Tivoli® NetView® for z/OS™

Customization: Using REXX and the NetView Command List Language

Version 5 Release 1

SC31-8862-00

Tivoli® NetView® for z/OS™

Customization: Using REXX and the NetView Command List Language

Version 5 Release 1

SC31-8862-00

Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language

Programming Interfaces

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of Tivoli NetView for z/OS.

Contents

Preface . . . vii
Who Should Read This Document . . . vii
What This Document Contains . . . vii
Publications . . . viii
  Prerequisite and Related Documents . . . viii
  Accessing Publications Online . . . viii
  Ordering Publications . . . viii
  Providing Feedback about Publications . . . ix
Contacting Customer Support . . . ix
Accessibility Information . . . ix
  Keyboard Access . . . ix
Conventions Used in This Document . . . ix
Platform-specific Information . . . x
Terminology . . . x
Reading Syntax Diagrams . . . xi
  Required Syntax . . . xi
  Optional Keywords and Variables . . . xii
  Default Values . . . xii
  Long Syntax Diagrams . . . xii
  Syntax Fragments . . . xii
  Commas and Parentheses . . . xiii
  Highlighting, Brackets, and Braces . . . xiv
  Abbreviations . . . xv

Part 1. Basic Command List Topics . . . 1

Chapter 1. Getting Started . . . 3
The Benefits of Using Command Lists . . . 3
  Examples of Common Startup Command Lists . . . 4
  Examples of Activating a Network Control Program . . . 4
Creating Command Lists . . . 5
Controlling Access to Command Lists . . . 6
Loading Command Lists into Storage . . . 7
Running Command Lists . . . 8
  Running Command Lists When NetView Is Started . . . 8
  Running Command Lists When Logging On . . . 8
  Running Command Lists after Receiving a Message or MSU . . . 9
  Running Command Lists from a Terminal . . . 9
  Running Command Lists at a Specified Time or Time Interval . . . 9
  Running Command Lists from Another Command List . . . 10
  Passing Information from One Command List to Another . . . 11
  Running Command Lists from a User-Written Command Processor . . . 11
Using Network Commands in Command Lists . . . 11
  Using System Commands . . . 12
  Using Long-Running Commands . . . 12
  Using NetView Pipelines . . . 14
  Using the VIEW Command . . . 14
  Using Full-Screen Commands . . . 14
  Primary POI Task Restrictions . . . 15
  AUTOTASK OST Restrictions . . . 16
Controlling Command List Output . . . 16

Part 2. Writing Command Lists in REXX Language . . . 17

Chapter 2. REXX Language Overview 19
Introduction to the REXX Language 19
Compiling and Executing REXX Command Lists 20
Using Data REXX 20
/*%DATA 21
/*%LOGIC 22
Coding Conventions for REXX Command Lists and Data REXX Files 23
Record Size 23
Using Quotation Marks 23
Suppressing Display of Non-REXX Commands 25
NetView Restrictions on REXX Instructions. 26
Pausing for Operator Input 26
Using the SAY Instruction 26
Using the CALL Instruction 27
NetView Restrictions on REXX Functions 27
Writing REXX Function Packages 28
Changing the Environment Addressed by REXX Command Lists 28
Data REXX Host Command Environment 29
Using the EXECIO Command 29
Using MVS and VTAM Commands 30
Using the NetView ALLOCATE and FREE Commands 30
Using REXX Command Lists 31
Nesting REXX Command Lists from Assembler, C, or PL/I 32
Parsing in REXX Command Lists 32
Tracing REXX Command Lists 32
Return Codes in REXX Command Lists 33
Recovering from Errors in REXX Command Lists. 34

Chapter 3. REXX Instructions Provided by NetView 37
Using TRAP in Nested REXX Command Lists 38
Using WAIT in Nested Command Lists 38
Using MSGREAD in Nested Command Lists 39
Functions Set by MSGREAD. 39

Part 3. Writing Command Lists in the NetView Command List Language 41

Chapter 4. Writing Simple Command Lists in the NetView Command List Language . . 43
What the NetView Command List Language Includes 43
Coding Conventions for NetView Command List Language Statements 43
Conventions for General Coding 44
Conventions for Continuing a Statement. 44
Conventions for Double-Byte Character Set Text 45
Conventions for Suppression Characters. 46
Labels 46
Variables 47
Variable Substitution Order 47
Parameter Variables 48
Passing Parameter Variable Information to a Command List 49
Using Parameter Variables in a Command List 50
Passing Parameter Variables to a Nested Command List 50
Using Quoted Strings or Special Characters in Parameter Variables 51
Null Parameter Values. 51
Control Variables 52
User Variables 52
Hexadecimal Notation. 53
Comments. 54
Null Statements 54
Assignment Statements 54
Control Statements 56
&CONTROL Statement 56

Writing to the Operator . . . 57
Using NetView Commands with &PAUSE . . . 62
An Example Using &PAUSE. . . . 63
NetView Built-in Functions . . . 63
&BITAND . . . 64
&BITOR . . . 64
&BITXOR . . . 65
&CONCAT . . . 66
&HIER . . . 66
&LENGTH . . . 69
&MSUSEG . . . 70
&NCCFID . . . 71
&NCCFSTAT . . . 72
&SUBSTR . . . 73

Chapter 5. NetView Command List Language Branching . . . 77
&IF Control Statement. . . . 77
&GOTO Control Statement . . . 79
&EXIT Control Statement. . . . 79
&WAIT Control Statement . . . 81
Coding an &WAIT Control Statement . . . 82
Using NetView Commands with &WAIT . . . 86
Control and Parameter Variables Used with &WAIT. . . . 87
Using &WAIT in Nested Command Lists . . . 88
Customizing the &WAIT Statement . . . 89
Suggestions for Coding &WAIT . . . 91
Sample Using &WAIT . . . 92

Chapter 6. NetView Command List Language Global Variables. . . . 95
Using &TGLOBAL and &CGLOBAL . . . 96
&TGLOBAL . . . 96
&CGLOBAL . . . 97
Updating Task Global Variables Using &TGLOBAL . . . 98
Extent of Variables When Using &TGLOBAL and &CGLOBAL . . . 99
GLOBVAR1 Example . . . 100
GLOBALV Command . . . 102

Part 4. Advanced Command List Topics . . . 103

Chapter 7. Automation Resource Management . . . 105
Defining NetView Automation Table Command Lists . . . 105
Routing Messages from Automation-Table-Driven Command Lists . . . 105
Implementing NetView Automation . . . 105
Suppressing Messages . . . 106
Determining What Task Controls a Command List . . . 106
Testing Automation Command Lists. . . . 106
Recovering from Looping Command Lists. . . . 107
Considering Operator Interaction. . . . 107
Common Automation Problems . . . 107

Chapter 8. Common Operations Services Commands . . . 109
Common Operations Service . . . 109
Common Operations Services Return Codes . . . 110
LINKDATA and LINKTEST Results . . . 110
LINKDATA and LINKTEST Variables . . . 110
LINKTEST Additional Variables . . . 111
LINKPD Results . . . 111
RUNCMD Results. . . . 112
Using RUNCMD in a Pipeline. . . . 112

Part 5. Commands, Functions, and Variables 115

Chapter 9. REXX Functions Provided by NetView 117
Translation Functions 118
Command List Information. 120
Cross-Domain Information Functions 125
Data Set Information Functions 126
Global Variable Information Functions 128
Message Processing Information Functions 128
MVS-Specific Message Processing Information 135
ROUTCDE Examples 142
REXX Management Services Units (MSU) Information Functions 143
Hardware Monitor (HMxxxxxx) Examples 152
MSUSEG Syntax and Examples 156
Operator Information Functions 158
Session Information Functions 158
REXX Environment Information Functions 163
Terminal Information Functions 163
Time and Date 164
Nulls and Blanks Stripping 164

Part 6. Appendixes 167

Appendix A. Comparison of REXX and NetView Command List Language 169
Comparison of REXX Instructions and NetView Command List Language Control Statements 169
Comparison of REXX Functions and NetView Command List Language Control Variables and Functions . . . 170
Commands Used in Command Lists 173

Appendix B. Command List Examples Index 175
REXX Command List Examples 175
NetView Command List Language Examples. 175

Appendix C. Examples of REXX Command Lists for NetView 177
ACTAPPLS Example 177
ACTLU Example 179
CHKOPNUM Example 179
CHKRSTAT Example 181
DSPRSTAT Example 183
GETCG Example 184
GREETING Example 185
LISTVAR Example. 186
PRINT Example 186
TYPE Example 188
TYPEIT Example 188

Index 191

Preface

This document describes how to write command lists for the Tivoli® NetView® for z/OS™ product using either the Restructured Extended Executor language (REXX) or the NetView command list language.

Who Should Read This Document

System programmers and network operators, who use command lists or write command lists, can find helpful information in this document. You should be familiar with how the NetView program is used in your network and what the operator's tasks are. This document does not provide descriptions of NetView operator commands. If a command is not familiar, refer to the NetView online help.

What This Document Contains

This document is organized into the following sections:

"Part 1. Basic Command List Topics" on page 1 contains an overview of basic command list topics that are common to command lists written in either REXX or the NetView command list language.

"Part 2. Writing Command Lists in REXX Language" on page 17 describes how to write command lists using REXX.

"Part 3. Writing Command Lists in the NetView Command List Language" on page 41 describes how to write command lists using the NetView command list language.

"Part 4. Advanced Command List Topics" on page 103 describes advanced topics that pertain to command lists written in either REXX or the NetView command list language.

"Part 5. Commands, Functions, and Variables" on page 115 provides descriptions of all control keywords, control variables, and functions for the NetView command list language and for REXX functions provided by the NetView program.

"Appendix A. Comparison of REXX and NetView Command List Language" on page 169 provides a comparative list of commands, variables, and functions common to REXX and NetView command list language, and shows the page numbers where each is described.

"Appendix B. Command List Examples Index" on page 175 contains reference tables for the REXX and NetView command list examples contained in this document.

"Appendix C. Examples of REXX Command Lists for NetView" on page 177 contains examples of REXX command lists written for the NetView program.

Publications

This section lists prerequisite and related documents. It also describes how to access Tivoli publications online, how to order Tivoli publications, and how to make comments on Tivoli publications.

Prerequisite and Related Documents

To read about the new functions offered in this release, refer to the *Tivoli NetView for z/OS Installation: Migration Guide*.

You can find additional product information on these Internet sites:

Table 1. Resource Web sites

IBM®	http://www.ibm.com/
Tivoli Systems	http://www.tivoli.com/
Tivoli NetView for z/OS	http://www.tivoli.com/nv390

The Tivoli NetView for z/OS Web site offers demonstrations of the NetView product, related products, and several free NetView applications you can download. These applications can help you with tasks such as:

- Getting statistics for your automation table and merging the statistics with a listing of the automation table
- Displaying the status of a JES job or cancelling 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 TSO commands and receiving responses

Accessing Publications Online

You can access many Tivoli publications online using the Tivoli Information Center, which is available on the Tivoli Customer Support Web site:

http://www.tivoli.com/support/documents/

These publications are available in PDF format. Translated documents are also available for some products.

Ordering Publications

You can order many Tivoli publications online at the following Web site:

http://www.ibm.com/shop/publications/order

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, for a list of telephone numbers, see the following Web site: http://www.tivoli.com/inside/store/lit_order.html

Providing Feedback about Publications

We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:

- Send an e-mail to pubs@tivoli.com.
- Complete our customer feedback survey at the following Web site:

  http://www.tivoli.com/support/survey/

Contacting Customer Support

If you have a problem with any Tivoli product, you can contact Tivoli Customer Support. See the *Tivoli Customer Support Handbook* at the following Web site:

http://www.tivoli.com/support/handbook/

The handbook provides information about how to contact Tivoli Customer Support, depending on the severity of your problem, and the following information:

- Registration and eligibility
- Telephone numbers and e-mail addresses, depending on the country you are in
- What information you should gather before contacting support

Note: Additional support for Tivoli NetView for z/OS is available at the NetView for z/OS Web site:

http://www.tivoli.com/nv390

Under Related Documents, select **Other Online Sources.**

The page displayed contains a list of newsgroups, forums, and bulletin boards.

Accessibility Information

Refer to *Tivoli NetView for z/OS User's Guide* for information about accessibility.

Keyboard Access

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.

Refer to *Tivoli NetView for z/OS User's Guide* for more information about keyboard access.

Conventions Used in This Document

The document uses several typeface conventions for special terms and actions. These conventions have the following meaning:

Bold	Commands, keywords, flags, and other information that you must use literally appear like **this**, in **bold**.

Italics	Variables and new terms appear like *this*, in *italics*. Words and phrases that are emphasized also appear like *this*, in *italics*.
`Monospace`	Code examples, output, and system messages appear like `this`, in a `monospace` font.
ALL CAPS	Tivoli NetView for z/OS commands are in ALL CAPITAL letters.

Platform-specific Information

For more information about the hardware and software requirements for NetView components, refer to the *Tivoli NetView for z/OS Licensed Program Specification.*

Terminology

For a list of Tivoli NetView for z/OS terms and definitions, refer to http://www.networking.ibm.com/nsg/nsgmain.htm.

For brevity and readability, the following terms are used in this document:

NetView

- Tivoli NetView for z/OS Version 5 Release 1
- Tivoli NetView for OS/390® Version 1 Release 4
- Tivoli NetView for OS/390 Version 1 Release 3
- TME 10™ NetView for OS/390 Version 1 Release 2
- TME 10 NetView for OS/390 Version 1 Release 1
- IBM NetView for MVS Version 3
- IBM NetView for MVS Version 2 Release 4
- IBM NetView Version 2 Release 3

MVS OS/390, or z/OS operating systems.

RACF®

RACF is a component of the SecureWay® Security Server for z/OS and OS/390, providing the functions of authentication and access control for OS/390 and z/OS resources and data, including the ability to control access to DB2® objects using RACF profiles. Refer to:

http://www-1.ibm.com/servers/eserver/zseries/zos/security/racfss.html

Tivoli Enterprise™ software

Tivoli software that manages large business networks.

Tivoli environment

The Tivoli applications, based upon the Tivoli Management Framework, that are installed at a specific customer location and that address network computing management issues across many platforms. In a Tivoli environment, a system administrator can distribute software, manage user configurations, change access privileges, automate operations, monitor resources, and schedule jobs. You may have used TME 10 environment in the past.

TME 10

In most product names, TME 10 has been changed to Tivoli.

V and R

Specifies the version and release.

VTAM® and TCP/IP
VTAM and TCP/IP are included in the IBM Communications Server element of the OS/390 and z/OS operating systems. Refer to http://www.ibm.com/software/network/commserver/about/.

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.

Reading Syntax Diagrams

Syntax diagrams start with double arrowheads on the left (►►) and move along the main line until they end with two arrowheads facing each other (►◄).

As shown in the following table, syntax diagrams use ***position*** to indicate the required, optional, and default values for keywords, variables, and operands.

Table 2. How the Position of Syntax Diagram Elements Is Used

Element Position	**Meaning**
On the command line	Required
Above the command line	Default
Below the command line	Optional

Required Syntax

The command name, required keywords, variables, and operands are always on the main syntax line. Figure 1 specifies that the *resname* variable must be used for the CCPLOADF command.

CCPLOADF

```
►►──CCPLOADF resname──────────────────────────────────────────────────►◄
```

Figure 1. Required Syntax Elements

Keywords and operands are written in uppercase letters. Lowercase letters indicate variables such as values or names that you supply. In Figure 2, MEMBER is an operand and *membername* is a variable that defines the name of the data set member for that operand.

TRANSMSG

```
►►──TRANSMSG MEMBER=membername────────────────────────────────────────►◄
```

Figure 2. Syntax for Variables

Optional Keywords and Variables

Optional keywords, variables, and operands are below the main syntax line. Figure 3 specifies that the ID operand can be used for the DISPREG command, but is not required.

DISPREG

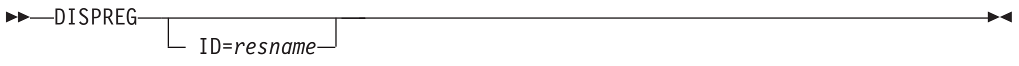


Figure 3. Optional Syntax Elements

Default Values

Default values are above the main syntax line. If the default is a keyword, it appears only above the main line. You can specify this keyword or allow it to default.

If an operand has a default value, the operand appears both above and below the main line. A value below the main line indicates that if you choose to specify the operand, you must also specify either the default value or another value shown. If you do not specify an operand, the default value above the main line is used.

Figure 4 shows the default keyword STEP above the main line and the rest of the optional keywords below the main line. It also shows the default values for operands `MODNAME=*` and `OPTION=*` above and below the main line.

