            /* SELECT on all events          */
    WHEN EVENT() = 'M' THEN
      DO
        'MSGREAD'
        SELECT                      /* SELECT on message             */
          WHEN MSGID()='IST350I' THEN
            CALL FIRST
          OTHERWISE
            CALL ALLELSE
        END                         /* END - SELECT                  */
        'WAIT CONTINUE'
      END                           /* EVENT() = M do loop           */
    OTHERWISE
      DO
        'TRAP NO MESSAGES'
        'FLUSHQ'
      END
  END                               /* END - SELECT                  */
END                                 /* END - DO WHILE                */

/*                                                                    */
/*  ALL NON-INFORMATIONAL MESSAGES GO HERE                            */
/*                                                                    */
ALLELSE:
RETURN
/*                                                                    */
/*  THE MULTILINE WTO WITH THE APPL INFORMATION COMES HERE            */
/*                                                                    */
```

Figure 49. ACTAPPLS Example (Part 1 of 2)

```
FIRST:
'GETMSIZE NUMLINES'                /* Determine the number of lines */
I = 0                              /* Initialize line number counter*/
TOTALACT = 0                       /* Initialize total active appls */
DO WHILE NUMLINES ¬= I             /* DO for all lines              */

  NUMACT = 0                       /* Number of active appls found
                                      on this line                  */
  I = I + 1                        /* Bump the line counter         */
  'GETMLINE LINE' VALUE(I)         /* How many lines in the MLWTO?  */
/* PARSE OUT THE LINE, A1 A2 A3 ARE APPL NAMES, S1 S2 S3 ARE STATUS  */
  PARSE VAR LINE MSG A.1 S.1 A.2 S.2 A.3 S.3 .

  DO CURR = 1 TO 3
    IF S.CURR ¬= '' THEN           /* Do we have a status?           */
      DO
        IF S.CURR = 'ACTIV' THEN   /* Is the current APPL active?    */
          NUMACT = NUMACT + 1      /* Bump the number active count   */
        ELSE
          DO
            S.CURR = ''            /* APPL not active, so blank out  */
            A.CURR = '
          END
      END
    ELSE
      A.CURR = ''                  /* Not an APPL                   */
  END                              /* END - DO CURR                 */
  IF NUMACT ¬= 0 & (A.1 ¬= '' | A.2 ¬= '' | A.3 ¬= '') THEN
    SAY STRIP(A.1 A.2 A.3,'L')
  TOTALACT = TOTALACT + NUMACT     /* Bump the total active counter */
END                                /* END - DO WHILE                */

SAY ' '                            /* Blank line                    */
SAY 'NUMBER OF APPLICATIONS ACTIVE: 'TOTALACT
EXIT
```

Figure 49. ACTAPPLS Example (Part 2 of 2)

ACTLU Example

```
/*  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 TIME-OUT 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 TIME-OUT ENCOUNTERED?  */
            SAY 'NO RESPONSE TO ',  /* INFORM USER                 */
                'ACTLU CLIST FOR 'NODE
          OTHERWISE                 /* NO-OP                       */
        END                         /* OF SELECT FOR ERROR/TIME-OUT */
    END                             /* IF RC=0  (VALID NODENAME)   */
  END                               /* IF NODE¬='STOP' PROCESSING  */
EXIT
```

Figure 50. ACTLU Example

CHKOPNUM Example

Figure 51 on page 172 is an example of a command list that uses the PARSE instruction.

```
/*********************************************************************/
/*                                                                   */
/*  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' COMMANDS.   */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*  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 AT A CERTAIN TIME (COULD BE           */
/*     ANY TIME PERIOD);                                             */
/*     ->  'AT 08:00:00,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                               */
/*  -----------  --------  ----------------------------------------- */
/*                                                                   */
/*********************************************************************/
```

Figure 51. CHKOPNUM Example (Part 1 of 2)

```
SIGNAL ON ERROR
PARSE ARG OPTION
'GLOBALV GETC CHKOPTIMES, CHKOPNUM, CHKOPMAX'
IF OPTION = 'DISPLAY' THEN DO;

  IF CHKOPTIMES = '' THEN
    SAY 'NUMBER OF OPERATORS HAS NEVER BEEN CHECKED'
  ELSE DO;
  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;
  EXIT 0;
END;
CUROPNUM = 0
'TRAP AND SUPPRESS MESSAGES OPERATOR:,END'
'LIST STATUS=OPS'
DO UNTIL MSGID()='END'
  'WAIT FOR MESSAGES'
  'MSGREAD'
  IF MSGID() = 'OPERATOR:' THEN CUROPNUM = CUROPNUM +1
  ELSE NOP
END
IF CHKOPTIMES = '' THEN CHKOPTIMES = 1
ELSE CHKOPTIMES = CHKOPTIMES + 1
IF CHKOPNUM = '' THEN CHKOPNUM = CUROPNUM
ELSE CHKOPNUM = CHKOPNUM + CUROPNUM
IF CHKOPMAX = '' THEN CHKOPMAX = CUROPNUM
ELSE IF CHKOPMAX < CUROPNUM THEN CHKOPMAX = CUROPNUM
'GLOBALV PUTC CHKOPTIMES, CHKOPNUM, CHKOPMAX'

EXIT 0;
ERROR: SAY 'ERROR OCCURRED.  RETURN CODE = ' RC
EXIT -1;
```

Figure 51. CHKOPNUM Example (Part 2 of 2)

CHKRSTAT Example

Figure 52 on page 174 is an example of a command list that uses the INTERPRET instruction.

```
/*********************************************************************/
/*                                                                   */
/*  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 MULTILINE 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                               */
/*  -----------  --------  ----------------------------------------- */
/*                                                                   */
/*********************************************************************/
SIGNAL ON ERROR            /* SIGNAL IF ERROR OCCURS                 */
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
/* 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         */
```

Figure 52. CHKRSTAT Example (Part 1 of 2)

```
'MSGREAD'
IF MSGID() = 'IST097I' THEN DO;
  'WAIT CONTINUE'
  'MSGREAD'
  /* 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 'STATUS=' 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 NONNUMERIC    */
  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 PERFORM 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 NONNUMERIC  */
   ELSE
     NACT# = NACT# + 1
  INTERPRET VARNAME'=NACT#'
  'GLOBALV PUTC 'VARNAME
END
EXIT 0;
ERROR: SAY 'ERROR OCCURRED.  RETURN CODE IS  ' RC
EXIT -1;                    /* END COMMAND LIST FOR ERROR             */
```

Figure 52. CHKRSTAT Example (Part 2 of 2)

CNMS1101

Figure 53 on page 176 is an example of PPI communication and of full screen automation.