RID

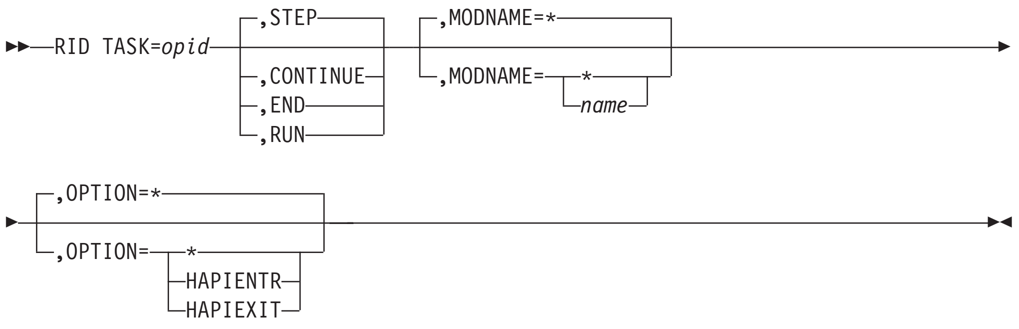


Figure 4. Sample of Defaults Syntax

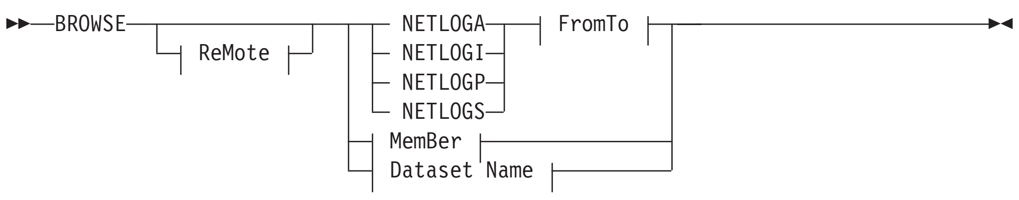
Long Syntax Diagrams

When more than one line is needed for a syntax diagram, the continued lines end with a single arrowhead (►). The following lines begin with a single arrowhead (►), as shown in Figure 4.

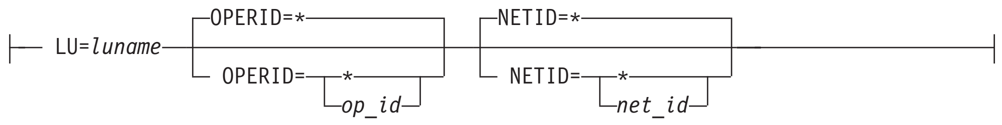
Syntax Fragments

Commands that contain lengthy groups or a section that is used more than once in a command are shown as separate fragments following the main diagram. The fragment name is shown in mixed case. See Figure 5 on page xiii for a syntax with the fragments `ReMote` and `FromTo`.

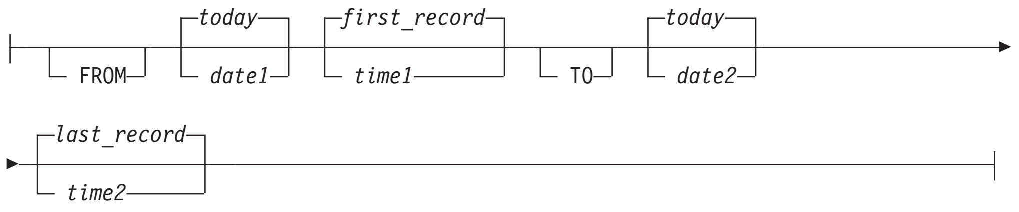
BROWSE



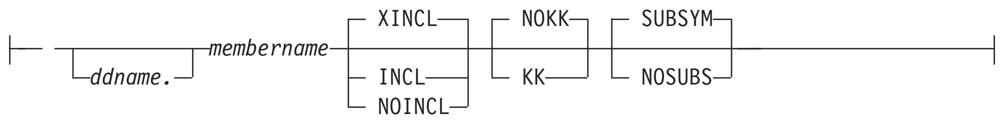
ReMote:



FromTo:



MemBer:



Dataset Name:

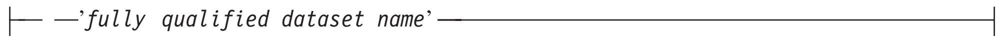


Figure 5. Sample Syntax Diagram with Fragments

Commas and Parentheses

Required commas and parentheses are included in the syntax diagram. When an operand has more than one value, the values are typically enclosed in parentheses and separated by commas. In Figure 6 on page xiv, the OP operand, for example, contains commas to indicate that you can specify multiple values for the *testop* variable.

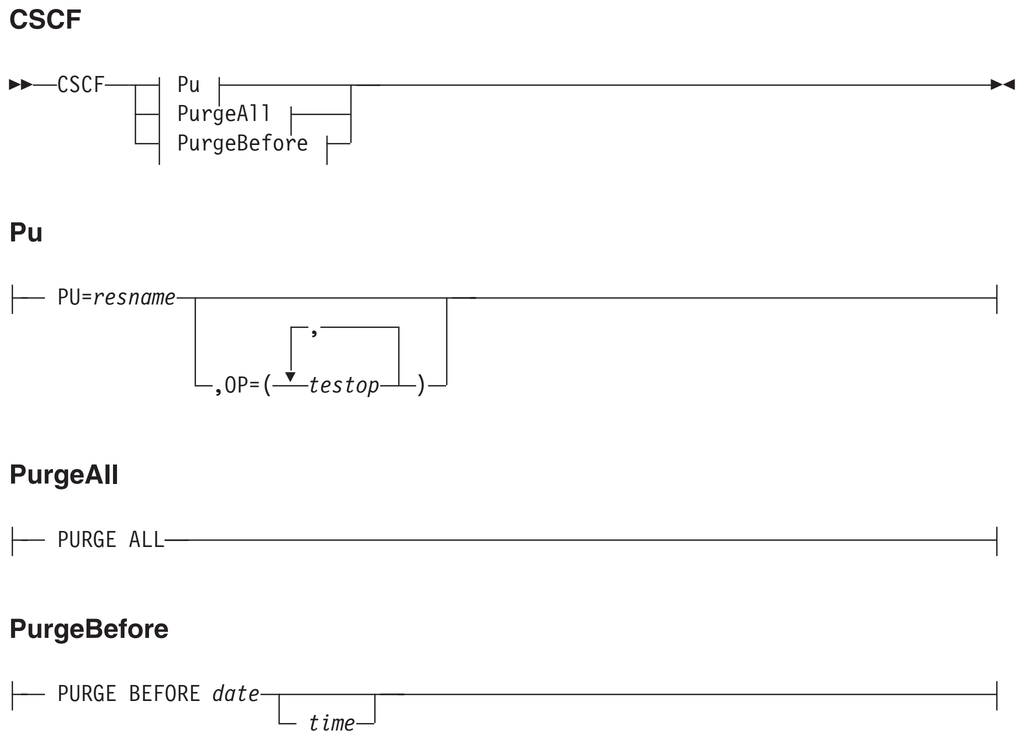


Figure 6. Sample Syntax Diagram with Commas

If a command requires positional commas to separate keywords and variables, the commas are shown before the keyword or variable, as in Figure 4 on page xii.

For example, to specify the BOSESS command with the *sessid* variable, enter:

```
NCCF BOSESS applid,,sessid
```

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

Highlighting, Brackets, and Braces

Syntax diagrams do not rely on highlighting, underscoring, brackets, or braces; variables are shown italicized in hardcopy or in a differentiating color for NetView help and BookManager® online books.

In parameter descriptions, the appearance of syntax elements in a diagram immediately tells you the type of element. See Table 3 for the appearance of syntax elements.

Table 3. Syntax Elements Examples

This element...	Looks like this...
Keyword	CCPLOADF
Variable	*resname*
Operand	MEMBER=*membername*
Default	<u>*today*</u> or INCL

Abbreviations

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

Part 1. Basic Command List Topics

Chapter 1. Getting Started 3
The Benefits of Using Command Lists 3
Examples of Common Startup Command Lists . . 4
Examples of Activating a Network Control Program 4
Creating Command Lists 5
Controlling Access to Command Lists 6
Loading Command Lists into Storage 7
Running Command Lists 8
Running Command Lists When NetView Is Started 8
Running Command Lists When Logging On. . . 8
Running Command Lists after Receiving a Message or MSU 9
Running Command Lists from a Terminal . . . 9
Running Command Lists at a Specified Time or Time Interval 9
Running Command Lists from Another Command List 10
Passing Information from One Command List to Another 11
Error Handling 11
Running Command Lists from a User-Written Command Processor 11
Using Network Commands in Command Lists . . 11
Using System Commands 12
Using Long-Running Commands 12
Using Major Long-Running Commands . . . 12
Using Minor Long-Running Commands . . . 13
Queuing Long-Running Commands 13
Using NetView Pipelines 14
Using the VIEW Command 14
Using Full-Screen Commands 14
Primary POI Task Restrictions 15
AUTOTASK OST Restrictions 16
Controlling Command List Output 16

Chapter 1. Getting Started

Tivoli NetView for z/OS (NetView) enables you 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 NetView program, a command list can be written in either Restructured Extended Executor language (REXX) or the NetView command list language

When you type a command list name at a terminal, the commands and instructions in that command list are interpreted and executed. 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 8 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 improve an operator's efficiency. Command lists obtain information from operators, other tasks, system resources, or the contents of messages. The command list uses this information to perform processing or to decide the next action. This flexibility enables you to 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 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 take action 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's screen
- Ask the operator questions and take action based on the answers
- Simplify 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 operator’s role 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.

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 7, 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 7. STARTUP1 Command List

Figure 8, written in the NetView command list language, establishes the same TAF sessions.

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

Figure 8. STARTUP2 Command List

Instead of having to remember and enter three commands, operators can enter the command list name STARTUP1 or STARTUP2. The command list starts the three sessions and operators receive the same messages they would receive if they had entered 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 9 is an example of a REXX command list that activates an NCP.

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

Figure 9. Example of Activating an NCP Using REXX

Figure 10 on page 5 shows a NetView command list language example that activates the same NCP:

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

Figure 10. Example of Activating an NCP Using NetView Command List Language

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 command list name unless you define another name for the command list on a CMDSYN statement. For more information about CMDSYN, refer to the *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. Valid characters are: 0–9 A–Z @ $ #.

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 NetView-provided command lists.

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

1. Create the data set that will 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 11.

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

Figure 11. 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.

Notes:

1. When the NetView program is operating on an MVS system and you plan to update or create command lists while the NetView program 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 may 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 invoke the command list is made, the command list will then be available.

   Recycle NetView if the data set becomes full and you need to compress the data set to add more command lists.
2. 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.

To create NetView command lists enter the following statement as the first statement in the command list:

```
label CLIST
```

The NetView program ignores the label. If present, the label needs to start in column 1. Be sure that the CLIST statement begins in column 2 or later, and is preceded by at least one blank.

Controlling Access to Command Lists

You can use command authorization to specify the operators who 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 CNMSTYLE and can
| be changed dynamically using the NetView REFRESH command.

Command lists are different from other types of commands because specified keywords and values when invoking the command list cannot be protected with command authorization. Verify keywords and values that are entered with a command list in the following ways:
- 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.

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.

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

Loading Command Lists into Storage

The NetView program enables you to load command lists into main storage before execution of the command list. Although it is not mandatory that you load a command list into main storage before it is executed, preloading promotes improved performance of your computer system.

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

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

LOADCL
Loads command lists into main storage shared by all operators

DROPCL
Drops a command list that was previously loaded into main storage using the LOADCL command

MAPCL
Lists command lists that currently reside in main storage

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

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

If you have command lists that are frequently executed, load them into main storage when the NetView program is initialized. For example, if the command lists STARTJOB and SETTERM are run often, you can load them into main storage by coding the statement in your initialization command list as follows:

```
LOADCL STARTJOB,SETTERM
```

After initialization, the STARTJOB and SETTERM command lists reside in main storage and are available for execution. For more information about loading command lists into storage when the NetView program is initialized, see “Running Command Lists When NetView Is Started” on page 8.

Running Command Lists

Design command lists that run with little assistance from operators. Some of the ways you can run command lists are:
- When the NetView 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

Running Command Lists When NetView Is Started

You can specify that command procedures run automatically when the NetView program is started by defining them in CNMSTYLE as an auxInitCmd. Refer to sample CNMSTYLE for more information. These commands run under the primary program operator interface task (PPT). See “Primary POI Task Restrictions” on page 15 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 10 for rules that apply when calling another command list.

Code the name of the command list you want to run in the operator’s profile 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 operator’s profile as follows:

```
PROFBEG  PROFILE  IC=HELLO
```

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

Note: Some operator IDs, known as autotasks, are started by using the AUTOTASK command. Tasks started by the AUTOTASK command do not have a terminal attached. Therefore, if an initial command list is to run after the autotask initializes, the initial command list cannot set any function keys or invoke any NetView full-screen panels.

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 program's 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 NetView 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. See "Chapter 7. Automation Resource Management" on page 105 for more information.

Running Command Lists from a Terminal

You can enter a command listname from the terminal in the same way 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.

NetView operators can activate, stop, suspend, or restart command list processing by entering the NetView commands GO, RESET, STACK, or UNSTACK.

For command lists written in REXX, the commands are entered when the command list is waiting for a response to a PARSE EXTERNAL, PULL, PARSE PULL, or WAIT instruction.

Note: PARSE EXTERNAL is supported by TSO/E. It is not supported by Systems Application Architecture®.

For command lists written in the NetView command list language, the commands are entered during command list &PAUSE processing or command list &WAIT processing.

The GO command must precede any data entered in response to a PARSE EXTERNAL, PARSE, PARSE PULL, or &PAUSE. For more information about the GO, RESET, STACK, and UNSTACK commands, refer to the NetView online help.

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 NetView program to run the command list after a specified period of time.

AT
Instructs the NetView program to run the command list at a particular time.

CHRON
Instructs the NetView program to run NetView commands at timed intervals.

EVERY
Instructs the NetView program to run the command list repeatedly at a certain time interval.

Note: 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. This is done with the PPT operand or by issuing the timer command under an autotask. However, some commands cannot be used in a command list running under the PPT. See “Primary POI Task Restrictions” on page 15 for more information.

You can define command lists so that they always interrupt the processing of other command lists. You do this using the TYPE=B (for Both) operand of the CMDMDL statement, or by issuing the timer command under an autotask. For more information about how to code CMDMDL statements, refer to the *Tivoli NetView for z/OS Administration Reference* book.

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 the NetView program reaches a statement with the name of a command list, it starts running the nested command list. When the NetView program reaches the end of the nested command list, it returns control to the calling command list and proceeds to the next statement, as shown in Figure 12 on page 10.

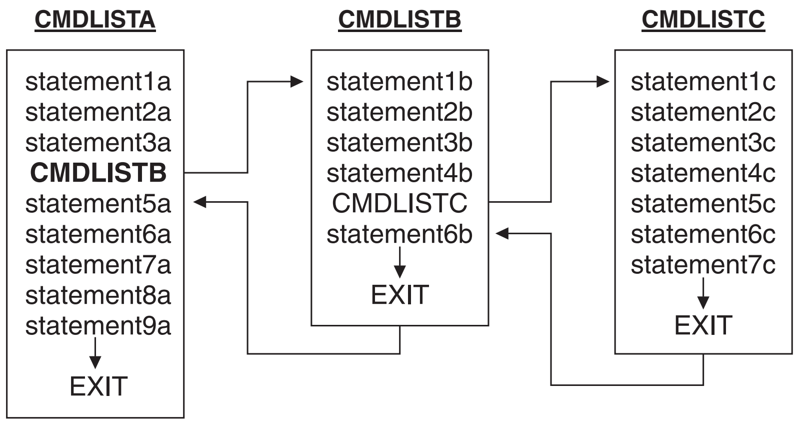


Figure 12. Nested Command Lists

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

- A REXX command list can be invoked as a REXX command, subroutine, or function.
- A REXX command list can call a command list written in the NetView command list language as a command but not as a subroutine or a function.
- A command list written in the NetView command list language can call another command list written in the NetView command list language or a REXX command list as a command.
- 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 invoked as commands, external subroutines, or external functions count as one of the 250 levels of externally nested command lists. You can invoke up to 250 REXX command lists as internal subroutines and functions, but they do not count toward the 250 levels of externally nested command lists.

- You should test each command list 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 variables between the calling command list and the nested command list, use NetView global variables. The GLOBALV command in the *Tivoli NetView for z/OS Command Reference* provides information about setting and retrieving global variables in REXX command lists. For information about defining global variables in command lists written in the NetView command list language, see "Chapter 6. NetView Command List Language Global Variables" on page 95. Alternatively, REXX variables in the caller can be accessed by the PIPE VAR stage. For more information, refer to the NetView online help.

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.

If the calling command list is written in REXX, it might be able to take action to recover from the error passed to it from the nested command list. For information about coding REXX command lists that can recover from errors, see "Recovering from Errors in REXX Command Lists" on page 34.

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 invokes a command list. Command processors are programs written in assembler, PL/I, or C. For information about how to write command processors, refer to *Tivoli NetView for z/OS Customization: Using Assembler* or *Tivoli NetView for z/OS Customization: Using PL/I and C*.

Using Network Commands in Command Lists

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

Some of the types of network commands you can include are:
- NetView commands
- User-written NetView commands
- VTAM commands

The commands used within command lists are subject to command authorization, unless SEC=BY was specified on the CMDMDL statement or AUTOSEC=BYPASS is in effect. For more information, refer to *Tivoli NetView for z/OS Administration Reference*.

Notes:
1. The NetView RETURN command is not valid in a command list.
2. You can use only NetView and user-written commands that are defined on the CMDMDL statement as regular or both (TYPE=R or TYPE=B).
3. You must use the appropriate prefix:
   - NLDM for session monitor commands
   - NPDA for hardware monitor commands
   - STATMON for status monitor commands

Using System Commands

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 NetView online help for more information.

Using Long-Running Commands

You can use long-running commands in your command lists. There are two types of long-running commands:
- Major
- Minor

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 execution priority.

Using long-running commands in your command list enables other commands on the low-priority queue to begin executing.Refer to the NetView online help for more information about command priorities.

Using Major Long-Running Commands

With the exception of the BGNSESS (FLSCN) and NCCF commands, most long-running commands are major long-running commands. When a command is issued from a command procedure, the command completes execution before the command procedure resumes. The return code from the command is available when the next command procedure instruction is executed. Execution for certain long-running commands (for example, NLDM) can take an extended period of time (until the operator chooses to exit).

Note: An exception is provided by the DSIPUSH MINOR function, which provides for a command to resume after completion of an invoking command procedure. The invoking procedure can obtain a return code or messages to be trapped only from the initial execution phase of such a command, not from the resumption.

Among NetView-supplied commands, only BGNSESS (FLSCN) uses this MINOR option. When the issuing command list is complete, the minor long-running command resumes execution.

If a command list issues a major long-running command, and while the command is executing, the same major long-running command is entered, the first command is canceled. The major long-running command then passes control to the issuing command list. In such a case, be aware of the following:

- When the issuing command list is written in REXX, code SIGNAL ON HALT. If you do not code SIGNAL ON HALT, the operator sees inappropriate termination messages. Code EXIT -5, and do not generate any messages in the HALT subroutine. See “Recovering from Errors in REXX Command Lists” on page 34 for more information about coding SIGNAL ON HALT.
- When the issuing command list is written in the NetView command list language, the command list is also canceled.

You can also cancel the calling command list with the UNIQUE command. Refer to the *Tivoli NetView for z/OS Customization Guide* book for more information about UNIQUE. Refer to the *Tivoli NetView for z/OS Customization: Using Assembler* book for information about DSIPUSH.

Using Minor Long-Running Commands

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

Notes:

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

To define a user-written command as a minor long-running command, use the DSIPUSH macro. Refer to the *Tivoli NetView for z/OS Customization: Using Assembler* book for information about DSIPUSH.

Queuing Long-Running Commands

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

To ensure that TAF command output is displayed before the command list resumes processing, use `CMD HIGH BGNSESS FLSCN`. If the operator rolls from the current long-running command, the command list continues. If the long-running command is canceled, the cancelation is not passed back to the issuing command list. For more information about TAF, refer to the NetView online help.

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

Using NetView Pipelines

NetView 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 *Tivoli NetView for z/OS Customization: Using 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.

When the VIEW command is invoked with the INPUT option, the contents of the following variables are passed from the command list to the VIEW command:
- VIEWICROW
- VIEWICCOL

When the VIEW command is invoked with the INPUT option, the contents of the input fields on a panel and the following variables are returned to the invoking command list:
- VIEWAID
- VIEWCURROW
- VIEWCURCOL
- VIEWCOLS
- VIEWROWS

The VIEW command has several other commands and command lists associated with it:
- UPPER command
- UNIQUE command
- SHOWCODE command list

Refer to the *Tivoli NetView for z/OS Customization Guide* book 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 executed from a full-screen processor issues a full-screen command, the NetView 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 suspendsany 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 CMDMDL definition statement with ECHO=N for the command list. Refer to the *Tivoli NetView for z/OS Administration Reference* for information about coding a CMDMDL 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, command lists should be executed under an autotask. See "AUTOTASK OST Restrictions" on page 16 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 execution as a result of NetView automation.
- Coded on a CNMSTYLE definition statement to run when the NetView 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.)

The following restrictions 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
  - CLOSE
  - GO
  - INPUT
  - LOGOFF
  - MOVE
  - PIPE with the CONSOLE stage
  - RETRIEVE
  - ROUTE
  - SET
  - START
  - STOP
  - SUBMIT
  - SWITCH
  - VIEW
  - WTO

  - WTOR
- Do not use the following REXX instructions:
  - FLUSHQ
  - MSGREAD
  - PARSE EXTERNAL
  - PARSE PULL if there is nothing in the REXX data stack
  - PULL if there is nothing in the REXX data stack
  - TRAP
  - WAIT
- Do not use the following NetView command list language control statements:
  - &PAUSE
  - &WAIT
- Do not execute command processors that use the MVS operating system STIMER macro.
- Do not use the PIPE command with the CORRWAIT stage.

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

AUTOTASK OST Restrictions

The following restrictions apply for automation tasks that are started with the AUTOTASK command:
- Use the ATTACH command if you want to automate full screen commands.
- Do not use commands that use keyboard functions; for example, setting function keys

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

To control the amount of data displayed to the operator during the execution of a REXX command list, use the NetView PIPE CONSOLE command, TRAP instruction, or the suppression character (see “Suppressing Display of Non-REXX Commands” on page 25). Refer to the NetView online help and the REXX library for information about the TRAP instruction.

To control the amount of data displayed to the operator during the execution of a command list written in the NetView command list language, use the &CONTROL control statement (see “&CONTROL Statement” on page 56), the &WAIT SUPPRESS control statement (see “Customizing the &WAIT Statement” on page 89), or the suppression character (see “Conventions for Suppression Characters” on page 46).

The commands and messages displayed during execution of a command list appear in the message area of the NetView panel. 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 *Tivoli NetView for z/OS User’s Guide*.

Part 2. Writing Command Lists in REXX Language

Chapter 2. REXX Language Overview 19
Introduction to the REXX Language 19
Compiling and Executing REXX Command Lists . . 20
Using Data REXX 20
  /*%DATA 21
  /*%LOGIC 22
Coding Conventions for REXX Command Lists and Data REXX Files 23
  Record Size 23
  Using Quotation Marks 23
  Suppressing Display of Non-REXX Commands 25
NetView Restrictions on REXX Instructions 26
  Pausing for Operator Input 26
  Using the SAY Instruction 26
  Using the CALL Instruction 27
NetView Restrictions on REXX Functions 27
Writing REXX Function Packages 28
Changing the Environment Addressed by REXX Command Lists 28
Data REXX Host Command Environment 29
Using the EXECIO Command 29
Using MVS and VTAM Commands 30
Using the NetView ALLOCATE and FREE Commands 30
Using REXX Command Lists 31
Nesting REXX Command Lists from Assembler, C, or PL/I 32
Parsing in REXX Command Lists 32
Tracing REXX Command Lists 32
Return Codes in REXX Command Lists 33
Recovering from Errors in REXX Command Lists . . 34

Chapter 3. REXX Instructions Provided by NetView 37
Using TRAP in Nested REXX Command Lists . . . 38
Using WAIT in Nested Command Lists 38
Using MSGREAD in Nested Command Lists . . . 39
Functions Set by MSGREAD. 39

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 NetView program.

Notes:

1. The power and flexibility of the REXX programming language makes it easy to inadvertently write programs that have unpredictable results. Therefore, be sure to thoroughly test all NetView REXX command lists before putting them into production.
2. For REXX on MVS systems support, TSO/E Version 2 or a later version of TSO must be installed, but not necessarily active. For complete information about REXX, refer to the REXX library.

Introduction to the REXX Language

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

As with all non-NetView REXX execs and macros, 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 executed by using specific commands that alter the processing order.

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

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

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

The REXX language enables you 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. There are also several built-in functions 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 Executing 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 will 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 *Tivoli NetView for z/OS Tuning Guide*.

Notes:

1. You do not need to install or start the compiler on the system where NetView resides.
2. The compiled executable might be larger (take up more space) than the original uncompiled command list.
3. The NetView 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.V5R1M0.EAGPRC.
     - The load module can only be invoked through or by a REXX CALL instruction, or as a REXX function.
5. To execute CEXEC format compiled REXX command lists, place the output file into a member of one of the DSICLD data sets.
6. Install the compiler run-time library in an authorized library on the system that executes the compiled REXX command lists.

Using Data REXX

Data REXX enables you to include REXX instructions and functions in data files. Data REXX uses two formats: DATA and LOGIC. Use DATA for sections of the file that are primarily data and use LOGIC for sections of the files that are primarily logic. Sample DSICMD is an example of a DATA file and sample BNJMBDST is an example of a LOGIC file. CNMSTASK uses both formats.

The following information applies to both DATA and LOGIC files:

- Data REXX is supported only in files that are contained in the following libraries:

  - DSIPARM
  - CNMPNL1
  - DSIPRF
  - DSIVTAM
  - BNJPNL1
  - BNJPNL2
  - DSIOPEN
  - DSILIST
  - DSIMSG
  - DSIASRC
  - DSIARPT
- Data REXX is not supported in VSAM files.
- The first line of the file must begin with either the /*%DATA or /*%LOGIC NetView REXX directive.
- A file can contain both DATA and LOGIC NetView REXX directives.
- DATA REXX files should be small because they are read entirely into storage. Note that %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, %INCLUDE files that contain a large amount of data should not be data REXX files.
- REXX clauses, which are treated as external commands by REXX, are treated as external data by Data REXX.
- Strings output by the SAY and TRACE instructions are written to the network log.
- The REXX keyword instruction ADDRESS can be used; however, the only address that is supported is NETVDATA.
- 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 only be used 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.

/*%DATA

The /*%DATA NetView REXX directive starts a Data REXX file in data mode. The file remains in data mode until either the /*%LOGIC directive is encountered or the end of file (EOF) is reached.

The syntax for the /*%DATA NetView REXX directive is:

/*%DATA

```
►►──/*%DATA comments*/─────────────────────────────────────────────────►◄
```

Where:

/*%DATA
The first line of the file must begin with /*%DATA and the / must be the first character on the line. A space is required after DATA and the word DATA must be in all capital letters.

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

*/ Specifies the end of the DATA statement.

Usage Notes:

- In data mode, all REXX instructions must be preceded by %> (for example, %> ELSE). Lines that do not begin with %> in column one are treated as data.
- The % sign must be in column one.

Example: The following is an example of 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 /*%LOGIC NetView REXX directive starts a Data REXX file in logic mode. The file remains in logic mode until either the /*%DATA directive is encountered or the end of file (EOF) is reached.

The syntax for the /*%LOGIC NetView REXX directive is:

/*%LOGIC

```
►►──/*%LOGIC comments*/────────────────────────────────────────────────►◄
```

Where:

/*%LOGIC
The first line of the file must begin with /*%LOGIC and the / must be the first character on the line. A space is required after LOGIC and the word LOGIC must be in all capital letters.

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

*/
: Specifies the end of the LOGIC statement.

Usage Notes:

- REXX clauses, which are treated as external commands by REXX, are treated as external data by Data REXX.
- In logic mode, REXX instructions do not need to be preceded by %>.

Example: The following is an example of 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 NetView program.

Record Size

The data portion of records in REXX command lists or Data REXX files for the NetView program can be up to 80 characters in length (the records need to be a fixed length of 80 characters). If the first record of a REXX command list or Data REXX file contains a sequence number in columns 73 through 80, all records in that command list or Data REXX file are truncated to 72 characters. The NetView 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 quote (single or double) on a command list statement or Data REXX statement , it stops interpreting until it reaches a matching ending quote.

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 at your terminal when using REXX or writes 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 do the following:

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

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 NetView 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. REXX clauses, which are treated as external commands by REXX, 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'
```

However, if the command list contains a variable named NODE and you want to deactivate the node whose name is the current value of the NODE variable, 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's input into a variable called DDNAME. The TSO/E EXECIO command is then used to read a line of that DDNAME. ADDRESS MVS is a REXX instruction, so it is not enclosed in 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 NetView program truncates trailing blanks from all REXX command list records before executing the command list, REXX clauses that have quoted strings that span multiple records may not execute as

expected. For example, in the following set of REXX clauses that span records, The NetView program removes the blanks in the middle of the quoted string from the output.

Enter the following:

```
say 'ABC
DEF'
```

The output is:

```
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:

```
ABC                                      DEF
```

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

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

The output is:

```
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 as follows:

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

The output is:

```
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 operand in CNMSTYLE to influence the suppression of non-REXX (for example, NetView) commands.

| **Note:** IGNRLSUP is ignored for commands issued from a REXX command list.
| Refer to *Tivoli NetView for z/OS Command Reference* for more information.

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 CNMSTYLE, 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 158 for more information about suppression characters. You can also do suppression with the HOLE stage of the PIPE command. Refer to *Tivoli NetView for z/OS Customization: Using Pipes* for information.

NetView 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.

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 should 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, there is data 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 20.

Using the SAY Instruction

The REXX SAY instruction enables a character string of any length; however, NetView can display only 32728 characters at a time.

When you issue a REXX SAY instruction in a REXX command list for the NetView program, a 12-character header precedes the data displayed on the operator’s screen. The header contains the 1-character NetView message type of the message (HDRMTYPE()), followed by three blanks and the identifier of the domain under which the command list is running (APPLID()). For more information about HDRMTYPE() and APPLID(), see “Chapter 9. REXX Functions Provided by NetView” on page 117.

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

Do not use 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

When you use the CALL instruction in REXX command lists or Data REXX files for the NetView program, enclose the name of the command list or Data REXX file you want to call within single quotation marks. You can call only REXX command lists and assembler programs (*not* NetView command list language command lists or Data REXX files) with the CALL instruction. Operands to be passed to the called command list or assembler program need to be outside the quotation marks enclosing the name of the command list. If you want to avoid variable substitution for an operand, enclose the operand in quotation marks. For example, if you code the following CALL instruction to call an external command list named CLIST2:

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

and CLIST2 contains the following PARSE UPPER ARG statement,

```
PARSE UPPER ARG RES1 RES2 STATUS
```

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

If you execute CLIST2 as a command from another command list without the CALL instruction, for example:

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

CLIST2 receives the same values for the variables on the PARSE UPPER ARG statement, but the value of the ARG() function is set to 1 and the entire parameter string is placed in the variable RES1. RES2 and STATUS are set to null.

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

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, do not use the CALL instruction to invoke another command list.

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 NetView program.

Some REXX functions return different values depending on the operating system under which the command list containing the functions runs. For example, DATE() returns the current date in different formats depending on the operating system.

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

For restrictions on using REXX functions in Data REXX files, see “Using Data REXX” on page 20.

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

Note: 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 NetView program. The NetView 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.

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

Refer to *Tivoli NetView for z/OS Customization: Using Assembler* for information about the DSIRXEBS macro and function packages, and about writing function package directories.

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

Changing the Environment Addressed by REXX Command Lists

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

Refer to 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, as follows:

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

You can also define the command name in a CMDMDL statement and the synonym (using lowercase characters) in a CMDSYN statement. For example, if you want to enter the WTO NetView command in lowercase characters, change the CMDMDL and CMDSYN statements for WTO in DSICMD as follows:

```
WTO    CMDMDL    MOD=DSIWTC...
       CMDSYN    wto
```

If you code a CMDSYN statement for the WTO NetView command as shown in the previous example, you can use the following coding technique:

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

Notes:

1. The command must either be uppercase, or exactly match the case in a CMDSYN statement.
2. 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.
3. Programs, such as SDSF, that do program calls will abend if they are linked or attached. Because the NetView status monitor can perform program calls, NetView cannot link or attach to a program that can perform program calls.
4. The NetView program does not support a TSO/E environment in the NetView address space.
5. The NetView program returns an error if you try to execute a command that is routed to an incorrect addressing environment.
6. The ADDRESS command only supports 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.

Coding the chain this way enables you to use SIGNAL ON HALT to try to recover if EXECIO with the FINIS option encounters an error closing a file. If the EXECIO command encounters an error, it sets the RC variable to a nonzero return code. Refer to 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 FNDMBR(...) NetView REXX function to determine the members existence before issuing the EXECIO command.

See “PRINT Example” on page 186 and “TYPE Example” on page 188 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.

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

If you are issuing these command as a shortcut, you should be aware that MVS and VTAM commands do not have a definite indication of ending like NetView commands. In many situations, issuing these commands will cause an extended wait period that continues after your procedure ends. These waits are generally 60 seconds, and are seen when operators use procedures from NMC, using CONSOLE=*ANY* support, when issuing labeled commands and similar situations.

To avoid wait conditions, do one of the following:

1. Define appropriate wait conditions for your command to NetView. Use the CCDEF command to do this.
2. Isolate MVS and VTAM commands inside pipelines within your procedure as follows:
   ```
   'PIPE CC MVS D A,L | CONS'
   ```
   The wait due to MVS D A,L is resolved directly by the pipeline and is not inherited by the procedure that it is in.

   Note: Pipelines cannot be issued with Data REXX files.

Using the NetView ALLOCATE and FREE Commands

The NetView program provides the ALLOCATE and FREE commands to enable you to dynamically allocate and deallocate data sets from the NetView program.

These commands closely resemble the TSO/E commands for allocating and deallocating data sets. However, because these commands are provided by the NetView program, you do not need to use the ADDRESS MVS instruction when using these commands in a command list. Simply enclose the commands in quotation marks as you do for other NetView commands. The TYPE, TYPEIT, and PRINT examples in “Appendix C. Examples of REXX Command Lists for NetView” on page 177 use the NetView ALLOCATE command.

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

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

Using REXX Command Lists

Each time a REXX command list is executed in the NetView program, REXX sets up a REXX environment for NetView. When the command list ends, this unique environment can be held for reuse by the same task. If two command lists are executing 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 caller's REXX environment.

Before executing REXX command lists, consider how many concurrent REXX command lists are normally 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. Refer to the NetView online help for additional information about the DEFAULTS and OVERRIDE commands.

The NetView program retains REXX environments to improve REXX environment initialization performance. If more than one REXX environment is available when a REXX command list is executed, the REXX command list can execute using a different REXX environment. Whether this occurs depends on the order in which other REXX command lists were started and ended during concurrent execution 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 NetView program retains, to minimize the storage each task using REXX requires. However, if you set this number to zero, the NetView 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 executing any REXX command lists. By default, REXX gets sufficient storage for a REXX command list with about six levels of nested invocations. 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 invoked 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 NetView 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.