```
/*********************************************************************/
/* Licensed Materials - Property of IBM                              */
/* 5697-ENV (C) Copyright IBM Corp. 1997, 2007                       */
/* All rights reserved.                                              */
/* US Government Users Restricted Rights - Use, duplication or       */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*  NAME(PIPESAMP)  SAMPLE(CNMS1101)  RELATED-TO(PIPE)               */
/* ----------------------------------------------------------------- */
/* ---------------  The Purpose of this Sample  -----------------    */
/*  This is a compilation of "examples" given in the Customization:  */
/*  Pipelines book and elsewhere, related to NetView Pipelines.  The */
/*  sample is intended only to save you the effort of typing in      */
/*  examples in order to try them.                                   */
/*                                                                   */
/*  You can run these example directly from CNMS1101, if desired, by */
/*  doing this:                                                      */
/*  1  Copy CNMS1100 into the concatenation under DSICLD             */
/*  2  Invoke it with one or more numbers to indicate the example(s) */
/*     you wish to run.                                              */
/*                                                                   */
/*********************************************************************/
 SIGNAL ON HALT
 sep = '08'X;  endc = '09'X;  delim = '0A'X
    /*  It is often helpful to use non-printable characters for the  */
    /*  various delimters in our pipeline specifications.  This      */
    /*  reduces the chance that some character in the data will       */
    /*  cause a conflict.                                            */

    /* Examples included are...                                      */
ex.1 = "A LIST '' with a twist."
ex.2 = "Retrieve Alert History."
ex.3 = "MAPCL 'automated' display."
ex.4 = "D A,L with columnar data separated."
ex.5 = "NetView tasks output remapped."
ex.6 = "Picking thru the data with PICK."
ex.7 = "PPI 'responder' - use with 'requestor'..."
ex.8 = "PPI 'requestor' - use with 'responder' above."
ex.9  = "    Preceding two functions can work together when run on"
ex.10 = "    different NetViews running in the same host.         "
ex.11 = "Allocate a new sequential file and write to it.          "
ex.12 = "Interrupt and terminate a previous instance of CNMS1101. "
ex.13 = "Format and send an alert to NPDA."
ex.14 = "Display dynamically updated VIEW of RESOURCE data."
ex.15 = "Display persistant VIEW of RESOURCE data, auto update."
ex.0 = 15
>>--------------------*/
```

Figure 53. CNMS1101 Example (Part 1 of 11)

```
arg functions
IF functions = '' THEN
 DO;
  address netvasis,
  'PIPE (NAME FUNLIST END \)',
  '| STEM EX.',
  '| CM: NLOC 1.1 / /',
  '| COUNT EACHLINE',
  '| COLOR WHITE',
  '| EDIT LINECOUNT 1 1.* NEXTWORD',
  '| T: FANINANY',
  '| A: COLLECT',
  '| CONSOLE ONLY',
  '\ LITERAL /Functions supported are.../',
  '| COLOR PINK',
  '| A:',
  '\ CM:',
  '| COLOR BLUE',
  '| T:'
 END;
ELSE
 DO
  parse var functions step funcArgs
  SELECT
    WHEN (¬datatype(step,'W'))  THEN
      say 'Function codes must be numeric'
    WHEN (step=1)  THEN  CALL PATLIST   funcArgs
    WHEN (step=2)  THEN  CALL ShowALH   funcArgs
    WHEN (step=3)  THEN  CALL MAPCL     funcArgs
    WHEN (step=4)  THEN  CALL DALspread funcArgs
    WHEN (step=5)  THEN  CALL TasksOut  funcArgs
    WHEN (step=6)  THEN  CALL PickData  funcArgs
    WHEN (step=7)  THEN  CALL Responder funcArgs
    WHEN (step=8)  THEN  CALL Requestor funcArgs
    WHEN (step=9)  THEN  CALL NewFile   funcArgs
    WHEN (step=10) THEN  CALL EndMe     funcArgs
    WHEN (step=11) THEN  CALL MakeAlert funcArgs
    WHEN (step=12) THEN  CALL resDyn    funcArgs
    WHEN (step=13) THEN  CALL presDyn   funcArgs
    OTHERWISE
      say 'Function number' step 'out of range.'
  END
 END
EXIT
```

Figure 53. CNMS1101 Example (Part 2 of 11)

```
/*---------------------<<< red white and blue >
/* This one is not found in the book.  See if you can predict what it */
/* does before you run it...                                          */
PATLIST:
   'PIPE (NAME PATLIST END \)',
   '| NETV LIST ''''',           /* what a way to type LIST ''!       */
   '| COLLECT MAX 3',            /* make three line groups ...        */
   '|A: SEPARATE SEQUENCE',      /* separate to many output streams   */
   '| COLOR RED',
   '|B: FANINANY',               /* bringing the stream back together */
   '| CONS ONLY',
     ,       /* ------------------ end of simple pipeline 1 -------- */
   '\A:',                        /* secondary output of separate...   */
   '| COLOR WHITE',
   '|B:',                        /* and send it back upstream         */
     ,       /* ------------------ end of simple pipeline 2 -------- */
   '\A:',                        /* tertiary output of separate...    */
   '| COLOR BLUE',
   '|B:'                         /* and sent it upstream, too         */
RETURN 0

ShowALH:  PROCEDURE    /*-----      Retrieve Alert History       -----*/
'ATTACH NPDA'                     /* Output, including messages will be   */
                                  /* saved for future VET call            */
'PIPE (NAME AHIST1 END +)',       /* Start a pipe                         */
   '| VET NEXT ROWS',            /* give update when it arrives, as image*/
   '| CORRWAIT MOE 60',           /* Wait 60 sec for first NPDA screen    */
   '| NLOCATE 1.7 /BNH150I/',     /* I KNOW what first screen looks like  */
   '| CONSOLE',                  /* show badness                         */
   '| A: LOCATE 1.7 /DWO369I/',  /* Separate timeout message (MOE)       */
   '| VAR timeout',              /* save timeout message                 */
   '+ A:',                       /* other message?                       */
   '| VAR npdamsg'               /*  ...save first for test              */

If  symbol('timeout') = 'VAR' THEN   /* timeout?                          */
   Signal TIMEOUT                 /*  handle unexpected error           */
If  symbol('npdamsg') = 'VAR' THEN   /* some NPDA error                   */
   RETURN 20                      /* messages from NPDA already shown     */
/*-----------------  Down to business  ----------------------*/
/* Type a 'ALH' (Alerts History) in the command area and push enter.      */
'VET /ALH/'                    /* Sending the 'ALH' and an enter key.     */
/* Unlike the example in the book, the following saves the alert history  */
/* data in a SAFE, so that it can be COLLECTed before being shown.        */
```

Figure 53. CNMS1101 Example (Part 3 of 11)

```
Do UNTIL(thispage = lastpage | RC¬=0)
   'PIPE (NAME AHISTLP END =)',    /* start a big pipe                     */
      '| VET NEXT ROWS',           /* give update when it arrives, as image*/
      '| CORRWAIT MOE 60',         /* Wait 60 sec for rnd trip to DST...   */
      '| SC: LOCATE 1.7 /BNH150I/',/* expected screen with data..          */
      '| SEPARATE',                /* handle lines individually...         */
      '| PG: DROP 4',              /* drop header area...                  */
      '| DROP LAST 1',             /* command line                         */
      '| MSG: DROP LAST 3',        /* message area, hopefully blank        */
      '| STRIP TRAILING',          /* shorten line ending in blank, so we  */
      '| LOCATE 1 //',             /*  can toss out blank lines            */
      '| SAFE CNMS1101 APPEND',    /* save data for report to user         */
      '| TAKE 1',                  /* AND use ONE new data line to trigger */
      '| NETV VET /FORWARD/',      /* ...new data, then go to next screen  */
      '= MSG:',                   /* Immed message area from NPDA         */
      '| LOCATE 2.3 /BNJ/',        /* any error message in immed area      */
      '| CONSOLE',                 /* REPORT it, too.                      */
      '| VAR npdamsg',             /*  save for test                       */
      '= SC:',                    /*  message instead ??  This is bad.    */
      '| A: LOCATE 1.7 /DWO369/',  /* Separate timeout message (MOE)       */
      '| VAR timeout',             /* save timeout message                 */
      '= A:',                     /*  other message ? ??  This is bad.    */
      '| CONSOLE',                /*  show badness                        */
      '| VAR npdamsg',             /*  ...save first for test              */
      '= PG:',                    /* get header area                      */
      '| DROP 2',                 /* drop BNH150 and NPDA head-date line  */
      '| VAR pagedata'            /*  ...save pagedata for test           */
   If  symbol('timeout') = 'VAR' THEN   /* timeout?   is possible?         */
      Signal TIMEOUT              /*  handle unexpected error             */
   Parse var pagedata . 'PAGE' thispage . lastpage  /* parse out page nos. */
End                               /*                                      */
'VET /END/'

'PIPE SAFE CNMS1101|COLLECT|CONS'
'PIPE VET |CORRWAIT 10|HOLE'
'DETACH'   /* In the context of CNMS1101, automatic DETACH is not approp.  */
RETURN 0
```

Figure 53. CNMS1101 Example (Part 4 of 11)

```
MAPCL:  PROCEDURE      /* Change the display of MAPCL...                   */
/* ---------------------------------------------------------------------- */
/* Originally designed to run from automation of CNM429I, output of the    */
/* MAPCL command, this version will invoke MAPCL.  If you desire to run    */
/* this code from the automation of CNM429, chaage the "NETV MAPCL" to     */
/* "SAFE *"  The purpose is to insert "Y" or "N"  into the DP              */
/* (drop pending) column according to whether the column is blank.         */
/*            Message attributes are preserved.                            */
     /*------------<<< The color is just for fun. >>>------------*/

'PIPE (NAME CNM429 END >)',
'| NETVIEW MAPCL',                  /* from automation, use "SAFE *"    */
'| TOP: SEPARATE DATA',             /* Send three label lines to "TOP"  */
'| BOTM: TOSTR NOINCL 1.8 /--------/',  /* Send totals lines to "BOTM"  */
'| COLOR GREEN',
'| PLN: LOCATE 61.2 /  /',          /* blank here means 'not pending'   */
'| EDIT 1.* 1 /N/ 62',              /* copy all, then insert "N"        */
'| ALL: FANINANY',
'| COLLECT',
'| CONSOLE ONLY',
 ,                        /*================= end of main pipeline       */
'> TOP:',
'| COLOR WHITE',
'| ALL:',
 ,                        /*================= end of label processing    */
'> BOTM:',
'| COLOR BLUE',
'| ALL:',
 ,                        /*================= end of totals line procsng */
'> PLN:',
'| EDIT 1.* 1',           /* Copy over all of the input text             */
       '/Y/ 62',          /* Insert "Y" where asterisk was               */
'| ALL:'                  /*  and send it back upstream                  */
RETURN 0

DALspread:                /*  Output of D A,L is spread out              */
  /* There are three (?) groups of lines:                                 */
  /*  1. Control and label lines,  we preserve these -- check out       */
  /*     the function "SEPATATE DATA".                                    */
  /*  2. Job status data lines.  These have TWO jobs per line; we'd      */
  /*     like to see ONE job per line.  Here we use CHOP to split        */
  /*     then apart; could be done with EDIT, too.                        */
  /*  3. TSO user status lines.  Not differentiated from type 2, they    */
  /*     get a bit mangled.  More should be done...                       */
```

Figure 53. CNMS1101 Example (Part 5 of 11)

```
'PIPE (NAME DAL END \)',     /*                                      */
'| CORRCMD MVS D A,L',       /* corrcmd adds then needed wait        */
'|A: SEPARATE DATA',         /* short non-data lines to "white" below*/
'|C: CHOP 35',               /* throw "out" data after column 35     */
'| COLOR BLUE',              /* left part of data becomes blue       */
'|Z: FANINANY',              /* parts of data coming back.  Try FANIN*/
'|LABELS: COLLECT',          /* reassemble into  MLWTO               */
'| CONS ONLY',              /*                                      */
  ,                 /* ------------- end of simple pipeline 1 ---------- */
'\C:',                       /* One, last right half may have nothing.  */
'| STRIP TRAILING',          /* This combo strips it, tests for any-    */
'| LOCATE 1 //',             /* thing left? ...so we don't get a blank  */
'| COLOR TUR',               /* line.                                   */
'|Z:',                       /*                                         */
  ,                 /* ------------- end of simple pipeline 2 ---------- */
'\ A:',                      /* ctl & labels come here from SEP DATA    */
'| COLOR WHITE',             /* then we color them and send them back   */
'| LABELS:'                  /* collect above                           */
RETURN 0

TasksOut:                    /* Reformat LIST STATUS=TASKS  */
    /* A demonstration of how to add multiple label lines to a multi-    */
    /* line message.  We also refomat the tabular data to remove         */
    /* extraneous verbiage.  The output lines of LIST STATUS=TASKS       */
    /* look like this:  (scale added below)                              */
/*  TYPE: PPT TASKID: NTV7EPPT RESOURCE: NTV7EPPT STATUS: ACTIVE         */
/*  |...+....1....+....2....+....3....+....4....+....5....+....6....     */

  address NETVASIS,
  'PIPE (NAME TASKLIST END \)',/*                                      */
  '| NETV LIST STATUS=TASKS',  /* issuing the synchronous command       */
  '| DROP LAST 1',             /* throw out "END OF ..."                */
  '| COLOR GREEN',             /* nice color for the data lines         */
  '| EDIT WORD 2 1',           /* getting "type" value; it's first      */
        '19.8   8',           /* at char 19, taskid begins; put in col 8*/
        '38.8  19',           /* at char 38, find resource; move to 19  */
        '55.*  35',           /* at char 55, find status; move to col 35*/
  '| LABS: COLLECT',           /* everything gets collected here        */
  '|CONS ONLY',
  ,                 /* ------------- end of simple pipeline 1 ----------  */
  ,                 /* It's a little weird, but the stuff below here     */
  ,                 /* is absorbed by COLLECT before the data lines      */
  ,                 /* we worked on above.                               */
  '\ FAN: FANIN',              /* stuff from below brought together here,*/
  '| LABS:',                  /* IN ORDER, and then sent upstream      */
  ,                 /* ---------- end of VERY simple pipeline 2  -------- */
  '\ LIT !Status of NetView Tasks!',       /* first label line is       */
  '| COLOR YEL',                           /* made yellow and passed up  */
  '| FAN:',                                /* to FANIN's first input!    */
  "\ LIT !Task   Task's     Taskname or     Current!",   /* second label */
  '| COLOR PINK',                          /* is pink and goes to FANIN's*/
  '| FAN:',                                /* second output, etc.        */
  "\ LIT !type   ID         Resource        Status!",
  '| COLOR PINK',
  '| FAN:'
       /* Notice how the colored label lines remain fixed to the top    */
       /* both on the NCCF screen and in WINDOW.                        */
RETURN RC
```

Figure 53. CNMS1101 Example (Part 6 of 11)

```
PickData: PROCEDURE        /* use PICK to segregate the data             */

'PIPE (NAME LOWUSERX)',
'| NETVIEW MAPCL',         /* obtain display of all REXX in storage */
'| SEPARATE',             /*  handle lines individually            */
'| DROP 3',               /* header lines                          */
'| DROP LAST 2',          /* trailer line and totals               */
'| PICK 14.5 < /    6/',  /* compare 5 chars from data with "6"    */
'| CONSOLE ONLY'          /* display result                        */