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.

Refer to 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 NetView 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, and you should 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'
```

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

Note: Data REXX only supports the REXX PARSE instruction.

Tracing REXX Command Lists

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

TS

```
►►──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 execute 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 syntax of the TE command is:

TE

```
►►──TE──────────────────────────────────────────────────────────────►◄
```

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

Return Codes in REXX Command Lists

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

Possible RC values and their meanings are as follows:

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. The –3 return code passes control to the FAILURE label if you code SIGNAL ON FAILURE.

–5 The command list 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” on page 34 for more information about using the SIGNAL instruction with the NetView 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 three different labels in the command list. These labels are as follows:

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

  If your command list recovers from the error, you can return the appropriate return code to the calling command list. If your command list does not recover from the error, pass the failure to the calling command list with EXIT –1.
- 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 should reflect the severity of the error. A zero (0) return code is recognized by all NetView commands as an indication of successful completion, while all positive return codes indicate that an error occurred.
- SIGNAL ON HALT passes control to a label named HALT when the command list is canceled. A command list is canceled when:
  - A RESET NORMAL command is executed on the current operator task while your command list is running.
  - A CLOSE IMMED command is executed on any task in your NetView 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’s 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 NetView 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 following statement of the command list tests the RESULT variable for the –5 cancel condition.
3. If you code SIGNAL ON FAILURE, the NetView program passes only the halt condition to the calling command list if you code EXIT –1.

For more information about the SIGNAL instruction, refer to the REXX library.

Chapter 3. REXX Instructions Provided by NetView

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 NetView 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 A. Comparison of REXX and NetView Command List Language" on page 169 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.

The REXX instructions provided by the NetView program are:
- TRAP
- WAIT
- WAIT CONTINUE
- MSGREAD
- FLUSHQ
- GLOBALV

Note: These instructions are not supported by Data REXX.

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

For information about NetView pipelines, refer to the *Tivoli NetView for z/OS Customization: Using Pipes*.

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

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

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

When a WAIT instruction is issued, the command list stops processing until one or more of the messages specified on the TRAP instruction are received or until the specified period of time elapses. When a WAIT instruction completes, the value returned by the EVENT() function will indicate the reason that the WAIT instruction completed.

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

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.

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

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 invoking command list. Therefore, TRAP commands issued from subroutines or functions operate the same as if they were invoked in the calling command list.

If you used the REXX CALL instruction to invoke 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 invoked 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 will be available to the calling command list and will be placed in the message queue. It is possible, therefore, for the message queue to grow large enough for NetView to run out of storage.

To prevent that from happening, you can do one of the following:
- 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 that call other command lists or are called by other command lists can issue a WAIT instruction.

Notes:

1. The WAIT instruction cannot be used in Data REXX files.
2. You do not need to use the WAIT instruction if the command list starts VIEW. For more information on VIEW, refer to the *Tivoli NetView for z/OS Customization Guide*.

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 instructions 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 invoke 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 invoked the nested command list without using the CALL instruction, the trapped messages are available only to the command list that issued the TRAP instruction.

Using MSGREAD in Nested Command Lists

You can code the MSGREAD instruction in both a nested REXX command list and the initial REXX command list. If you use the REXX CALL instruction to invoke a nested command list, trapped messages are available to both the initial and nested command lists. If you invoke 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

NetView sets the values of the following functions based on the information contained in a message read by a MSGREAD instruction. The functions are:

ACTIONDL()
ACTIONMG()
AREAID()
ATTNID()
AUTOTOKE()
CART()
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()
MSGVAR(*n*)
MVSRTAIN()
NVDELID()
PARTID()
PRTY()
REPLYID()
ROUTCDE()
SESSID()
SMSGID()
SYSCONID()
SYSID()

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

```
DSI008I SPAN1  NOT ACTIVE
```

The functions are set as follows:

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

For more information about these and other message processing functions, see "Message Processing Information Functions" on page 128 and "MVS-Specific Message Processing Information" on page 135.

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 MSGREAD 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 Volume 2 of the *Tivoli NetView for z/OS Command Reference* for information concerning working with multiline messages.
4. The MSGVAR(1) - MSGVAR(31) functions can be given values when a command list is invoked 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 invoked, save those values in variables before issuing a MSGREAD instruction. This lets you use the values that are modified by MSGREAD.
5. After using MSGREAD, save the values of the message functions in variables before issuing another MSGREAD instruction.

Part 3. Writing Command Lists in the NetView Command List Language

Chapter 4. Writing Simple Command Lists in the NetView Command List Language 43
What the NetView Command List Language Includes 43
Coding Conventions for NetView Command List Language Statements 43
Conventions for General Coding 44
Conventions for Continuing a Statement. . . . 44
Conventions for Double-Byte Character Set Text 45
Conventions for Suppression Characters. . . . 46
Labels 46
Variables 47
Variable Substitution Order 47
Parameter Variables 48
Passing Parameter Variable Information to a Command List 49
Using Parameter Variables in a Command List . 50
Passing Parameter Variables to a Nested Command List 50
Using Quoted Strings or Special Characters in Parameter Variables 51
Null Parameter Values. 51
Control Variables 52
User Variables 52
Hexadecimal Notation. 53
Comments. 54
Null Statements 54
Assignment Statements 54
Control Statements 56
&CONTROL Statement 56
Writing to the Operator 57
&WRITE Control Statement 58
&BEGWRITE Control Statement 59
&PAUSE Control Statement 61
Using NetView Commands with &PAUSE . . . 62
An Example Using &PAUSE. 63
NetView Built-in Functions 63
&BITAND 64
&BITOR 64
&BITXOR 65
&CONCAT 66
&HIER 66
&LENGTH 69
&MSUSEG 70
&NCCFID 71
&NCCFSTAT 72
&SUBSTR 73
Using &SUBSTR with DBCS Characters . . . 74

Chapter 5. NetView Command List Language Branching 77
&IF Control Statement. 77
&GOTO Control Statement 79
&EXIT Control Statement. 79
&WAIT Control Statement 81
Coding an &WAIT Control Statement 82
The Event=-Label Pair 83
Error Conditions. 86
Coding Message=-Label Pairs 86
Ending an &WAIT 86
Using NetView Commands with &WAIT . . . 86
Control and Parameter Variables Used with &WAIT 87
Using &WAIT in Nested Command Lists . . . 88
Customizing the &WAIT Statement 89
Ending &WAIT If CONTWAIT Is in Effect . . 91
Suggestions for Coding &WAIT 91
Sample Using &WAIT 92

Chapter 6. NetView Command List Language Global Variables 95
Using &TGLOBAL and &CGLOBAL 96
&TGLOBAL 96
&CGLOBAL 97
Updating Task Global Variables Using &TGLOBAL 98
Extent of Variables When Using &TGLOBAL and &CGLOBAL 99
GLOBVAR1 Example 100
GLOBALV Command 102

Chapter 4. Writing Simple Command Lists in the NetView Command List Language

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

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:
- 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 "Chapter 6. NetView Command List Language Global Variables" on page 95.

The NetView command list language enables you 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 "Chapter 5. NetView Command List Language Branching" on page 77.

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

Coding Conventions for NetView Command List Language Statements

Like any other language, the NetView command list language requires that you follow syntax rules. The following coding conventions for NetView are divided into sections describing the conventions for:
- General coding
- 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 as follows:
  - Optionally, code a label. The label must begin in column 1. You cannot branch to this label.
  - 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 there is one) or before a keyword.
- Code at least one blank between a control statement and the first operand.
- Separate operands with one or more blanks, or a single comma with no blanks.
- Code any number of leading or trailing blanks on your statements.
- Use lowercase letters only as comments or part of a message sent to the operator. In all other cases, use uppercase for alphabetical characters A-Z.
- Code statements so that the maximum length is 32000 characters after variable substitution.

  Note: To familiarize yourself with how variable substitution works, see "Variable Substitution Order" on page 47.
- Code comment lines with an asterisk (*) as the first nonblank character of the command list line. Place the comment after the asterisk. Comment lines cannot appear on the first line of a command list.
- Code the command list so that it ends by processing the last command list statement, or by reaching an &EXIT statement. An operator entering RESET also ends the command list.

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

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

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

  All the blanks from the `P` in `KEEP` to the `B` in `BLANKS` would be kept. The first line would write 64 characters to the output screen (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 would cause the first four characters of the second line to be placed at the end of the first line. In the example, this would be two blanks and the letters BL. However, because NetView will not split a word across lines of the output screen, the message is displayed as:

  ```
  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 NetView 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 there are an even number of bytes in each DBCS string. (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. This 11 bytes includes 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 appear 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

You can define a suppression character in CNMSTYLE and use it to prevent a command list command or statement from being displayed on the operator's screen during execution.

Note: IGNRLSUP is ignored for commands issued from a NetView command list.

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 label. The suppression character will prevent you from branching to the label unless the command list line containing that label has already been processed.

In Figure 13 on page 46, 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 13. 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 "Chapter 5. NetView Command List Language Branching" on page 77.

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 NetView cannot find the label, processing stops, and NetView issues an error message.

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, #, @, $). 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, NetView ends the command list. You can also code labels as internal comments to show where different parts of your command list start. For example, you can use labels to highlight certain processing routines.

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 "Chapter 5. NetView Command List Language Branching" on page 77.

Variables

Variables enable you 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 "Chapter 6. NetView Command List Language Global Variables" on page 95 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 to 11 characters (A-Z, 0-9, #, @, $).

Variable Substitution Order

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

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. NetView then scans to the right and takes any following numbers as part of the parameter variable. When NetView comes to a blank or a letter, the search stops. If a special character (nonalphanumeric) is found, NetView delimits the variable name.

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

Control keywords
For more information, see "&CONTROL Statement" on page 56.

&PAUSE statement
The variables are assigned values when you enter a GO command. For more information, see "&PAUSE Control Statement" on page 61.

&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 77.

Any statements in an &BEGWRITE NOSUB series of messages
For more information, see "&BEGWRITE Control Statement" on page 59.

Built-in functions
For more information, see "NetView Built-in Functions" on page 63.

Parameter Variables

A parameter variable is a positional variable that is defined at the time a command list is run. You specify parameter variables by entering them as operands following the name of the command list that you are running. Parameter variables have the following characteristics:
- Identified within the command list by a numbered position, for example, &1
- Entered following the command list name at run time
- Delimited by commas, apostrophes, or blanks

When you code your command list with parameter variables, use the following guidelines:
- You can use up to 31 parameter variables in a single command list
- You can use the same parameter variables more than once in a command list

- The value of a parameter variable can be 238 characters long
- Parameter variables can contain either numerical or character values
- When used in an arithmetic expression (for example, addition or subtraction), a parameter variable can have a numerical value between -2147483647 and 2147483647. When used in a nonarithmetic 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 NetView receives a message coded in an &WAIT statement, NetView sets some control variables (for example, &MSGORIGIN, &MSGID, &MSGCNT, and &MSGSTR) and also changes the values of the parameter variables (&1 – &31) to reflect the information in the received message.

See “Control Variables” on page 52 for information about these variables. LINKPD sets the same control and parameter variables. See “LINKPD Results” on page 111 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 would be coded &2 at the place where you want the value of that parameter.

Assume that you wrote a command list named RESC to start resource LU100 as shown in 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 inactivate 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 7. Automation Resource Management” on page 105.

Using Parameter Variables in a Command List

There is no set order 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 there are two or more parameter variables in one command list statement, the rightmost variable is changed first. NetView continues to scan right to left and replaces the next variable. You can use this method to change the meaning of some of your parameter variables. If you need to test how many parameters an operator entered or what parameter values were entered, use the control variables &PARMCNT and &PARMSTR. They are described in “Control Variables” on page 52.

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 these variables are known to the next activated command list.

The following are examples of 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:

Variable	**Value**
&1	55
&2	ALPHA
&3	BETA

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

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

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

Note: 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 quotes. If you want a text string to be taken as the value for one parameter, it must also be made a special character string.

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

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

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

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

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

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

Suppose you activated the RESC command list by entering:

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

The parameter variables in this case contain the values:

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

Null Parameter Values

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

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

In this example, TSO is &1, TAF01F00 is &2, &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 NetView based on system information. To use a control variable, place the variable name in the command list at the location where you want the information to be accessed. When the command list runs, NetView gives the correct values to each control variable. Use the LISTVAR command to view the values of 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 110.

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

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 54.

The &PAUSE control statement halts the command list, enables the operator to enter data, and picks up the value of the user variable from the operator when the command list continues. &PAUSE is described in "&PAUSE Control Statement" on page 61.

When you create user variables, observe the following rules:

- The first character must be an ampersand (&).
- The first character following the ampersand must be a letter or a symbol, not a number. Otherwise, NetView treats it as a parameter variable.
- The ampersand 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 numerical 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 numerical 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 NetView has already defined as a control statement, control variable, or built-in function. However, you cannot use that NetView provided control statement, control variable, or built-in function anymore in the command list.

Table 4 on page 53 shows some examples of user variable names.

Table 4. User Variable Names

Valid	Non-valid	Reason
&A	&2A	Will be 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:

```
►►──X'n'──────────────────────────────────────────────────────►◄
```

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, NetView prefixes the number with a zero (for example, X'2C6' would be converted to two bytes whose hex value would be 02C6). The maximum length of *n* is 255 hexadecimal digits.

The following are examples of 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:
- 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 46 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, NetView ignores the statement when the command list is run.

Assignment Statements

Assignment statements give values to variables and do arithmetic operations within a command list. The syntax of an assignment statement is:

assignment

```
►►──&variable = expression──────────────────────────────────────────►◄
```

Figure 14. Assignment Statement

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

When the command list runs, the value of the user variable is set to the value of the expression. For example, 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:

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 quotes. 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:

```
&USERVAR = &OPID
```

Note: Using a control statement as a variable is not valid, even if the control statement is enclosed in single quotes. 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 will be set to a number. The only characters you can use in a numerical value are 0-9. The numerical value can be immediately preceded by a character indicating whether the value is positive (+) or negative (−).

The plus or minus sign must be separated from the numbers on each side by at least one blank unless it indicates a positive or negative number (−2, −4). For example, both 4 − 2 and 4 − −2 are correct, but 4 −2 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:

```
&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 63 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 NetView acts on other statements in the command list. You can use the control statements in this chapter either for straight-line coding or in conjunction with the statements described in "Chapter 5. NetView Command List Language Branching" on page 77 for conditional processing.

You can use control statements to change the sequential order of processing. Command list control statements enable you to:
- Send messages to the operator from the command list.
- Control the order in which commands are run.
- Ask the operator to enter information needed to continue the command list.
- Wait for a solicited message to arrive before continuing the command list.

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

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

The control statements follow:

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

&CONTROL
Indicates the command list statements that are shown on the operator's 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's terminal while the command list is running. The indicated command list statements are displayed after all substitutions have been made and before the command list statements run. 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 syntax of the &CONTROL control statement is:

&CONTROL

```
              ┌─ALL─┐
►►──&CONTROL ─┼─────┼──────────────────────────────────────────────────►◄
              ├─CMD─┤
              └─ERR─┘
```

Where:

ALL
Displays all command list statements at the operator’s 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. Once your command list is tested, &CONTROL CMD or &CONTROL ERR 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 &CONTROL without operands, or if you do not code &CONTROL, the default is &CONTROL ALL.

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

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, message DSI013I is not displayed.

Writing to the Operator

The &WRITE and &BEGWRITE statements send messages to the operator terminal. &WRITE sends a one-line message, and &BEGWRITE 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 control statement, the text is sent to the operator as a message, but it is not run as a command list command.

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

If you are sending more than one message line or displaying a table that takes up the whole screen, you might want to use the NetView VIEW command instead of using &WRITE or &BEGWRITE.

&WRITE Control Statement

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

&WRITE

```
►►──&WRITE message_text──────────────────────────────────────────────────►◄
```

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

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

Activating this command list by entering PATH HD3790N1 causes Figure 15 on page 59 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 15. 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 16 on page 59 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 16. Sending One-line Messages to the Operator

&BEGWRITE Control Statement

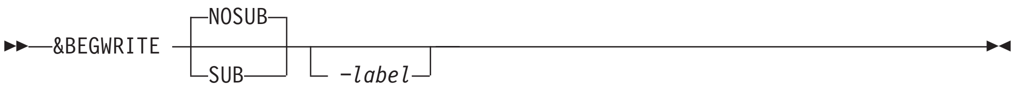
You can use &BEGWRITE to write a series of lines to the operator terminal. You can also control whether variables are replaced before sending the messages.

You 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 46 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 syntax for the &BEGWRITE statement is:

&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 NetView cannot find the label, the rest of the command list statements are sent to the operator as comments and the command list is ended. If there is no label on &BEGWRITE, only the first command list statement after &BEGWRITE is sent to the operator.

You can code a variable for your label on &BEGWRITE. 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, NetView assumes NOSUB.

SUB
Causes NetView to carry out substitution on the message text before sending the messages to the operator. See "Variable Substitution Order" on page 47 for information about how NetView carries out variable substitution.