/* Luckily, blanks are "less than" number in EBCIDC order.  So the  */
/* comparison works when the number in the data line is more than   */
/* one digit long.                                                  */
RETURN 0

Responder:  PROCEDURE         EXPOSE step

/* PPI "responder":  This program waits (forever) for a request.    */
/* When a request is received, new lines are added to the begining  */
/* and end, color is added and the result sent back.  "Responder"   */
/* is designed to be used with the "requestor", below.              */
/*                                                                  */
/* "Forever", in this context, means "never timeout".  The wait can */
/* end in several ways, including having an operator issue RESET or */
/* a remote operator issue STOP FORCE.  This sample provides for    */
/* an option to end this wait by command.  Since PPI is being used  */
/* as a receiver, commands queued at low priority or by automation  */
/* will be executed promptly, despite the "WAIT *".   The "end me"  */
/* option uses UNIQUE to end a previous instance of CNMS1101.       */
/*                                                                  */


 address NETVASIS,
 'PIPE (NAME PPIW1101 END ;)',
 '|A: PPI PPIS1101',                     /* PPI as receiver, RC -> A: */
 '| WAIT *',                            /* wait "forever"            */
 '| COUNT EACHMSG',                     /*  counting requests        */
 '| COLOR RED REV',                     /*                           */
 '| EDIT  "TC" LINETYPE /Your''s is request/ 1',
        'MSGCOUNT NW',
        'WL',                           /* build "stuff" onto his    */
        '"TL" LINETYPE',                /* request to make our       */
        '/label line / 1',              /* response                  */
        'WL',
        '1.* 1',
 '| EDIT  COPY *   /last line/ NW',     /* more "stuff"              */
 '| COLLECT MAX 1',                    /* fixup line types, ??      */
 '|B: PPI (NV) *',                     /* rtrn answer to sender, NV */
 '| EDIT /Response sent to/ 1 ',       /* report what we did        */
       'IFRAUSDR NW',
```

Figure 53. CNMS1101 Example (Part 7 of 11)

```
 '| CONS',
 '; A:',
 '| NLOCATE 1.11 /+0000000000/',
 '| COLOR WHI',
 '| EDIT  /Error code from receive:/ 1',
        '1.* NW',
 '| CONS',
 ';B:',
 '| NLOCATE 1.11 /+0000000000/',
 '| COLOR YEL',
 '| EDIT  /Error code from send:/ 1',
        '1.* NW',
 '| CONS',
 '; LIT /CNMS1101 7 ready to respond to requests./',
 '| CONS'

RETURN 0

Requestor:  PROCEDURE      /* Send a request to 'responder' coded   */
                           /*  above.                               */
 address NETVASIS,
 'PIPE (NAME PPIQ1101 END ;)',
 '| A: PPI PPIS1101 /req command from' opid() ' at' domain() '/',
 '|  WAIT 52',
 '|  CONS',
 '; A:',
 '| NLOCATE 1.11 /+0000000000/',
 '| COLOR PIN',
 '| EDIT  /Error code from request:/ 1',
        '1.* NW',
 '| CONS'

RETURN 0

NewFile:  PROCEDURE        /* Allocate a new file and write to it.  */

fiName = "'USER1.SEQTEST'"
"ALLOC DATASET("fiName") FILE(OUTFILE)",
  "RECFM(V) CATALOG NEW",
  "BLKSIZE(260) LRECL(256) DSORG(PS) VOLUME(yourname)"

IF rc <> 0 THEN
  SAY 'ALLOCATE ERROR CODE = 'rc
ELSE DO
  'FREE FI(OUTFILE)'
  "PIPE (END ;) LIT /data line/ | > "fiName" | CONS ONLY"
END

RETURN 0

EndMe:  PROCEDURE
'UNIQUE'

RETURN 0
```

Figure 53. CNMS1101 Example (Part 8 of 11)

```
MakeAlert:  PROCEDURE   /* Make an alert and send it to NPDA.       */

altxt = '41038D0000000000'X              /* NMVT header             */
altxt = altxt||'005E0000'X               /* Major vector len & key  */
altxt = altxt||'0B92000001330104059737'X /* Add subvectors...       */
altxt = altxt||'1010000D110E0A0040F2F3F4F5F6F7F8'X
altxt = altxt||'2A9512011711171217131714171517171717171718'X
altxt = altxt||'16811014101510161017101810191020102110221023'X
altxt = altxt||'1103030109C7C5D5C14BF34040C3D6D4C3'X
altxt = altxt||'04931001'X
'pipe (end ;) var altxt|a:ppi *alert|hole;a:|cons'
RETURN 0
/* -------------- Dynamic Resource Display (option 12) -------------- */
/*  A demonstration of using VIEW and TRAP to dynamically update a    */
/*  full screen display.  We use the SPILL option of pipe's KEEP      */
/*  (new for V5) to create a message after the specified refresh      */
/*  interval.  This message is TRAPped, causing VIEW to return        */
/*  control to this procedure WITHOUT removing the displayed panel.   */
/*  The 'RESUME', below is a REINVOCATION of the original VIEW!!!     */
/*                                                                    */
/*  Note that the first call to "fillVars" passes an extra little     */
/*  bit of pipe to the subroutine. The purpose is to get the first    */
/*  word of the second data line (STC name) for the panel.            */
/*                                                                    */
/* ------------------------------------------------------------------ */
resdyn:
  interval = 10              /* refresh at 10 second intervals              */
  privMsgID  = 'CNMRESDYN'   /* special purpose "msgid" for trapping        */
  getSTC = '% STC:|DROP 1|TAKE 1|EDIT W1|VAR JBN'
  Call fillVars getSTC       /* set local variables with data from RESOURCE */
  'TRAP AND SUPPRESS MESSAGES' privMsgID   /* TRAP our special message      */
  'PIPE VAR privMsgID | KEEP RESDYN' interval 'SPILL'  /* make msg later    */
  'VIEW RESDYN CNMSRESP EXTEND'
  DO WHILE (rc = 2)          /* RC indicates "message trapped"?             */
    'MSGREAD'                /* just getting msg off trap queue  */
    CALL fillVars
    'PIPE VAR privMsgID | KEEP RESDYN' interval 'SPILL'  /* make msg later  */
    'RESUME'                 /* Invoke VIEW, previously suspended           */
     /* NOTE: RC, at this point, is RC from VIEW, which was resumed.        */
  END
 'pipe hole | keep resdyn'    /* empty safe created above           */

return
/* ----------- Obtain data for RESDYN display (option 12) ----------- */
/*   Notice that the stem variable "out." is in our local variable    */
/*   dictionary.  VIEW could always read these value; new for V5R1,   */
/*   we will have an opportunity to update them while VIEW is active. */
/*   ---------------------------------------------------------------  */
```

Figure 53. CNMS1101 Example (Part 9 of 11)

```
fillVars:
  ARG xtraStg                          /* use extra first time only     */
  'PIPE (NAME RESDYN END %)',
  '| NETVIEW RESOURCE',
  '| SEPARATE DATA',                   /* No use for DSI386I title line */
  '| STC: FANOUT',                     /* MAYBE need extra copies       */
  '| EDIT SKIPTO /=/ 2.* STRIPL 1 ',
  '| COLOR WHITE',
  '| $STEM OUT.',
  xtraStg
  TM = date() time()
  $TM = 'CB HR'
return

/* -------------- Dynamic Resource Display (option 13) -------------- ::
  This is superficially like the previous like the previous sample
  (excuse our lack of imagination), but shows useful (and, we think,
  EASIER) ways of getting asynchronous information to update a VIEW
  panel.

  We continue to use RESOURCE, making it asynchronous by running it at
  a VOST (hence the ATTACH) and making it repeat by using the time-out
  code from a PIPE WAIT to redrive the command found in the secondary
  stream.  See definition of TimdResc below -- this command could be
  used from the NCCF command line, if desired, but use ACTION=DISPLAY
  with your ATTACH in that case.

  The output of TimdResc is captured by the PERSIST and delivered to
  our trap queue.  The presence of a message there makes VIEW give
  control back to this REXX so that the message can be processed
  (RC=2 from VIEW for this).

  We use the "termination text" option on PERSIST - TRAP to determine
  when we must exit; VIEW always sets EVENT() to M when it returns for
  a message.  We will get the tText when the PERSIST ends either from
  time-out or because of STOP PERSIST, STOP VOST, etc.

  Notice that this sample does not pre-populate the VIEW variables
  with values as in resdyn (12) above. Instead, a LITERAL value in
  the TimdResc command causes a message to be returned to us promptly.
  The first instance of VIEW initializes, then returns control here
  and we populate the values from the first "asynchronous" response.
```

Figure 53. CNMS1101 Example (Part 10 of 11)

```
:: -------------- Dynamic Resource Display (option 13) -------------- */
presDyn:
  address netvasis           /* don't fold tText in pipe below        */
  interval = 10              /* refresh at 10 second intervals        */
  TimdResc =,                /* create the async command              */
   "PIPE (NAME TimdResc END % STAGESEP &)",
    "& A:  WAIT" interval,   /* time-out with R-code every 10 secs    */
    "& HOLE",                /* just here to burn primary stream      */
    "% A:",                  /* time-out code (msg) comes here        */
    "& LITERAL /first msg/", /* hurry!  need a msg quickly            */
    "& NETV RESOURCE",       /* issue RES cmd each time a msg comes   */
    "& CONSOLE"

  tText = "---End---"        /* text of LAST message from trapping    */
  suppchar() ||,             /* do not echo attached cmd              */
  "PIPE (NAME AttTiRs)",
  "| NETV ATTACH (MONO,ACTION=CORR)",   /* correlate msgs from cmd    */
      TimdResc,              /* command to make data every 10 secs    */
  "| PERSIST 1440 TRAP" tText  /* No wait->ALL msgs to the trap queue */

  'VIEW RESDYN CNMSRESP EXTEND'
  DO WHILE (rc = 2)          /* RC indicates "message trapped"?       */
    'MSGREAD'                /* get one msg off trap queue -> current */
    SELECT;
      WHEN msgID()='DSI386I' THEN  /* this is expected,...            */
       do;
        CALL preFillVars     /* Update local variables from msg data  */
        'RESUME'               /* Invoke VIEW, previously suspended   */
       end;
      WHEN  msgID() = tText  THEN  /* persist ended... t/o or stop'd  */
         RC = 0                 /* to exit loop                       */
      OTHERWISE                 /* Unexpected msg.  what failed?      */
        'PIPE SAFE * | LIT /pResDyn ends.../|LOGTO *' /* also set RC  */
    END;
     /* NOTE: RC, after RESUME, is RC from VIEW, which was resumed.   */
  END

return
/* ----------- Obtain data for pRESDYN display (option 13) ---------- */
/*   Notice that the stem variable "out." is in our local variable    */
/*   dictionary.  VIEW could always read these values; new for V5R1,  */
/*   we will have an opportunity to update them while VIEW is active. */
/*   ---------------------------------------------------------------  */
preFillVars:
  'PIPE (NAME PRESDYN END %)',
  '| SAFE *',
  '| SEPARATE DATA',                 /* No use for DSI386I title line */
  '| STC: FANOUT',                   /* MAYBE need extra copies       */
  '| EDIT SKIPTO /=/ 2.* STRIPL 1 ',
  '| COLOR WHITE',
  '| $STEM OUT.'
  TM = date() time()
  $TM = 'CB HR'
return
HALT:
 IF symbol('step') = 'VAR'  THEN
   say 'CNMS1101 function' step 'ends.'
 'pipe hole | keep resdyn'    /* empty safe created by resdyn (12)  */
EXIT -5
```

Figure 53. CNMS1101 Example (Part 11 of 11)

CNME1080

Figure 54 is an example of updating a common global variable re-entrantly.

```
/*REXX*---------------------------------------------------------------*/
/*  Licensed Materials - Property of IBM                             */
/*  5697-ENV © Copyright IBM Corp. 2009                              */
/*  All rights reserved.                                             */
/*                                                                   */
/*  US Government Users Restricted Rights - Use, duplication or      */
/*  disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*-------------------------------------------------------------------*/
/* ----------->
Sample NETVONLY action command.  This command gets control as a result
of a Command Revision action:  NETVONLY=CNMSRVMC

It is designed to respond to a pseduo-command "V NetView,LOGON" issued
from a regular MVS console.  Or to a START command for a special
procedure that should not be run during normal working hours.

To see what triggers this command, read sample CNMSCRT1.
Customers can code any number of different REXX procedures to be
invoked by the NETVONLY action.  This sample is only a model.

Use any REXX function here, but the message based functions that return
meaningful data are only sysconid() and sysid(). Use the recedata()
function to obtain information about the origin of the command.
   <---------- */
ADDRESS NETVASIS
/* important!  prevent abuse by checking for proper environment      */
callerASID = RECEDATA()            /* Obtain ASID of cmd issuer      */
IF callerASID = 0 THEN             /* no RECE data?  Then call must  */
  SIGNAL BadEnvr                   /* not be from NETVONLY action.   */

/* The "command" that triggered NETVONLY is our arg...               */
fullCmd = arg(1)
parse arg verb Parm1 ',' theRest
/* We handle two verbs:  'S' or 'START' and 'V' or 'VARY'.  Consider
   writting two procedures for simplicity.
   Both verbs come here ONLY when other conditions in the revision
   table are satisfied. See CNMSCRT1.                              */
SELECT
  WHEN (LEFT(verb,1) = 'S') THEN    /* It was a START cmd           */
     CALL startTooBig
  WHEN (LEFT(verb,1) = 'V') THEN    /* VARY cmd with NetView arg    */
     CALL getNVassoc
  OTHERWISE           /* Should Not Occur unless CNMSCRT1 changed.  */
   do;
     msg = 'RECE command CNMSRVMC driven by system command "' ||,
           arg(1) || '". Please review CRT.'
     'pipe VAR msg | ROUTE AUTRCVR'
     /* We purposely do not REISSUE here. Operator will get CNM017E */
   end;
END

exit 0
```

Figure 54. CNME1080 Example (Part 1 of 2)