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

Figure 17 on page 60 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 17. &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 18 on page 61 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 18. 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

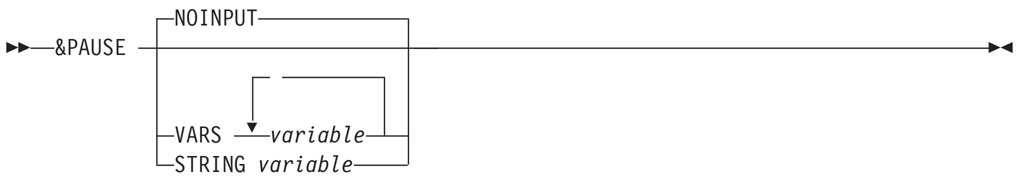
Using the &PAUSE control statement along with other commands, you can code command lists that ask the operator questions and pick up the entered responses. Use the &BEGWRITE and &WRITE control statements to send instructions to the operator. For example, you can code the command list to instruct the 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 52 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 syntax for the &PAUSE statement is:

&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` appears 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 enable the operator to suspend and then resume command list processing during an &PAUSE. Once 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` reappears 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 the first operand becomes the first user variable, the second operand becomes the second user variable, and so on. Operands are separated by either a blank or a comma. If you want to include a blank or a comma as part of one variable, use either &PAUSE STRING or put the operand between single quotes.

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 invoke your command list from any NetView component. If you expect the command list to run from components other than the command facility, use the NetView command NCCF in the command lists to present the operator with the command facility 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 NetView 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 operator's last name, first name, and middle initial. The command list pauses until the operator enters a GO or RESET command. To continue processing the current command list, the operator enters the GO command followed by the string required by the command list.

If the operator enters the following:

```
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 77 for information about the &IF control statement.) In an assignment statement, the value of the user variable is set to the result of the built-in function's operation. Two of NetView's 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 9. REXX Functions Provided by NetView" on page 117.) Although they appear similar, they are not the same. With the exception of HIER() and MSUSEG(), both built-in functions and NetView command list language variables start with an ampersand (&). The difference is as follows:

- A variable is replaced by its value when the command list runs. The variable is really 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 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 77.

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 following syntax describes the &BITAND function:

&BITAND

```
►►──&BITAND string1 ──┬─────────┬──────────────────────────────────►◄
                      └─string2─┘
```

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 are two examples of the &BITAND operation:

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

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

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 following syntax describes the &BITOR function:

&BITOR

```
►►──&BITOR string1 ──┬─────────┬──────────────────────────────────────►◄
                     └─string2─┘
```

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 are two examples of the &BITOR operation:

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

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

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 following syntax describes the &BITXOR function:

&BITXOR

```
►►──&BITXOR string1 ──┬─────────┬─────────────────────────────────────►◄
                      └─string2─┘
```

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 are two examples of the &BITXOR operation:

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

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

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 syntax of the &CONCAT built-in function is:

&CONCAT

```
►►──&CONCAT ──┬─&variable─┬── ──┬─&variable─┬──────────────────────────────►◄
              └─constant──┘     └─constant──┘
```

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 as follows:

&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.

The &HIER syntax is:

HIER

```
►►──&HIER ─┬───┬──────────────────────────────────────────────────────────►◄
           └─n─┘
```

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 than 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 `aaaaaaaa1111` that corresponds to *n*. If there is no name/type pair that corresponds to *n*, then a null value is returned.
3. &HIER returns null under the following conditions:
   - If the command list is not executed 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 19 on page 68.
5. If a complex link exists in a resource hierarchy, there might be resource levels that do not appear in the information returned by the &HIER function. 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 19. 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 19. Example of a &HIER Parsing Template (Part 2 of 2)

&LENGTH

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

&LENGTH

```
►►──&LENGTH ──┬─&variable─┬──────────────────────────────────────────────►◄
              └─constant──┘
```

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 is an example of 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 variable settings are:

&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.

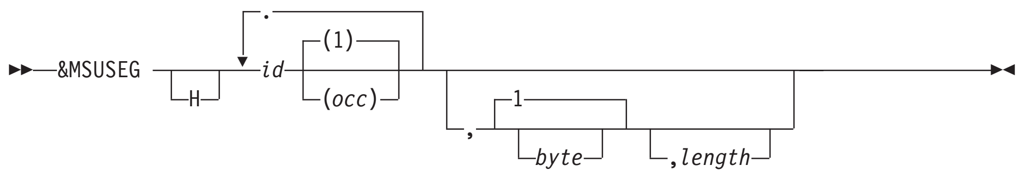
&RESLEN becomes the length of the first parameter variable. The first parameter variable is LU2525, so &RESLEN becomes 6.

&MSUSEG

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

The &MSUSEG syntax is:

&MSUSEG



Where:

byte
: The byte position into the lowest ID specified in *id*, counting from 1 in decimal. 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 quotes when quotes 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) 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 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. 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* would be considered as the *byte* operand.

2. If the location is not found, or if the command list containing the &MSUSEG function was not executed 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.
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.

For more information about the automation table, refer to the*Tivoli NetView for z/OS Automation Guide* book. For more information about vector definitions, refer to the SNA library. For more LU6.2 and MSU information, refer to the *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 CNMSTYLE. 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 *Tivoli NetView for z/OS Administration Reference*.

The syntax of NCCFID is:

&NCCFID

```
►►──&NCCFID number─────────────────────────────────────────────────►◄
```

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 is an example of how to use &NCCFID:

```
&DOM1 = &NCCFID 1
&DOM2 = &NCCFID 2
&DOM3 = &NCCFID 3
START DOMAIN=&DOM1
START DOMAIN=&DOM2
START DOMAIN=&DOM3
```

Assume 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 as follows:

&DOM1	ALPHA
&DOM2	BETA
&DOM3	GAMMA

The three domains are then started with the NetView START command.

In this example, the operator must know there are three domains that can be started. You can also use the &IF control statement to test &NCCFCNT to find the number of domains and start them.

&NCCFSTAT

The &NCCFSTAT function returns a value indicating whether you have an active cross-domain session with the specified domain. The syntax of &NCCFSTAT is:

&NCCFSTAT

```
►►──&NCCFSTAT domain──────────────────────────────────────────────►◄
```

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 as follows:

&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 77).

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. The syntax of &SUBSTR is:

&SUBSTR

```
►►──&SUBSTR &variable start ──┬────────┬──────────────────────────────────►◄
                              └─length─┘
```

Where:

length
: The number of characters to parse, beginning with the specified *start* position. If no *length* is specified, the parsing will be 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 had 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 as follows:

&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. NetView never pads substrings with blanks. If you specify a length that is too long, NetView assumes no length was specified and uses the entire string beginning at the starting position. If the length is 0, or the starting position is beyond the variable length, the result of &SUBSTR is null.

Figure 20 on page 74 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 20. Using &APPLID to Determine the Domain Name

Using &SUBSTR with DBCS Characters

When using double-byte characters along with Roman characters (A-Z; a-z), the &SUBSTR function adjusts the variable as follows:

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 is an example of 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; and D1, D2, D3 are double-byte characters.

Using this value, &SUBSTR works as follows:

```
&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.

Chapter 5. NetView Command List Language Branching

This chapter describes the conditional and unconditional branching statements in 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 syntax of the &IF control statement is:

&IF&THEN

```
►►──&IF expression_1 ──┬──=───┬── expression_2 &THEN statement──────────────────►◄
                       │ └EQ─┘ │
                       ├──¬=──┤
                       │ └NE─┘ │
                       ├──<───┤
                       │ └LT─┘ │
                       ├──>───┤
                       │ └GT─┘ │
                       ├──<=──┤
                       │ └LE─┘ │
                       ├──>=──┤
                       │ └GE─┘ │
                       ├──¬>──┤
                       │ └NG─┘ │
                       └──¬<──┘
                         └NL─┘
```

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 54.

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 quotes ('') 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 could be equal to null, then the following syntax should be used:

```
7 > 3 + 0&1
```

In this example, the zero (0)&1 will evaluate to zero (0) because &1 is null. Therefore, the expression 3 + 0 is compared to 7. If &1 was equal to 9, the expression 3 + 09 would be 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 syntax of the &GOTO control statement is:

&GOTO

```
►►──&GOTO -label────────────────────────────────────────────────────►◄
```

Where:

-label
Identifies the target statement in this command list where processing continues.

When NetView interprets the &GOTO statement, it searches the command list for a statement starting with this same label. NetView transfers control to that statement and continues the command list processing. The statement identified by the label can be before or after the &GOTO statement.

You can code a variable for your label as long as the variable is replaced by a value before NetView processes the &GOTO statement. See “Labels” on page 46 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 there is an error. You can use &EXIT with &GOTO to control the flow of the command list. The syntax of the &EXIT control statement is:

&EXIT

```
►►──&EXIT ──┬────────┬──────────────────────────────────────────────►◄
            └─number─┘
```

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 take action based on the return code. See “Command List Information” on page 120 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 there is no &EXIT statement, a zero return code is set.

Figure 21 on page 80 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 21. 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 15 for more information using &WAIT under the PPT. For additional information about using &WAIT with common operations services commands, see "Chapter 8. Common Operations Services Commands" on page 109.
2. NetView pipelines, invoked with the PIPE command, provide both extended function and reduced complexity for the automation of message handling. The PIPE command is therefore a recommended alternative to the &WAIT control statement. For information about NetView pipelines, refer to the *Tivoli NetView for z/OS Customization: Using Pipes* book.

If you use &WAIT in an automation task command list, be sure to specify a reasonable time-out value. For instructions about coding a time-out event, see "The Event=-Label Pair" on page 83.

If the trapped message satisfies the wait condition, processing of the waiting command procedure resumes. If you do not suppress the message at this point, it continues with the message flow. If you suppress the message, however, NetView marks it for deletion. In this case, automation-table processing does not occur and NetView does not display or log the message.

&WAIT does the following in a command list:

- It causes NetView to monitor the operator station task (OST) for specific messages and takes action if the message arrives. For example, the command list issues a VTAM command to activate a resource. When VTAM sends the message saying the resource is active, &WAIT initiates a specific action based on the successful activation of the resource.
- It initiates a specific action if a message does not arrive in a specified period of time. For example, for your installation, you 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 NetView 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.

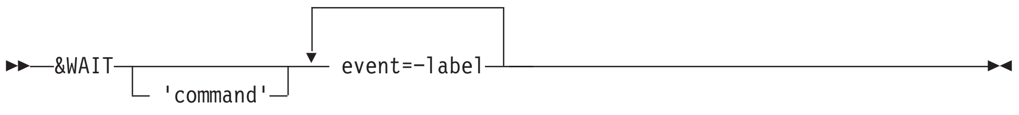
&WAIT and &PAUSE work differently. With &PAUSE, the command list does not continue until the operator enters the GO command. Operands on the GO command are used in the command list. However, because &WAIT causes the command list to wait for a specific event or events, GO is used to resume the command list only if the event never occurs. When a command list is in a wait state, NetView 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 89 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 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 syntax of the &WAIT control statement is:

&WAIT



Where:

'*command*'
: Is any command or command list that you can issue from NetView. This command is optional. It is usually the command from which the command list is waiting for messages. For example, if you want the command list to wait for a successful session startup, the entire BGNSESS command is coded between single quotes. Be sure to code command list continuation characters before the *event=-label* pairs. The command is run as soon as it is reached in the command list.

  You can code one of the NetView timer commands, AT, EVERY, or AFTER, in the &WAIT statement. If the scheduled command is a command list, it cannot run until either the current command list is complete or the STACK command is entered.

event=-label
: Is an *event=-label* pair. You can code as many of these pairs as you want on an &WAIT statement, up to the limit of 255 characters. The event is usually a message for which the command list is waiting. The event can be a trigger that ends the wait state before the message arrives. The &WAIT statement causes NetView to scan all messages sent to the operator. If a message matches one of the events coded, the command list goes to the line with the specified label and continues processing from the labeled statement. For more information about the types of events that can satisfy an &WAIT, see “The Event=-Label Pair” on page 83.

When NetView receives the message it is waiting for, the message is displayed on the operator terminal, as are all NetView messages. However, in this case, the message type is `W` unless the message satisfying the &WAIT originated from a command list, in which case the message type remains `C`. If you do not want the operator to see this message, see “Customizing the &WAIT Statement” on page 89.

NetView checks only messages that are intended for the operator's screen. If you code exit routine DSIEX02A (output to the operator), the &WAIT control statement might not set the message for matching. For example, if DSIEX02A deletes the message, &WAIT 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, you should wait only for messages that are displayed on the NetView console.

Notes:

1. When coding the &WAIT command control statement, it is important to code an *event=-label* pair for message DSI210I and *NN. 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 should notify the operator of the error and exit the command list.
2. The `W` signifying a wait state, if present, remains in the upper right corner of the panel while this initial &WAIT command is processed. The `W` tells the operator that NetView is still waiting for messages. If the operator enters GO before this command or command list completes processing, the GO is rejected with message `DSI016I NOT IN PAUSE OR WAIT STATUS`. When the command or command list is complete, the GO is accepted. RESET ends a command list that is in a wait state. If you enter the STACK command, the `W` does not remain in the upper right corner of the panel.
3. 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 89.

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 46. The label coded on the &WAIT statement can be a variable, but you should not use parameter variables.

You can pass control to the label on an &WAIT statement by specifying an *event=-label* pair. The events you can use are:

- *token*
- *ERROR
- **nn*
- *ENDWAIT

Descriptions of the previously mentioned events are as follows:

token This event occurs when NetView 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 are examples of some of the ways you can specify the messages for which you want the command list to wait:

domainid.token
: The event occurs when NetView receives any message whose domain identifier matches the 1–5 character *domainid* and whose first token matches *token*.

dom.token*
: The event occurs when NetView receives any message whose domain identifier matches the partial domain identifier specified by *dom** and whose first token matches *token*. For example, NCCF*.DSI463I means the event occurs when a DSI463I message is received from any domain with an identifier that starts with NCCF (such as NCCFA or NCCFB).

**.token*
: The event occurs when NetView receives any message whose first token matches *token*. The message can be from any domain.

token
: The event occurs when NetView receives any message whose first token matches *token*. The message can be from any domain.

*tok**
: The event occurs when NetView receives any message whose first token matches the partial token specified by *tok**. For example, DSI* means the event occurs when NetView receives any message whose first token begins with DSI (such as DSI463I or DSI386I).

*
: The event occurs when NetView receives any message 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, could cause a loop or other undesirable 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 nonnumeric and nonalphabetic characters, use the DOMAIN.TOKEN format. Remember, however, that NetView 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 22 on page 84 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 22. 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 the 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. Normally, 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 *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, NetView continues to wait for the messages associated with this command even if the command ends with an error. If NetView is waiting for a message that says the command was successful, the operators running this command list are delayed until someone issues GO or RESET. If *ERROR is satisfied, the message control variables are set as follows:

&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, NetView honors the MSGID=-*label* pair first because that event occurs first.

**nn* This event occurs after *nn* seconds. If no other event occurs, the &WAIT ends and control passes to the labeled statement. You can code a value between 1 and 32767 seconds (9 hours, 6 minutes, 7 seconds). If you do not code **nn* and none of the events of the &WAIT are satisfied, &WAIT continues until the operator enters a GO or RESET command.

***ENDWAIT**

This event occurs when the operator or a command list issues a GO command. If you do not code *ENDWAIT=-*label*, the GO command continues processing with the statement following the &WAIT command.

Error Conditions

If an error condition occurs, NetView should be able to go to another part of the command list and take appropriate action. Consider the types of errors you can have and plan to handle them by coding *ERROR, **nn*, and *ENDWAIT events.

Coding Message=-Label Pairs

The order in which you code MSGID=-*label* pairs is important. NetView scans the pairs in the order you code them, from left to right.

For example, assume you code the following statement:

```
&WAIT IST*=-ALL,IST123I=-SPECIAL
```

When NetView receives IST123I, it goes to the label -ALL, not -SPECIAL. You should code IST123I before IST*.

You can code as many events as required on one &WAIT control statement up to 255 characters. Remember to use continuation characters if the event pairs take up more than one line. Code the message and domain identifiers in the order that you want them processed. NetView scans the list from left to right until a match is found.

Ending an &WAIT

An &WAIT 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 as follows:

GO Ends the wait.

If *ENDWAIT is coded, processing continues with the labeled statement.

STACK
Suspends command list processing and causes any commands that 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 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

NetView 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	&KEY	&MSGGSYID
&ACTIONMG	&LINETYPE	&MSGGTIME
&AREAID	&MCSFLAG	&MSGID
&ATTNID (VSE only)	&MSGASID	&MSGORIGN
&AUTOTOKE	&MSGAUTH	&MSGSRCNM
&CART	&MSGCATTR	&MSGSTR
&DESC	&MSGCMISC	&MSGTOKEN
&HDRMTYPE	&MSGCMLVL	&MSGTSTMP
&IFRAUGMT	&MSGCMSGT	&MSGTYP
&IFRAUIND	&MSGCNT	&MVSRTAIN()
&IFRAUIN3	&MSGCOJBN	&NVDELID
&IFRAUI3X	&MSGCPROD	&PARTID (VSE only)
&IFRAUSB2	&MSGCSPLX	&PRTY
&IFRAUSC2	&MSGCSYID	&REPLYID
&IFRAUSDR	&MSGDOMFL	&ROUTCDE
&IFRAUSRB	&MSGGBGPA	&SESSID
&IFRAUSRC	&MSGGDATE	&SMSGID
&IFRAUTA1	&MSGGFGPA	&SYSCONID
&IFRAUWF1	&MSGGMFLG	&SYSID
&JOBNAME	&MSGGMID	&1 – &31
&JOBNUM	&MSGGSEQ	

NetView changes the values of the &1–&31 parameter variables to reflect the text of the message. Each parameter variable is set to a token of the message. Tokens are delimited by commas, apostrophes, or blanks. &1 is set to the first token following the message identifier (the token used by the &MSGID control variable). &2 is set to the next token to the right of &1, and so on up to a maximum of 31 variables.