```
/* Ask the operator to confirm.  Wait for response to the WTOR.
   Note: Waiting for the response here DOES NOT block other
         NETVONLY actions from proceeding.
   Note: if confirmed, this procedure issues the exact same command
         that caused this NETVONLY action.  This will not start a loop
         and the reissued command will appear to have originated in the
         same environment as the original command (username &
         console).
  ---------- */
startTooBig:
   WTO.TEXT = 'TLH916W Procedure' PARM1 'is intensive.',
   'Answer "Y" to continue'
   'WTOR'
   IF WTO.REPLY = 'Y' THEN
     'REISSUE MVS' fullCmd
   ELSE
    do;
     WTO.TEXT = 'TLH917I start' PARM1 'canceled.'
     'WTO'
     'REISSUE SUPPRESS'
    end;
RETURN            /* from startTooBig  */
getNVassoc:
  username = RECEDATA('U')
  SELECT
   WHEN (username = '*BYPASS*') THEN
    'WTO TLH006E Please logon to your system console'
   WHEN (sysconid() = 'INSTREAM' | sysconid() = 'INTERNAL') THEN
    'WTO TLH026E Console' sysconid() 'cannot associate with NetView.'
   OTHERWISE
    do
     'AUTOTASK OPID=' || username || 'CONSOLE=' || sysconid()
     IF rc = 0 THEN
      'MSG' username 'association with NetView,' domain()', successful.'
     ELSE
      'WTO TLH662E Association with NetView failed:' RC
    end
  END
  'REISSUE SUPPRESS'
RETURN            /* from getNVassoc   */
BadEnvr:
  'MESSAGE DSI290 CNMSRVMC' opid()
  exit 12
```

Figure 54. CNME1080 Example (Part 2 of 2)

CNMSRVAR Example

Figure 55 on page 189 is an example of updating a single Revision variable.

```
/*********************************************************************/
/* Licensed Materials - Property of IBM                              */
/* 5697-ENV © Copyright IBM Corp. 2009                               */
/* All rights reserved.                                              */
/*                                                                   */
/* US Government Users Restricted Rights - Use, duplication or       */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*********************************************************************/
/* --->
  This sample shows how to manipulate the "revision variable" table
  that exists in the NetView SSI and is loaded or queried using the
  SETRVAR command. This sample, invoked as a command, will accept a
  value of one variable.  If the variable currently has a value, the
  new value will be substituted, without altering the other, existing
  variables.  If the variable is new, it will be added.

  Remember that a null value (zero length) is the same (for revision
  variables) as not defined.  Therefore this sample can be used to
  delete one variable.

  Syntax: CNMSRVAR varname /new value/
  Where
    varname      is any alphanumeric string, 1 to 12 characters
    /new value/  is a delimited string, 0 to 16 characters (use any
                 delimiter, not only slash). Alternatively, user
                 may specify the new value as hex string, for example
                 CNMSRVAR myvar '94A840A58193A485'X

  Remember - only the variable named is affected.  Other variables
  are not changed.
  <--- */
SIGNAL ON FAILURE             /* Report any serious error            */
SIGNAL ON NOVALUE             /* Report serious error                */
SIGNAL ON HALT                /* Report reset condition              */
ADDRESS NETVASIS

  parse arg varname other     /*                                     */
  IF varname = ''  THEN       /* varname required; value may be null */
    SIGNAL NOARG
   /* The PARSE VAR below extracts a value from a delimited string OR
      from the ' 'X sequence that identifies hex                     */
```

Figure 55. CNMSRVAR Example (Part 1 of 3)

```
IF other =  '' THEN         /* no value?                           */
    parse value ' ' WITH wantHex theValue  /* nulls for delete action */
  ELSE                       /* value is delim string or hex        */
   do;
    other = strip(other,'L')      /* for easier parsing               */
    parse var other delim +1 theValue (delim) wantHex theRest

    IF delim = '7D'X & translate(wantHex) = 'X'  THEN  /* hex!        */
      wantHex = ''         /* absorb only valid value for this       */
    ELSE         /* SETRVAR reads only hex from current message, so.. */
       theValue = c2x(theValue)   /* we submit hex only to SETRAVAR   */
   end;
  IF wantHex <> '' THEN           /* not "X", so... what?             */
    SIGNAL BADKEY

/* --->
 */
We will feed the SETRVAR command the results of a SETRVAR QUERY. If
  that message is not altered then it simply reloads the table with the
  same values. (SETRVAR ignores label lines on input.)  However, this
  pipe will alter the message by deleting the line with our input
  variable name and then inserting a new one with the new value.
  When called for "current message," SETRVAR reads only the hex part
  of each line and it expects that hex to begin in position 33, as it
  does in the output of SETRVAR QUERY.
  <--- */

  varname = translate(varname)            /* upper case to match msg  */
  NewLine = left(varname,32) || theValue  /* mimic line of setrvar qry*/

  'PIPE (NAME CNMSRVAR END &)',
  '| NETV SETRVAR QUERY',
  '| WAIT 10',
  '| NLOCATE 1.S /DSI231I/',     /* no table?  we don't care          */
  '| x: LOCATE 1.S /BNH332I/',   /* only other valid response        */
  '| SEPARATE',
  '| NLOCATE 1.S /'left(varname,12)'/',   /* 12 = max name length     */
  '| in: FANIN',                /* remainging lines, plus one new     */
  '| collect',                  /* important: drive SETRVAR only once */
  '| NETV SETRVAR *',           /* load table from current message    */
  '| WAIT 10',                  /* even here,it's asynchronous        */
  '| CONSOLE',                  /* show any error message             */
  '& VAR NewLine',              /* read in substitute line            */
  '| in:',                      /* and pass up to FANIN               */
  '& x:',                       /* other msg? SSI down? rtr down?     */
  '| CONSOLE',                  /* show error                         */
  '| PIPEND 28'                 /* end pipe with error                */
```

Figure 55. CNMSRVAR Example (Part 2 of 3)

```
  /* The next bit is optional.  Let operator know that it all worked. */
  IF RC = 0  THEN               /* pipe & commands work?              */
   'SETRVAR QUERY'              /* reassure oper                      */

  /*
   Sample output from SETRVAR QUERY.
   Deleting a data line and substituting another causes the revision
   variable table to be loaded with one variable different.
   SETRVAR sets new values from the hex portion of the line.
BNH332I For NETV there are 2 revision variables.
BNH325I Table loaded by TOM at 09/09/08 19:00:33
MYFIRST       Lovely thing!     D396A58593A840A3888995875A
SOLEMVAR      keep Trucking?    9285859740E399A483928995876F
|                               |
1                               33
                                        */
exit RC

noARG:      /* TEXT OPERAND MISSING OR INVALID */
  'MESSAGE DSI004I'
  EXIT 8
BADKEY:
  'MESSAGE DSI486I' wantHex
FAILURE:
  'MESSAGE CNM996E' cmdname()
  EXIT sigl
NOVALUE:
'MESSAGE BNH355E' sigl 'CNMSRVAR 4' cmdname() CONDITION(D)
EXIT 4
HALT:                           /*                                    */
 EXIT -5                        /*
```

Figure 55. CNMSRVAR Example (Part 3 of 3)

CNMSRVMC Example

Figure 56 on page 192 is an example of the Command Revision NETVONLY action.

```
/*REXX*---------------------------------------------------------------*/
/*  Licensed Materials - Property of IBM                             */
/*  5697-ENV © Copyright IBM Corp. 2009                              */
/*  All rights reserved.                                             */
/*                                                                   */
/*  US Government Users Restricted Rights - Use, duplication or      */
/*  disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*-------------------------------------------------------------------*/
/* ----------->
Sample NETVONLY action command.  This command gets control as a result
of a Command Revision action:  NETVONLY=CNMSRVMC

It is designed to respond to a pseduo-command "V NetView,LOGON" issued
from a regular MVS console.  Or to a START command for a special
procedure that should not be run during normal working hours.

To see what triggers this command, read sample CNMSCRT1.
Customers can code any number of different REXX procedures to be
invoked by the NETVONLY action.  This sample is only a model.

Use any REXX function here, but the message based functions that return
meaningful data are only sysconid() and sysid(). Use the recedata()
function to obtain information about the origin of the command.
   <---------- */
ADDRESS NETVASIS
/* important!  prevent abuse by checking for proper environment      */
callerASID = RECEDATA()            /* Obtain ASID of cmd issuer      */
IF callerASID = 0 THEN             /* no RECE data?  Then call must  */
  SIGNAL BadEnvr                   /* not be from NETVONLY action.   */

/* The "command" that triggered NETVONLY is our arg...               */
fullCmd = arg(1)
parse arg verb Parm1 ',' theRest
/* We handle two verbs:  'S' or 'START' and 'V' or 'VARY'.  Consider
   writting two procedures for simplicity.
   Both verbs come here ONLY when other conditions in the revision
   table are satisfied. See CNMSCRT1.                               */
SELECT
  WHEN (LEFT(verb,1) = 'S') THEN    /* It was a START cmd           */
     CALL startTooBig
  WHEN (LEFT(verb,1) = 'V') THEN    /* VARY cmd with NetView arg    */
     CALL getNVassoc
  OTHERWISE            /* Should Not Occur unless CNMSCRT1 changed.  */
   do;
     msg = 'RECE command CNMSRVMC driven by system command "' ||,
           arg(1) || '". Please review CRT.'
     'pipe VAR msg | ROUTE AUTRCVR'
     /* We purposely do not REISSUE here. Operator will get CNM017E */
   end;
END

exit 0
```

Figure 56. CNMSRVMC Example (Part 1 of 3)

```
/* Ask the operator to confirm.  Wait for response to the WTOR.
   Note: Waiting for the response here DOES NOT block other
         NETVONLY actions from proceeding.
   Note: if confirmed, this procedure issues the exact same command
         that caused this NETVONLY action.  This will not start a loop
         and the reissued command will appear to have originated in the
         same environment as the original command (username &
         console).
  ---------- */
startTooBig:
   WTO.TEXT = 'TLH916W Procedure' PARM1 'is intensive.',
   'Answer "Y" to continue'
   'WTOR'
   IF WTO.REPLY = 'Y' THEN
     'REISSUE MVS' fullCmd
   ELSE
    do;
     WTO.TEXT = 'TLH917I start' PARM1 'canceled.'
     'WTO'
     'REISSUE SUPPRESS'
    end;
RETURN            /* from startTooBig  */
```

Figure 56. CNMSRVMC Example (Part 2 of 3)

```
getNVassoc:
  username = RECEDATA('U')
  SELECT
   WHEN (username = '*BYPASS*') THEN
    'WTO TLH006E Please logon to your system console'
   WHEN (sysconid() = 'INSTREAM' | sysconid() = 'INTERNAL') THEN
    'WTO TLH026E Console' sysconid() 'cannot associate with NetView.'
   OTHERWISE
    do
     'AUTOTASK OPID=' || username || 'CONSOLE=' || sysconid()
     IF rc = 0 THEN
      'MSG' username 'association with NetView,' domain()', successful.'
     ELSE
      'WTO TLH662E Association with NetView failed:' RC
    end
  END
  'REISSUE SUPPRESS'
RETURN            /* from getNVassoc   */
BadEnvr:
  'MESSAGE DSI290 CNMSRVMC' opid()
  exit 12
```

Figure 56. CNMSRVMC Example (Part 3 of 3)

DSPRSTAT Example

Figure 57 on page 194 is an example of a command list that uses the same type of function as Figure 52 on page 174.

```
/*********************************************************************/
/*                                                                   */
/*  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*/
  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 57. DSPRSTAT Example

GETCG Example

```
/*************************************************************/
/* 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 58. GETCG Example

GREETING Example

```
/*********************************************************************/
/*                                                                   */
/*  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 OCCURRED WAITING FOR DATE COMMAND RESPONSE'
  WHEN (EVENT()='T') THEN           /* DID WE GET A TIME-OUT?        */
      SAY 'NO MESSAGE RETURNED FROM DATE COMMAND'
  OTHERWISE
END                                 /* OF SELECT                     */
EXIT
```

Figure 59. GREETING Example

LISTVAR Example

```
/* Use NetView HELP for more information about this clist.
*********************************************************************
*    Licensed Materials - Property of IBM                           *
*    5697-ENV © Copyright IBM Corp. 1997, 2007                      *
*    All rights reserved.                                           *
*                                                                   *
*    US Government Users Restricted Rights - Use, duplication or    *
*    disclosure restricted by GSA ADP Schedule Contract with IBM Corp. *
*********************************************************************
*                                                                   *
*    NAME(LISTVAR)  SAMPLE(CNME1006)  RELATED-TO()                  *
*                                                                   *
*    CNME1006 Change Activity:                                      *
*   ID=Reference,Ver,Date,Developer:  Description                   *
*   ------------------------------------------------------          *
*                                                                   *
*********************************************************************/

IF arg(1) ¬= '' THEN SIGNAL ERROR
address NETVASIS
mySys = left(opsystem(),3)        /* get e.g. "MVS"                  *

call say353 'OPSYSTEM' opsystem()
SELECT;
  WHEN (mySys = 'VSE')  THEN
     call say353 'CURPART' curpart()     /* Partition ID   */
  WHEN (mySys = 'MVS')  THEN
    DO;
     call say353 'MVSLEVEL' mvsLevel()   /* MVS level      */
     call say353 'ECVTPSEQ' ecvtpseq()   /* ECVTPSEQ value*/
     call say353 'CURSYS' curSys()       /* System name    */
    END;
  WHEN (mySys = 'VM/') THEN
     NOP
  OTHERWISE
     'MESSAGE DWO053I'
 END;
call say353 'VTAMLVL' vtam()             /* VTAM version          */
call say353 'VTCOMPID' vtcompid()
call say353 'NetView' NETVIEW('T')
call say353 'NETID' netid()
call say353 'DOMAIN' domain()
call say353 'APPLID' applid()
call say353 'OPID' opid()
call say353 'LU' lu()
call say353 'TASK' task()
call say353 'NCCFCNT' nvcnt()
call say353 'HCOPY' hcopy()
 /* call say353 'TYPE' type()  there is only one type     */
call say353 'IPV6ENV' ipv6env()
```

Figure 60. LISTVAR example (Part 1 of 2)

```
towerList = tower('*')
'PIPE var towerList | edit word 1.*|var TowerList'
CALL say353 'TOWERS' towerList

 /* Display MVS console name (or ID) if in use.            */
IF mysys = 'MVS' THEN
 do
  call say353 'CURCONID' curconid()
  IF autotask() THEN
    call say353 'AUTCONID' autconid()
 end

IF VTAM() = '' THEN
  'MESSAGE CNM386I LISTVAR'
EXIT 0

/*------------------<<< Format a CNM353 Message >>>------------------*/
/*  Format a MESSAGE CNM353 with extra spaces to make the values     */
/*  line up in the same column.  Using 30, at present; it could      */
/*  get bigger if longer named values are needed.                    */
/*                                                                   */
/*------------------<<< Format a CNM353 Message >>>------------------*/
SAY353:
  parse arg name value
  'MESSAGE CNM353 LISTVAR' "'"left(name,8)"'" "'"value"'"
return
/*
CNM353I LISTVAR : OPSYSTEM = MVS/ESA
  ********************************************************************/
ERROR:
'MESSAGE CNM306E,LISTVAR,' arg(1)
exit 4
```

Figure 60. LISTVAR example (Part 2 of 2)

PRINT Example

Figure 61 on page 198 is an example of a command list used for printing a data set.