For more information, see “Message Processing Information Functions” on page 128 and “MVS-Specific Message Processing Information” on page 135.

The following is an example of 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 NetView 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 invoking 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’s screen. If an &WAIT CONTINUE statement is encountered after a MSGID=-*label* is matched, and there is no other statement to end the command list or the &WAIT, the command list 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. You should be aware of the following considerations when using &WAIT with nested command lists:

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

  If 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 NetView 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

```
            ┌─ DISPLAY──┐  ┌─ ENDWAIT──┐
►►──&WAIT───┤           ├──┤           ├──────────────────────────────────────►◄
            └─ SUPPRESS─┘  └─ CONTWAIT─┘
```

-or-

&WAIT

```
►►──&WAIT CONTINUE────────────────────────────────────────────────────────────►◄
```

Where:

DISPLAY
Indicates that the message the command list is waiting for is to be displayed at the operator's terminal upon arrival to NetView. DISPLAY is the default value.

SUPPRESS
Indicates that any messages that satisfy an &WAIT are not displayed, logged, or automated.

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 5 on page 90 for valid option combinations.
3. The DISPLAY and SUPPRESS options can be changed at any point in a command list. Once 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.

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. ENDWAIT indicates that the wait ends after the first event

that satisfies the &WAIT. Although ENDWAIT does not end a wait already in process, operators can still use the GO command to end the wait. The RESET command, which ends a wait, also ends the command list.

Notes:

1. If neither ENDWAIT nor CONTWAIT is specified, then either DISPLAY or SUPPRESS must be specified.
2. ENDWAIT is the default *only if* DISPLAY or SUPPRESS is specified and CONTWAIT is *not* specified. See Table 5 on page 90 for valid option combinations.
3. The ENDWAIT and CONTWAIT options can be changed at any point in a command list. Once CONTWAIT starts, you must code another &WAIT statement with the ENDWAIT operand to return to the default value.

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.

Table 5. &WAIT Customization Options Matrix.
S = Specified operand
• = Default invoked

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 invoked 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 86 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 there is an error 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 normally.
- By entering the RESET command.

  The command list, including the command list that initiated it, ends.

Note: Because &WAIT CONTWAIT queues NetView messages, you should also code &WAIT CONTINUE to receive these queued messages. If you code &WAIT CONTWAIT with SUPPRESS and end the wait, you 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 86.
   - 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 91. The command list goes to the label specified for the event and continues processing.

To complete this section of the command list, do one of the following:

– Continue the wait by coding &WAIT CONTINUE.
– Specify that the next event is the last 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 23 on page 92 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's 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 23. 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) caused the wait to end because ENDWAIT was specified. Processing continues at the specified label (-REFORM for IST093I, -FAIL for IST105I). The awaited messages are not displayed because SUPPRESS was specified, but any other messages are displayed.

Upon successful activation of NODE1, the message text shown in Figure 24 on page 93 is displayed on the operator's terminal:

```
ACTONE NODE1
IST097I  VARY      ACCEPTED
VTAM MESSAGE IST093I WAS RECEIVED
NODE1 IS NOW ACTIVE
COMMAND LIST ACTONE COMPLETE
```

Figure 24. ACTONE NODE1 Message Text

Chapter 6. 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 only be referenced 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 execution.

NetView 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 both commands provide function, it is recommended that you use one or the other within a single NetView command list.
2. When you create global variables, the variable can be 1 to 11 characters in length. A-Z, 0–9, #, @, and $ are the only 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 numerical 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 &. Whenever you use the global variables (except when defining them), you must append an & to the variable name, just as you would for user variables.

You need two &s when referencing a global variable indirectly. See “Using Parameter Variables in a Command List” on page 50 and “Variable Substitution Order” on page 47 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 following control statement to define any variable as a task global variable. The syntax of the &TGLOBAL control statement follows:

&TGLOBAL

```
                 ┌──────────┐
                 ▼          │
►►──&TGLOBAL──── variable───┴──────────────────────────────────────►◄
```

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.

An example using the &TGLOBAL control statement is as follows:

```
&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 as follows:

- 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 99 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.

The following are suggestions for using task global variables:

- The PROFILE IC can set task global variables to indicate a message suppression level or message compression that is different for different types of operators. Command lists driven by various messages can test these variables to determine what information a particular operator needs.
- Any command list can set up and initialize any number of parameters for another command list running under the same operator task. This 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

```
            ┌──────────┐
            ▼          │
►►──&CGLOBAL─── variable─┴──────────────────────────────────────────────►◄
```

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.

Note: 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.

NetView Command List Language Global Variables

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 99 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 NetView-supplied command lists UPDCGLOB and SETCGLOB 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 are two examples of 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 25. CLIST1 Command List to Define, Update, and Reference Task Global Variables

CLIST1 in Figure 25 on page 98 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 hassignment 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 26. UPDT1 Command List to Update Task Global Variables

UPDT1 in Figure 26 on page 99 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 27 on page 100, 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.

In the following example, the values of the variables are shown in parentheses.

GLOBVAR1 Example

The following is an example of a command list containing global variables:

```
&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 27. 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 27. 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 27. GLOBVAR1 Example Showing Extent of Global Variables (Part 3 of 3)

GLOBALV Command

The GLOBALV command enables you to define, put, and get global variables in NetView command list language command lists. The GLOBALV command also enables you to save global variables in a VSAM database. You can restore saved global variables if NetView 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 on the GLOBALV command.

Part 4. Advanced Command List Topics

Chapter 7. Automation Resource Management 105
Defining NetView Automation Table Command Lists 105
Routing Messages from Automation-Table-Driven Command Lists 105
Implementing NetView Automation 105
Suppressing Messages 106
Determining What Task Controls a Command List 106
Testing Automation Command Lists 106
Verifying NetView Automation Table Entries 106
Keeping a Record of Automation Command Lists Executed 106
Testing Automation Command List Execution 107
Recovering from Looping Command Lists . . . 107
Considering Operator Interaction 107
Common Automation Problems 107

Chapter 8. Common Operations Services Commands 109
Common Operations Service 109
Common Operations Services Return Codes . . . 110
LINKDATA and LINKTEST Results 110
LINKDATA and LINKTEST Variables 110
LINKTEST Additional Variables 111
LINKPD Results 111
RUNCMD Results 112
Using RUNCMD in a Pipeline 112

Chapter 7. 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 upon many different possible criteria. This criteria includes:

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

As a result the automation table can cause various displaying, logging, or routing to occur. Commands or command lists can be invoked to analyze the message or MSU further before any action is taken.

To define an automation table, code automation statements in DSIPARM, then issue the AUTOTBL command using the name of that specific NetView automation table. You can enter the AUTOTBL command at a terminal, from a command list, or in an initialization command list at system startup.

You can parse important variable information in the text of a message into variables in the IF portion of an IF-THEN automation statement. You can use the variables as parameters of the command list you call as an action in the THEN portion of the statement. Using the variables this way enables you to ignore certain characters of the message text (such as commas and apostrophes) instead of treating them as command syntax elements.

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

Routing Messages from Automation-Table-Driven Command Lists

When command lists are invoked from the NetView automation table, the decision on where to route a message cannot always be made from within the automation table. For example, if you need to check the message text of a line other than the first line in a multiline write-to-operator (MLWTO) message before you decide where to route the message, you can do so in a command list, but not in the automation table. To route a message that causes a command list to be driven, use the MSGROUTE command from within automation-table-driven command lists 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 *Tivoli NetView for z/OS Automation Guide* book.

Suppressing Messages

You can suppress some messages so that the operator never 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 What Task Controls a Command List

In REXX, if you are not sure of the type of task under which a command list will run, have the command list check the TASK() 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. Refer to the REXX library for more information about conditional processing.

In NetView command list language, if you are not sure of the type of task under which a command list will run, have the command list check the &TASK control variable in the beginning of the command list. You can then use conditional processing to make the command list flexible enough to run differently under different tasks. See "Chapter 5. NetView Command List Language Branching" on page 77 for more information about conditional processing.

Testing Automation Command Lists

There are several ways to test automation command lists to ensure that the command list is called correctly from the NetView automation table and executes correctly after being called.

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 quotes.

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

If the message you create matches an entry in the NetView automation table, the table executes 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 Executed

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 executed directly from the NetView automation table. Having automation command lists send a message to the network log using the NetView MSG command enables you to keep track of which automation command lists are driven by which tasks and at what time.

For example, each automation command list could send a message to the log containing the name of the command list and the parameters passed upon entry. The task from which the message is issued and the time of the logged message are automatically saved in the log. You can remove these MSG commands from the automation command lists when testing is successfully completed.

Testing Automation Command List Execution

To test REXX automation command lists by tracing their execution, use the TRACE command. If your command list is being executed by a NetView automated operator (autotask), the result of SAY or TRACE is not visible unless the autotask is assigned an MVS console ID. The results are visible 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 execution, use the &CONTROL statement. If your command list is being executed by a NetView automated operator (autotask), the results of &WRITE, &BEGWRITE, or &CONTROL are not visible unless the autotask is assigned an MVS console ID. The results are visible 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 ID.

Recovering from Looping Command Lists

If you write a command list that is driven by a message issued by a command in the same command list, looping can occur. If a looping message-driven REXX command list contains an instruction which causes a wait, issue the STACK command from the operator's console to recover. Then turn off NetView automation with the command AUTOTBL OFF. If there is no instruction which causes a wait, you can issue the AUTOTBL OFF command from your terminal. Once the looping stops, you can revise the command list.

The REXX wait command is WAIT.

The NetView command list language wait statements are &WAIT and &PAUSE.

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 &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 invoked 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, NetView does not invoke automation for that message. If a message is assigned to SYSOP or LOG as the primary receiver, NetView does not invoke automation for that message. 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 a multiline write-to-operator (MLWTO) message is used to drive a command list and a WTO is issued from the command list, the WTO might or might not be displayed, depending 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.

Chapter 8. Common Operations Services Commands

This chapter describes how to use the common operations services (COS) commands.

Common Operations Service

The common operations services is a set of commands that supports and enhances the NetView program's 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 on 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 NetView

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 executed. 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 executing the command list are executed until the command list completes. This ensures that commands on the low-priority queue can be executed 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, with the exception of:

- LINKPD messages DSI533I, DSI534I, DSI535I, DSI536I, and DSI582I. These five messages are set to values you can use in the form of control and parameter variables. See "LINKPD Results" on page 111 for more information about LINKPD results.
- Responses to RUNCMD with the CLISTVAR keyword. CLISTVAR causes the responses to be stored in variables. See "RUNCMD Results" on page 112 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*. Possible values are:
- 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 executed.

DSITESTTYPE or **&DSITESTTYPE**
: Indicates the type of test data reported. Possible values are:
- BACKGROUND
- REQUESTED

DSIRESULT or **&DSIRESULT**
: Indication of the overall results of the test execution. Possible values are:
- PASSED
- FAILED
- INDETERMINATE

LINKPD Results

Results from the LINKPD command are returned in messages that 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 128. For more information about MSGVAR(1) through MSGVAR(31), see “Functions Set by MSGREAD” on page 39.

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 128. For more information about parameter variables used in command lists written in the NetView command list language, see “Parameter Variables” on page 48.

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's terminal, and a return code is set. See "Common Operations Services Return Codes" on page 110 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 results in the following:

- A return code is set (RC or &RETCODE); see "Common Operations Services Return Codes" on page 110 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.

  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 of 001–998.

  DSIRUN*xxx* or **&DSIRUN***xxx*
  : Contains the different responses from the COS application. The responses are numbered from 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 execute your RUNCMDs in a pipeline. The advantages of using pipelines are:

- You can run multiple invocations 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. NetView 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). NetView counts the commands executed 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 will respond faster than others. The number of lines stored in `RESULT.` will be found in `RESULT.0`.

For more information about using NetView Pipelines, refer to *Tivoli NetView for z/OS Customization: Using Pipes*.

Part 5. Commands, Functions, and Variables

Chapter 9. REXX Functions Provided by NetView 117
Translation Functions 118
Command List Information. 120
Cross-Domain Information Functions 125
Data Set Information Functions 126
Global Variable Information Functions 128
Message Processing Information Functions . . . 128
 MVS-Specific Message Processing Information 135
 ROUTCDE Examples 142
REXX Management Services Units (MSU) Information Functions 143
 Hardware Monitor (HMxxxxxx) Examples . . . 152
  HMASPRID 152
  HMBLKACT 153
  HMCPLINK 153
  HMEPNAU, HMEPNET, and HMFWDSNA 153
  HMEPNETV 154
  HMEVTYPE 154
  HMFWDED 154
  HMGENCAU 154
  HMONMSU 155
  HMORIGIN 155
  HMSECREC 155
  HMSPECAU 155
  HMUSRDAT 156
 MSUSEG Syntax and Examples 156
  Syntax. 156
  Examples. 157
Operator Information Functions 158
Session Information Functions. 158
REXX Environment Information Functions . . . 163
Terminal Information Functions 163
Time and Date 164
Nulls and Blanks Stripping. 164

Chapter 9. REXX Functions Provided by NetView

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 as part of the NetView program so that command lists and Data REXX files written in REXX can perform specific NetView activities. Because these functions are provided by NetView and are not standard REXX functions, you can use them only in command lists and Data REXX files that execute in a NetView environment. These functions do not execute in any REXX execs that are run in non-NetView environments.

You can improve the performance of your REXX command list by limiting the use of REXX functions provided by NetView. If the same function, provided by NetView, is used several times in the command list without a change in value, use the function once to set a local variable to the function's returned value. You can then use the local variable in place of the function. If the value returned by the function might change during execution of the command list, you will need to 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, NetView returns the current value of the function's related system information.

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

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

See "Appendix A. Comparison of REXX and NetView Command List Language" on page 169 for a complete list of the REXX functions that are equivalent to NetView command list language control variables. This list includes both functions provided by NetView and functions provided by REXX itself.

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 alphabetical 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 REXX function exists 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 will contain only the differences between the two versions.

3. REXX functions provided for use by the NetView program can be used only with NetView. These functions are not supported by the REXX interpreter and cannot be used in REXX EXECs executed in a non-NetView environment.
4. Not all NetView REXX functions can 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 11 on page 129, Table 12 on page 136, and Table 13 on page 144 return a value consistent with no message to process when used in Data REXX files.

Translation Functions

Table 6. Translation Functions

Function or Variable	Description
CODE2TXT(*table,code***)**	Provides translation for various types of code points to national language text. You can use NetView 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 syntax of the CODE2TXT function follows: **CODE2TXT** `►►──CODE2TXT(table,code)──────────────────────────────────►◄`

Table 6. Translation Functions (continued)

Function or Variable	Description
CODE2TXT(*table,code***)** *(continued)*	***Where:*** *code* Indicates the code point to translate. This field should be 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 *Tivoli NetView for z/OS Customization Guide* for more information. *table* Specifies the name of the table to use in the translation. The following are valid tables: **Table** **Description** **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 will translate code point `362B` in the SNAALERT table to “TRANSMITTER FAILURE”. ***Error Processing:*** Error conditions encountered by this function are handled as follows: • Non-valid operand: If a non-valid operand (such as a non-valid table name) is detected, NetView 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, NetView 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 6. Translation Functions (continued)

Function or Variable	Description
SUBSYM()	Returns a literal or variable character string (any character string that has multiple MVS system symbolics or a single MVS system symbolic imbedded in it) with the MVS system symbolics replaced within that string. Substitution is always performed on the &DOMAIN symbolic, unless substitution was disabled when NetView was started. For MVS and user-defined system symbolics, substitution is not performed if you are not running on an MVS system; you are running on an MVS system prior to MVS Version 5 Release 2; substitution was disabled when NetView was started; or you have not defined an MVS system symbolic on your MVS system. An example using SUBSYM to find out the name of the &DOMAIN follows: `SUBSYM('&DOMAIN')`

Command List Information

Table 7. Command List Information

Function or Variable	Description
| **AUTBYPAS**	For information on this function, refer to the *Tivoli NetView for z/OS Security Reference*.

Table 7. Command List Information (continued)

Function or Variable	Description
AUTHCHK(*keyword***=***value***)**	This REXX-only function enables you to make a command security check for keywords and values from a REXX program. This enables you to check the parameters that are passed to the command list, or any other items you would like to check as keywords and values related to this command list. The syntax of the AUTHCHK function is as follows:

```
                   ┌─,──────────────────┐
                   ▼                    │
►►──AUTHCHK (──┬─keyword=value─┬──┴─)──────────────────────────────►◄
               └─keyword───────┘
```

Where:

keyword
: Specifies the keyword to be authority checked. Each *keyword* can contain a maximum of 8 characters. A maximum of twenty keywords with optional values can be passed to the program. Because variable substitution could yield a null keyword, AUTHCHK will accept a null keyword. For example, `AUTHCHK()` is a valid invocation 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 *keyword*=.

  Because variable substitution could yield a null value, AUTHCHK will accept a null value and strip the '=' sign to yield a 'keyword only' security check. For example, after variable substitution, this is a valid invocation of the AUTHCHK function: `AUTHCHK(`*`keyword1=`*`,`*`keyword2=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:
     - The value of a single variable
     - Two variables with '=' in between. The = sign must be enclosed in single quotes.
     - Two literal strings with '=' in between. The = sign must be enclosed in single quotes.
     - A literal and a variable with '=' in between. The = sign must be enclosed in single quotes.

Table 7. Command List Information (continued)

Function or Variable	Description

AUTHCHK() *(continued)*	If all *keywords* 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 executed by entering `NVRXCMD START,LU=LU200`, authority checking of the keywords START and LU=LU200 can be done by coding the following 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` displays 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` displays and NVRXCMD ends.
	
	For information about keyword security, refer to the RACF library and the *Tivoli NetView for z/OS Administration Reference*.

Table 7. Command List Information (continued)

Function or Variable **Description**

AUTHCHKX(*command*,*keyword*=*value***)**

This REXX-only function enables you to make a command security check for a command and it's associated keywords and values from a REXX program. This enables you to check a command and any keywords, keywords and values or any other items you would like to check as keywords and values related to the command. The syntax of the AUTHCHKX function follows:

```
                        ┌─,──────────────────┐
                        ▼                    │
►►──AUTHCHKX (command,──┬─keyword=value─┬────┴─)──────────────────────────►◄
                        └─keyword───────┘
```

Where:

command
: Specifies the command to be used for the authorization checks. This is a required parameter. If a null command is passed, a syntax error will occur. Each command can contain a maximum of 8 characters.

keyword
: Specifies the keyword to be authority checked. Each *keyword* can contain a maximum of 8 characters. A maximum of nineteen keywords with optional values can be passed to the program. Because variable substitution could yield a null keyword, AUTHCHKX will accept a null keyword. For example, `AUTHCHKX(command)` is a valid invocation of the AUTHCHKX function. When a null keyword is passed to the AUTHCHKX function, authority is assumed to be granted for the command that was passed in the function call.

value
: Specifies a value for the keyword. Each *value* can contain a maximum of 8 characters. You cannot specify *value* without *keyword*=.

: Because variable substitution could yield a null value, AUTHCHKX will accept a null value and strip the '=' sign to yield a 'keyword only' security check. For example, after variable substitution, this is a valid invocation of the AUTHCHKX function:

```
AUTHCHKX(command,keyword1=,keyword2=value2,keyword3,keyword4=value4)
```

: *Keywords* and *keyword=value* combinations must be separated by commas.

Table 7. Command List Information (continued)

Function or Variable	Description

AUTHCHKX()*(continued)*	**Usage notes:**
	1. This function does not perform any syntax checks of the command, keywords or values. It can only determine if a combination of command, keyword and/or value is protected using any NetView security facility.
	2. If the keyword and value are both null, the null string is returned, which implies that authority is granted.
	3. *keyword=value* can be any of the following:
	• The value of a single variable
	• Two variables with '=' in between. The = sign must be enclosed in single quotes.
	• Two literal strings with '=' in between. The = sign must be enclosed in single quotes.
	• A literal and a variable with '=' in between. The = sign must be enclosed in single quotes.
	4. If all keywords or keyword=value combinations in the list pass authority checking, AUTHCHKX 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 the remaining keywords are not checked.
	For example, if a NetView command list language program NVCLCMD needed to perform authority checks on two parameters passed to it, it could call REXX program NVCHKAUT as: NVCHKAUT NVCLCMD, &P1, &P2. NVCHKAUT can be coded as follows:
	`/* NVCHKAUT:  SAMPLE REXX PROGRAM    */`
	`PARSE ARG P1','P2','P3','.`
	`RESULT=AUTHCHKX(P1,P2,P3)`
	`...`
	`IF RESULT ¬= '' THEN`
	`  DO`
	`  SAY OPID() ' IS NOT AUTHORIZED TO ISSUE 'P1' WITH 'RESULT`
	`  Return_Code = 8`
	`END`
	`...`
	`Return Return_Code`
	In this example, if either of the parameters passed in P2 and P3 does not pass authority checking with the command passed in P1, a non–null value is returned by AUTHCHKX. If a keyword fails, it is included in a message along with the command that was passed, 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, 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, NVCHKAUT returns a return code of 8 to NVCLCMD and NVCLCMD will terminate.
	For information about keyword security, refer to the RACF library and the *Tivoli NetView for z/OS Security Reference*.

CMDNAME()	This REXX-only function returns the name by which the program was called. This name may 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.

Table 7. Command List Information (continued)

Function or Variable	Description
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 there are no parameter variables, 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 there are no parameter variables, &PARMSTR is null. The maximum length of the string returned by &PARMSTR is 255 characters.
&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 79. On the &EXIT statement, you can set the return code to 0, −1, or a positive integer. NetView can set the return code to 0, −1, −2, −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 possible values and meanings of &RETCODE are as follows: **Value** **Meaning** *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 terminated as the result of a RESET or other failure.

Note: &RETCODE is similar to the REXX-provided RC variable.

Cross-Domain Information Functions

Table 8. 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 8. 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 CNMSTYLE. 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 *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 3-character alphanumeric value assigned by the 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 9. 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 executed. The allocated files must be a partitioned dataset (PDS). FNDMBR returns two determinant results, and a variety of indeterminate results which you may use to debug your REXX program. The arguments of the FNDMBR function are defined as follows: *DD_name* Specifies the DD name by which the allocated PDS file is known to NetView. *member_name* Specifies the name of the member of the allocated PDS file to be located.

Table 9. Data Set Information Functions (continued)

Function or Variable	Description
FNDMBR (*DD_name***,***member_name***)** *(continued)*	The possible results returned by the FNDMBR function are: **Value** **Meaning** **0** The specified member name was found. **4** The specified member name was not found. **100** 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 may mean that the DD statement is missing or the file is not allocated. The following is an example of the FNDMBR function usage: `IF FNDMBR('DSICLD','MYREXX') = 0 THEN` This REXX statement will evaluate as true if MYREXX exists in DSICLD or as false if MYREXX does not exist in DSICLD. **Notes:** 1. Refer to theMVS/DFP library for the OPEN macro return codes. Refer to theMVS/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 invoking the FNDMBR function. 3. If the NetView ALLOCATE command was used to allocate the dataset, 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 executes an MVS OPEN and CLOSE which will cause 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 dataset may be allocated down to the member level. However, it is not recommended because an IEC141I message will be issued if the member is not found.

Global Variable Information Functions

Table 10. 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 96. **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 96. **End of NetView Command List Language**

Message Processing Information Functions

Table 11 on page 129 lists functions and variables that, unless otherwise noted, are available for use on messages generated by all operating system platforms supported by NetView.

Table 12 on page 136 lists only those functions and variables that are available for use solely for messages originating from MVS systems.

In all cases, 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.

Table 11. Message Processing Information

Function or Variable	Description
ACTIONDL() **&ACTIONDL**	Returns the reason for which NetView deleted the associated message. Values are: **Value** **Meaning** **(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 NetView. **SMSGID** Deletion by MVS DOM using SMSGID. **TOKEN** Deletion by MVS DOM using TOKEN. **TCB** Deletion by MVS DOM for task end. **ASID** Deletion by MVS DOM for address end space. **INVALID** The DSIIFR has a combination of control flag settings that are not valid.
ACTIONMG() **&ACTIONMG**	Returns ′1′ if the message is an action message. Returns ′0′ if otherwise.
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.
EVENT()	The NetView event that satisfied the WAIT 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 possible returned values from EVENT() are as follows: **Value** **Meaning** **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. 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 *Tivoli NetView for z/OS Customization: Using Assembler*.

Table 11. Message Processing Information (continued)

Function or Variable	Description
IFRAUGMT() **&IFRAUGMT**	Returns the Greenwich mean time when the automation internal function request (AIFR) was created. IFRAUGMT is returned as an 8-byte hexadecimal value in store clock format.
IFRAUIND() **&IFRAUIND**	Returns two 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 bit positions are: **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** Message was from down-level domain (no AIFR received). **16** Message was unsolicited. **Notes:** 1. Other bits can be tested, but have no recommended use. All the bits are defined in the DSIIFR mapping control blocks. For more information, refer to the *Tivoli NetView for z/OS Customization: Using 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. When using extended consoles, only MVS system messages that are received by the task with load module name CNMCSSIR are considered unsolicited messages. 5. For more information about solicited and unsolicited messages, refer to the *Tivoli NetView for z/OS Automation Guide*.
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 bit positions and meanings are: **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.

Table 11. Message Processing Information (continued)

Function or Variable	Description
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.
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 bit positions are: **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. **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 recommended use. 2. Refer to the description of DSIIFR fields IFRAUTA1 through IFRAUTA6 in *Tivoli NetView for z/OS Customization: Using 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. Specific bit positions with recommended uses are: **Position** **Meaning** **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 recommended use. MLWTO flags in this area also have no recommended use. MLWTO indicators are moved into the data buffers.

Table 11. Message Processing Information (continued)

Function or Variable	Description
LINETYPE() **&LINETYPE**	Returns the multiline write-to-operator (MLWTO) line type or MSU data buffer type, as follows: **Type** **Description** **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** There is no message or MSU data buffer associated with this command list
MSGID() **&MSGID**	**REXX** **REXX Function:** Returns the message identifier of the message that drove this command list. The message identifier is generally the first token of the message (up to 255 characters). If the first token is longer than 255 characters, MSGID returns only the first 255 characters. If a reply ID is sent with the message, it is not used as the first token. For a multiline write-to-operator (MLWTO), MSGID uses the first token of the first line of the message. When an MSU buffer is being processed, MSGID is equal to null (''). See “Chapter 7. Automation Resource Management” on page 105 for more information about NetView automation. See “Control and Parameter Variables Used with &WAIT” on page 87 for more information about using control variables with &WAIT. See “LINKPD Results” on page 111 for more information about the LINKPD command. **Note:** For messages received over the VM PROP/PMX interface, MSGID may not be set to the actual message identifier because of information added to the front of the message. MSGID() is used in 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** **NetView Command List Language Control Variable (differences from REXX only):** &MSGID is used in NetView automation, with &WAIT, and with the LINKPD command. **End of NetView Command List Language**

Table 11. Message Processing Information (continued)

Function or Variable	Description
MSGORIGN() **&MSGORIGIN**	**REXX** **REXX Function:** 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** **NetView Command List Language Control Variable (differences from REXX only):** Specifies the domain where the message originated. &MSGORIGIN is used in NetView automation, with &WAIT, and with the LINKPD command. See “Chapter 7. Automation Resource Management” on page 105 for more information about NetView automation. See “Control and Parameter Variables Used with &WAIT” on page 87 for more information about using control variables with &WAIT. See “LINKPD Results” on page 111 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.

Table 11. Message Processing Information (continued)

Function or Variable	Description
MSGSTR() **&MSGSTR**	**REXX** **REXX Function:** Returns the message text of the last message read by MSGREAD. MSGSTR() does not include the message identifier—the token used by the MSGID() function. For an MLWTO message, MSGSTR() becomes the message text of the first line of the message. MSGSTR() is used with MSGREAD and with the LINKPD command. Refer to the NetView online help for more information about using functions with MSGREAD. **End of REXX** **NetView Command List Language** **NetView Command List Language Control Variable (differences from REXX):** 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 87 for more information about using control variables with &WAIT. See "LINKPD Results" on page 111 for more information about the LINKPD command. **End of NetView Command List Language**
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 11. Message Processing Information (continued)

Function or Variable	Description
MSGVAR(*n*)	Returns the text of a message. NetView 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. See "Chapter 7. Automation Resource Management" on page 105 for more information about NetView automation. Refer to the NetView online help for more information about using functions with MSGREAD. See "LINKPD Results" on page 111 for more information about the LINKPD command. The MSGVAR(*n*) functions can be given values when a command list is invoked in the same way as are the &1-&31 NetView command list language parameter variables.
MVSRTAIN() **&MVSRTAIN**	In the Automation Table, a 3 bit field describing MVS Retain Charateristics 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 NetView 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.
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 will be the message reply ID only if the sending system uses those characters to designate a reply ID for a message.
SESSID() **&SESSID**	Returns the 1–8 character ID of the TAF (terminal access facility) session that sent the message. See "Chapter 7. Automation Resource Management" on page 105 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 may give unpredictable results.

MVS-Specific Message Processing Information

This section contains descriptions of message information functions that are important for NetView message automation. This information is generated by MVS systems, but can be seen as cross-domain data in VM and VSE systems as well.

Message Processing

You can also generate user messages with this data as long as the information is consistent with the MVS definitions. You can find more information about these functions in the MVS/ESA™ library.

Some of these message information functions and control variables return valid meaningful values only if the message currently being processed was originally a message data block (MDB). Where applicable, this is noted in Table 12 on page 136.

Table 12. MVS-Specific Message Processing Information

Function or Variable	Description
AREAID() **&AREAID**	Returns a 1-letter (A-Z) identifier for the area on the multiple console support console panel that displays the message.
AUTOTOKE() **&AUTOTOKE**	Returns the 1–8 character name of the MVS message processing facility (MPF) automation token. **Note:** If you've specified `AUTO(YES)` or `AUTO(NO)` in the MPF table, the values `YES` and `NO` are not automation tokens.
CART() **&CART**	Returns the 8-byte MVS command and response token (CART). The CART might contain non-displayable characters. **Notes:** 1. This function has a value only if the message currently being processed was originally a message data block (MDB). 2. CART is available on systems running MVS 4.1 or a later release.
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.
JOBNAME() **&JOBNAME**	Returns the 1–8 character MVS job name identifier. Because the 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 terminated. If the message is issued during startup or termination, extract the job name from the message text rather than using the JOBNAME function. **Note:** The same information is available using MSGCOJBN (on page 138).
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. **Note:** This function has a value only if the message currently being processed was originally an MDB.

Table 12. MVS-Specific Message Processing Information (continued)

Function or Variable	Description
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 bit positions and their meanings are: **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. **Value** **Meaning** **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. Bit positions and their meaning are: **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). **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB. 2. Other bits can be tested, but have no recommended 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. Bit positions and their meaning are: **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. **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB. 2. Other bits can be tested, but have no recommended use.

Table 12. MVS-Specific 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. Bit positions and their meaning are: **1** WTOR **2** Immediate action **3** Critical eventual action **4** Eventual action **5** Informational **6** Broadcast **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB. 2. Other bits can be tested, but have no recommended 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. Bit positions and their meaning are: **1** Display job names **2** Display status **3** Monitor active **6** Monitor SESS **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB. 2. Other bits can be tested, but have no recommended use.
MSGCNT() **&MSGCNT**	**REXX** **REXX Function:** Returns the number of words in the message string of the last message read by MSGREAD. MSGCNT() is used 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 111 for more information about the LINKPD command. **End of REXX** **NetView Command List Language** **NetView Command List Language Control Variable (differences from REXX only):** Returns the number of words in a message string. &MSGCNT is used with &WAIT and with the LINKPD command. See “Control and Parameter Variables Used with &WAIT” on page 87 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, on page 136.) **Note:** This function has a value only if the message currently being processed was originally a message data block (MDB).

Table 12. MVS-Specific Message Processing Information (continued)

Function or Variable	Description
MSGCPROD() **&MSGCPROD**	Returns the 16-character MVS product level. The characters are defined as follows: • 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. **Note:** This function has a value only if the message currently being processed was originally an MDB.
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 above. **Note:** This function returns a value only if the message currently being processed was received from an MVS SYSPLEX, and the message was originally a message data block (MDB).
MSGCSYID() **&MSGCSYID**	Returns the 1–3 digit decimal number system identification for DOM. **Note:** This function has a value only if the message currently being processed was originally an MDB.
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. Bit positions and their meaning are: **1** DOM by message ID **2** DOM by system ID **3** DOM by ASID **4** DOM by job step TCB **5** DOM by token **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB.

MSGGBGPA()
 	
&MSGGBGPA
	Returns the 4-byte hexadecimal background presentation attributes. Bytes and their descriptions are:
	
	Byte **Description**
	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
	
	Note: This function has a value only if the message currently being processed was originally an MDB.

Table 12. MVS-Specific Message Processing Information (continued)

Function or Variable	Description
MSGGDATE() **&MSGGDATE**	Returns the message date in a 7-character format of *yyyyddd*, where *yyyy* is the year and *ddd* indicates a calendar day. **Notes:** 1. This is not necessarily the current date. It may be the date with which MVS associates the message as having been issued. 2. This function has a value only if the message currently being processed was originally an MDB.

MSGGFGPA()
 	
&MSGGFGPA
	Returns the 4-byte hexadecimal foreground presentation attributes. Bytes and their meaning are:
	
	Byte **Description**
	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
	
	Note: This function has a value only if the message currently being processed was originally an MDB.

MSGGMFLG() **&MSGGMFLG**	Returns the 16-bit MVS general message flags. Bit positions and their meaning are: **1** DOM (delete operator message) **Note:** This function has a value only if the message currently being processed was originally an MDB.
MSGGMID() **&MSGGMID**	Returns the 4-character hexadecimal value MVS message identifier field. **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB. 2. MSGGMID represents the same information as SMSGID, except that SMSGID returns a decimal value instead of a hexadecimal value.
MSGGSEQ() **&MSGGSEQ**	Returns the 1- to 8-character numerical decimal sequence number. This function represents the last three bytes of MSGGMID. **Note:** This function has a value only if the message currently being processed was originally an MDB.
MSGGSYID() **&MSGGSYID**	Returns the 1- to 3-character numerical decimal system identification. This is the first byte of MSGGMID. **Note:** This function has a value only if the message currently being processed was originally an MDB.