```
/*********************************************************************/
/* 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.                                      */
/*********************************************************************/
SIGNAL ON ERROR                          /* SIGNAL IF ERROR OCCURS*/
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     */
    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 PRINT CLIST IS NOW FINISHED'
          END                            /* PROCESS 2ND CNM272I   */
      END                                /* PROCESS 1ST CNM272I   */
  END                                    /* NAME WAS SPECIFIED    */

RETURN                                   /* RETURN TO CALLER/EXIT */
ERROR: SAY 'ERROR OCCURRED.  RETURN CODE IS  ' RC
EXIT -1;                                /* END COMMAND LIST FOR ERROR*/
```

Figure 61. PRINT Example

TYPE Example

Figure 62 is an example of a command list used to display the members of a data set.

```
/*********************************************************************/
/* 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.  */
/*********************************************************************/
SIGNAL ON ERROR                              /* SIGNAL IF ERROR OCCURS*/
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 */
ERROR: SAY 'ERROR OCCURRED.  RETURN CODE IS  ' RC
EXIT -1;                                  /* END COMMAND LIST FOR ERROR*/
```

Figure 62. TYPE Example

TYPEIT Example

Figure 63 on page 200 is an example of a command list that does essentially the same thing as the example in Figure 62, but closes the data set in case of error.

```
/*********************************************************************/
/* 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. */
/*********************************************************************/
signal on error
rc = 0
parse arg datasetname '(' member ')' extra
if datasetname = '' | parmcnt() > 1 | member = '' | extra ¬= '' then
  do
    say 'Incorrect syntax used.'
    say 'Correct syntax is :  '
    say '      TYPE dataset.name(member)  '
    rc = 24
  end
else
  do
    'trap and suppress only messages CNM272I'
    signal on halt
    'allocate da('datasetname'('member')) shr'
    'wait 5 seconds for messages'
    'msgread'
    if msgid() = 'CNM272I' then
      do
        ddname = msgvar(1)
        if fndmbr(ddname,member) ¬= 0 then
          do
            say 'Member' member 'does not exist.'
            rc = 4
          end
        else
          do
            address mvs 'execio * diskr' ddname '(finis'
            do while queued() ¬= 0
              pull record
              say substr(record,1,68)
            end
            signal off halt
            'free file('ddname')'
            'trap no messages'
            'message 309I ''TYPEIT'',''clist is now finished.'''
          end
      end
  end
return rc
halt:
'free file('ddname')'
'trap no messages'
return -5
error: say 'Error occurred. Return code is' rc
return -1
```

Figure 63. TYPEIT Example

UPDCGLOB Example

Figure 64 is an example of updating a common global variable reentrantly.

```
/*REXX*---------------------------------------------------------------*/
/*  Licensed Materials - Property of IBM                             */
/*  5697-ENV © Copyright IBM Corp. 2009                              */
/*  All rights reserved.                                             */
/*                                                                   */
/*  US Government Users Restricted Rights - Use, duplication or      */
/*  disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*-------------------------------------------------------------------*/
/* ----------->
Sample NETVONLY action command.  This command gets control as a result
of a Command Revision action:  NETVONLY=CNMSRVMC

It is designed to respond to a pseduo-command "V NetView,LOGON" issued
from a regular MVS console.  Or to a START command for a special
procedure that should not be run during normal working hours.

To see what triggers this command, read sample CNMSCRT1.
Customers can code any number of different REXX procedures to be
invoked by the NETVONLY action.  This sample is only a model.

Use any REXX function here, but the message based functions that return
meaningful data are only sysconid() and sysid(). Use the recedata()
function to obtain information about the origin of the command.
   <---------- */
ADDRESS NETVASIS
/* important!  prevent abuse by checking for proper environment      */
callerASID = RECEDATA()            /* Obtain ASID of cmd issuer      */
IF callerASID = 0 THEN             /* no RECE data?  Then call must  */
  SIGNAL BadEnvr                   /* not be from NETVONLY action.   */

/* The "command" that triggered NETVONLY is our arg...               */
fullCmd = arg(1)
parse arg verb Parm1 ',' theRest
/* We handle two verbs:  'S' or 'START' and 'V' or 'VARY'.  Consider
   writting two procedures for simplicity.
   Both verbs come here ONLY when other conditions in the revision
   table are satisfied. See CNMSCRT1.                              */
SELECT
  WHEN (LEFT(verb,1) = 'S') THEN    /* It was a START cmd           */
     CALL startTooBig
  WHEN (LEFT(verb,1) = 'V') THEN    /* VARY cmd with NetView arg    */
     CALL getNVassoc
  OTHERWISE           /* Should Not Occur unless CNMSCRT1 changed.  */
   do;
     msg = 'RECE command CNMSRVMC driven by system command "' ||,
           arg(1) || '". Please review CRT.'
     'pipe VAR msg | ROUTE AUTRCVR'
     /* We purposely do not REISSUE here. Operator will get CNM017E */
   end;
END
```

Figure 64. UPDCGLOB Example (Part 1 of 2)

```
   exit 0
/* Ask the operator to confirm.  Wait for response to the WTOR.
   Note: Waiting for the response here DOES NOT block other
         NETVONLY actions from proceeding.
   Note: if confirmed, this procedure issues the exact same command
         that caused this NETVONLY action.  This will not start a loop
         and the reissued command will appear to have originated in the
         same environment as the original command (username &
         console).
  ---------- */
startTooBig:
   WTO.TEXT = 'TLH916W Procedure' PARM1 'is intensive.',
   'Answer "Y" to continue'
   'WTOR'
   IF WTO.REPLY = 'Y' THEN
     'REISSUE MVS' fullCmd
   ELSE
    do;
     WTO.TEXT = 'TLH917I start' PARM1 'canceled.'
     'WTO'
     'REISSUE SUPPRESS'
    end;
RETURN            /* from startTooBig  */
getNVassoc:
  username = RECEDATA('U')
  SELECT
   WHEN (username = '*BYPASS*') THEN
    'WTO TLH006E Please logon to your system console'
   WHEN (sysconid() = 'INSTREAM' | sysconid() = 'INTERNAL') THEN
    'WTO TLH026E Console' sysconid() 'cannot associate with NetView.'
   OTHERWISE
    do
     'AUTOTASK OPID=' || username || 'CONSOLE=' || sysconid()
     IF rc = 0 THEN
      'MSG' username 'association with NetView,' domain()', successful.'
     ELSE
      'WTO TLH662E Association with NetView failed:' RC
    end
  END
  'REISSUE SUPPRESS'
RETURN            /* from getNVassoc   */
BadEnvr:
  'MESSAGE DSI290 CNMSRVMC' opid()
  exit 12
```

Figure 64. UPDCGLOB Example (Part 2 of 2)

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Programming Interfaces

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Tivoli NetView for z/OS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml .

Adobe is a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Index

Special characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Product Number: 5697-NV6

Printed in USA

SC27-2861-01