Table 12. MVS-Specific Message Processing Information (continued)

Function or Variable	Description
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. **Note:** This function has a value only if the message currently being processed was originally an MDB.
MSGSRCNM() **&MSGSRCNM**	Returns the 1- to 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) invocation. For more information about DSIMMDBS, refer to *Tivoli NetView for z/OS Customization: Using Assembler*. For more information about CNMPMDB, refer to *Tivoli NetView for z/OS Customization: Using 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 there is no source object For more information about how the source object is defined, refer to the DSIAIFRO mapping in *Tivoli NetView for z/OS Customization: Using Assembler*. **Note:** This function has a value only if the message currently being processed was originally an MDB with an associated source object.
MSGTOKEN() **&MSGTOKEN**	Returns a 1–10 digit decimal number that indicates the token associated with the message. **Notes:** 1. This function has a value only if the message currently being processed was originally an MDB. 2. You can use a TOKEN value to group WTOs by setting MSGTOKEN prior to issuing the WTO command. Subsequently, these messages can be deleted using a single DOM command by specifying the token value in MSGTOKEN.
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: **Bit** **Description** **1** SESS — Corresponds to IFRAUWF1(14) **2** JOBNAMES — Corresponds to IFRAUWF1(9) **3** STATUS — Corresponds to IFRAUWF1(10)
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. **Note:** This function has a value only if the message currently being processed was originally an MDB.

Table 12. MVS-Specific 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 will be the lowest multiple of 8 that contains one or more on (1) characters. Therefore, you should 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 sixteen zeros, a 1, and seven more zeros will be returned (000000000000000010000000). One method to test for bit 17 being on is shown in the REXX example in Figure 28 on page 142. The functionally equivalent code written in NetView command list language is shown in Figure 29 on page 142. 2. Another method to check for a specific bit is to use the REXX environment's POS (position) function, as shown in Figure 30 on page 143. 3. For details on using the REXX POS function, refer to the REXX library.
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 or console ID associated with the message. System console names are 2–8 characters in length; system console IDs are 2-digit decimal numbers.
SYSID() **&SYSID**	Returns the 1–8 character identifier of the MVS system from which a message arrived.
WTOREPLY **&WTOREPLY**	Returns an operator's 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() = '000000000000000010000000'
   THEN SAY 'ROUTCDE BIT 17 IS SET.'
```

Figure 28. REXX Example to Test for Bit 17

```
&IF &ROUTCDE = 000000000000000010000000 &THEN
 &WRITE ROUTCDE BIT 17 IS SET
```

Figure 29. 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 30. Using the REXX POS Function to Test for Bit 17

REXX Management Services Units (MSU) Information Functions

Table 13 on page 144 lists REXX functions for 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 *Tivoli NetView for z/OS Automation Guide*.

The following terms are used in Table 13 on page 144:

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's 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's that contain only statistical data are those with RECFMS Type (byte 8, 1-offset, into the RECFMS) 1, 4, and 5.

MSU Information

Table 13. 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 than that found in BNJ146I messages. The following are 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 there is no name/type pair that corresponds to *n*, then a null value is returned. 3. HIER(*n*) returns null under the following conditions: • If the command list is not executed 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. See Table 13 on page 144 for more information about the HMSECREC function. 5. If a complex link exists in a resource hierarchy, there might be resource levels that do not appear 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 13 on page 144 for more information about the HMCPLINK function. 6. For information about the NetView built-in function &HIER, see “&HIER” on page 66.
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 either a: • 1–4 character hardware product ID, or • 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 152.

Table 13. 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: • 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 • Not submitted to the automation table by the hardware monitor 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 153.
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 do not appear 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 144 for more information. Hardware monitor panels, such as Most Recent Events, indicate 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 153.

Table 13. 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 may 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 *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 153.
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 may 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 *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 153.

Table 13. 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 *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 154.

HMEVTYPE()	Returns the event type of an MSU. Any trailing blanks in the event type are removed. The event types include:
	`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, refer to the NetView online help
	HMEVTYPE returns null if an MSU is:
	• A PD statistic (X'0025')
	• A link configuration data (X'1332')
	• A statistics-only RECMS
	• A statistics-only RECFMS
	• Not submitted to automation by the hardware monitor
	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 154.

Table 13. 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 through the NV-UNIQ/LUC alert forwarding protocol. **0** An MSU was not forwarded from another NetView, or was forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol. Examples of when a 0 would be 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 **Notes:** 1. RECMSs and RECFMSs 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 receiving focal point's hardware monitor. These RECMSs and RECFMSs can only be automated by the sending entry point NetView program. 2. Refer to the *Tivoli NetView for z/OS Automation Guide* for more information about forwarding mechanisms. 3. The maximum length is 1 character. 4. HMFWDED applies to all MSUs submitted to automation by the hardware monitor. 5. See the examples in "HMFWDED" on page 154.
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. **Note:** Refer to the *Tivoli NetView for z/OS Automation Guide* book for more information about forwarding mechanisms. The maximum length is 1 character. HMFWDSNA applies only to MSUs submitted to automation by the hardware monitor. HMFWDSNA returns null for all other MSUs. See the example in "HMEPNAU, HMEPNET, and HMFWDSNA" on page 153.

Table 13. 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: • 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 • Not submitted to the automation table by the hardware monitor Otherwise, a general cause code is returned. Examples of MSUs that HMGENCAU 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 1 hexadecimal character. HMGENCAU applies to all MSUs submitted to automation by the hardware monitor. See the examples in “HMGENCAU” on page 154.
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 155.

Table 13. 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 *Tivoli NetView for z/OS Administration Reference* for a reference to 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 (145) and the description of HIER (144). 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 155.
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 (144) 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 155.

Table 13. 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 RECMSs and RECFMSs. HMSPECAU returns null if an MSU is: • 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 • Not submitted to the automation table by the hardware monitor Otherwise, a general cause code is returned. Examples of MSUs that HMSPECAU 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 2 hexadecimal characters. HMSPECAU applies to all MSUs submitted to automation by the hardware monitor. See the examples in “HMSPECAU” on page 155.

Table 13. 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 may 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: • 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 156.
MSUSEG(*operands*)	Provides the parsing capability needed to extract information from a management services unit (MSU) or other similarly architected pieces of data. Use this function in a command list that is invoked 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 156. For information about the built-in function &MSUSEG, see “&MSUSEG” on page 70.

Hardware Monitor (HMxxxxxx) Examples

HMASPRID

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

Figure 31. HMASPRID Example A

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

Figure 32. 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 33. 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 34. 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 35. HMBLKACT Example C

HMCPLINK

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

Figure 36. HMCPLINK Example A

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

Figure 37. 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 38. HMEPNAU, HMEPNET, and HMFWDSNA Example A

HMEPNETV

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

Figure 39. HMEPNETV Example A

HMEVTYPE

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

Figure 40. 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 41. 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 42. 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 43. HMFWDED Example B

HMGENCAU

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

Figure 44. HMGENCAU Example A

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

Figure 45. 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 46. 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 47. HMONMSU Example B

HMORIGIN

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

Figure 48. 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 49. HMSECREC Example

HMSPECAU

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

Figure 50. HMSPECAU Example A

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

Figure 51. 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 ....
```

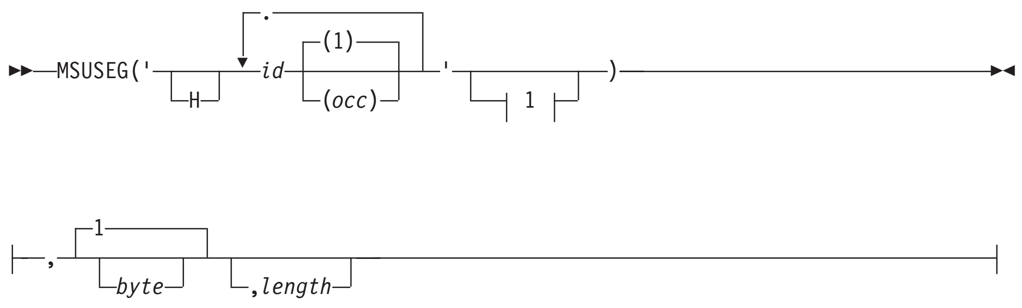
Figure 52. HMUSRDAT Example

MSUSEG Syntax and Examples

Syntax

The MSUSEG(*operands*) syntax is:

MSUSEG



Where:

byte
: The byte position into the lowest ID specified in *id*, counting from 1 in decimal. 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 quotes when quotes 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) 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 quotes shown in the REXX syntax diagram are only required when an *occ* is specified. If you do not explicitly code an *occ*, the quotes 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 executed by an automation table statement due to 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. f 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 156.
7. For more information about the automation table, refer to *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 *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 70.

Examples

The following are examples of using the MSUSEG() function:

In Figure 53, 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 53. MSUSEG() Example 1

In Figure 54, Alert subvector A0 has `'LINE'` followed by `'DOWN'` anywhere in it. Literals can be in hex as well as EBCDIC.

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

Figure 54. MSUSEG() Example 2

In Figure 55, 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 55. MSUSEG() Example 3

MSU Information

Figure 56 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 56. MSUSEG() Example 4

Operator Information Functions

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

Table 14. Operator Information Functions

Function or Variable	Description
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: **O** Returns the owner's identity. On a regular OST, it will be 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 executing. Special values, other than operator ID might be returned as follows: **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 be ignored. **Note:** There is currently no case where NetView invokes 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 executing.

Session Information Functions

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

Table 15. Session Information

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 NetView. For example, if your domain ID is PARIS, APPLID might be PARIS001. NetView 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 may be in this situation.
ASID() **&ASID**	Returns the current NetView address space identifier. The value of ASID is a 1–5 digit decimal number.

Table 15. Session Information (continued)

Function or Variable	Description
ATTENDED() **&ATTENDED**	Returns a single-character value of either 1 or 0. The values are defined as: **1** Indicates that the task is one of the following: • 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: • An autotask without an associated MVS console assigned using the AUTOTASK command • Another type of task, such as a DST or an OPT task **Usage Notes:** 1. If the associated operator is an AUTOTASK, the presentation data will not be 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 or console ID of the MVS console where NetView commands can be entered to run under this autotask. **Note:** MVS console names are available beginning with MVS 4.1.0.
AUTOTASK() **&AUTOTASK**	Returns a single-character value of either 1 or 0 indicating whether or not the task is an autotask. Values are: **1** An autotask **0** Not an autotask
CGI()	Returns a single-character value of either 1 or 0. Values are: **1** The procedure was invoked by the NetView Web server. **0** The procedure was not invoked by the NetView Web server.
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 or console ID of the MVS console that this task uses to enter MVS commands. **Note:** MVS console names are available on systems running MVS 4.1 or later.
CURSYS() **&CURSYS**	Returns the 1–8 character current system name.
DISC() **&DISC**	Returns a single-character value of either 1 or 0 that indicates whether the task is disconnected. Values are as follows: **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. Values are: **1** A distributed autotask **0** Not a distributed autotask **Note:** This corresponds to the value of TVBDAUT.

Table 15. Session Information (continued)

Function or Variable	Description
DOMAIN() **&DOMAIN**	Returns the 1- to 5-character name of the current NetView domain.
DOMAIN('*x*')	Returns the 1- to 5-character name of a NetView domain, where: **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.
ENVDATA('*x*')	Returns a numeric value or character string, where: **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 which was active when the ENVDATA was invoked. Each entry consists of two names seperated by a slash (/), in the format: *command/name*. The *command* is the command verb or synonym used to call the procedure. The *name* is one of the following: • 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.
MVSLEVEL() **&MVSLEVEL**	Returns the currently running version of MVS. For example, if you are running MVS/ESA 4.2.2, MVSLEVEL returns `SP4.2.2`.
NETID() **&NETID**	The VTAM network identifier. This field has a maximum length of 8 characters. If VTAM has never been active when NetView is active, the value of NETID is null.
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: **NV** Indicates NetView *v* Indicates the version number of NetView *r* Indicates the release number of NetView
NETVIEW('*x*')	Returns a text string, where: **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. OPSYSTEM can contain the following character values: MVS/ESA MVS/XA™ VM/ESA® VSE
PANEL()	Returns a single-character value of either 1 or 0. Values are: **1** Panel commands can be issued **0** Panel commands not allowed
PARTID() **&PARTID**	Returns the first two characters of the six-character prefix for VSE messages. The two returned characters will be the message partition ID only if the sending system uses those characters to designate a partition ID for a message.

Table 15. Session Information (continued)

Function or Variable	Description
STCKGMT() **&STCKGMT**	Returns the current Greenwich 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 NetView from writing the command out to the terminal, hard-copy log, and network log.) SUPPCHAR is a single character that you define in the CNMSTYLE member of DSIPARM. The default suppression character that is shipped with the NetView product is the question mark (X'6F'). If you do not specify a suppression character in CNMSTYLE, SUPPCHAR defaults to X'3F'. **Note:** The SUPPCHAR default character of X'3F' cannot be typed at the operator's 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 executing. Available when running under MVS/ESA Version 4 Release 2.2 or above. **Note:** This function returns a value only if the command list is executing on an MVS SYSPLEX.
TASK() **&TASK**	Returns the 3-character string indicating the type of task under which the command list is running. Possible values are: **PPT** Primary POI Task **OST** Operator Station Task **NNT** NetView-to-NetView Task For Data REXX, in addition to PPT, OST, and NNT, any the following can 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 Tivoli Customer 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, there are some restrictions for command lists running under the PPT. See "Primary POI Task Restrictions" on page 15.

Table 15. Session Information (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. Values are: **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 used either alone, or used together with a tower name to determine the subtowers that are enabled. For example, assume that the AON tower and the SNA and TCP subtowers are enabled, `SAY TOWER('*')` would return `AON SNA TCP` and `SAY TOWER('aon.*')` would return `SNA TCP`. **Notes:** 1. Input strings are not case sensitive and mixed case strings can be returned. 2. Tower and subtower combinations must be concatenated with a period (.). 3. Towers and subtowers are enabled in CNMSTYLE. Refer to the comments in CNMSTYLE for more information.
TYPE()	Returns a 3-character string that indicates the level of NetView that is installed. Possible values are: **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 VTAM component identifiers are: **MVS/ESA** 5685-08501-*xxx* (for VTAM Version 3) **MVS/ESA** 5695-11701-*xxx* (for VTAM Version 4) **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 may be added in future updates to VTAM. **Note:** The value of VTCOMPID is null if VTAM is not active.

Table 15. Session Information (continued)

Function or Variable	Description
WEEKDAYN() **&WEEKDAYN**	Returns a numeric value in the range of 1 to 7 indicating the day of week (from Monday through Sunday), as shown below: **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 NetView.

Refer to the *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 16. 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 executing. 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 17. Terminal Information Functions

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

Table 17. Terminal Information Functions (continued)

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

Time and Date

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

Table 18. Date and Time

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 CPU time in the format *hh:mm*, where *hh* is the hour and *mm* is the minutes. The time is based on a 24-hour clock, so 3:00 p.m. is shown as 15:00.

Note: Because &TIME and &DATE are separate variables, you may 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, so you would have the wrong time for the current date.

The following is an example of some NetView command list language code that can help you 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 NetView 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 (when the value is a name)
SYSID(), &SYSID	None
SYSPLEX(), &SYSPLEX	Nulls and blanks

Part 6. Appendixes

Appendix A. 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 alphabetical sequence based on the name of the NetView command list language control statement.

The last column of the table indicates whether the corresponding REXX instruction is a standard instruction provided by REXX or an instruction provided by the NetView program.

Instructions provided by the NetView program can be used only in conjunction with NetView. These instructions are not supported by the REXX interpreter and cannot be used in REXX EXECs executed in a non-NetView environment.

Table 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	59	N/A
CGLOBAL(*name*)	128	&CGLOBAL	97	NetView
TRACE	32	&CONTROL	56	REXX
EXIT	79	&EXIT	79	REXX
SIGNAL	33	&GOTO	79	REXX
IF	77	&IF	77	REXX
PARSE EXTERNAL	26	&PAUSE	61	REXX
PARSE PULL	26	&PAUSE	61	REXX
TGLOBAL(*name*)	128	&TGLOBAL	96	NetView
TRAP	*	&WAIT	81	NetView
WAIT	*	&WAIT	81	NetView
MSGREAD	*	&WAIT	81	NetView
FLUSHQ	*	&WAIT	81	NetView
SAY	26	&WRITE	58	REXX

***:** Refer to the *Tivoli NetView for z/OS Command Reference*.

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 in conjunction with NetView and is not supported by SAA® REXX.

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions

REXX Function	Described on	NetView Control Variable	REXX Function Provided By
ACTIONDL()	129	&ACTIONDL	NetView
ACTIONMG()	129	&ACTIONMG	NetView
APPLID()	158	&APPLID	NetView
AREAID()	136	&AREAID	NetView
ASID()	158	&ASID	NetView
ATTENDED()	159	&ATTENDED	NetView
ATTNID()	129	&ATTNID	NetView
AUTCONID()	159	&AUTCONID	NetView
AUTHCHK(...)	121	None	NetView
AUTHCHKX(...)	123	None	NetView
AUTOTASK()	159	&AUTOTASK	NetView
AUTOTOKE()	136	&AUTOTOKE	NetView
BITAND(...)	64	&BITAND	REXX
BITOR(...)	64	&BITOR	REXX
BITXOR(...)	65	&BITXOR	REXX
CART()	136	&CART	NetView
CGI()	159	None	NetView
CGLOBAL(*name*)	128, 97	&CGLOBAL	NetView
CMDNAME()	124	None	NetView
CODE2TXT(...)	118	None	NetView
||	66	&CONCAT	REXX
CURCONID()	159	&CURCONID	NetView
CURSYS()	159	&CURSYS	NetView
DATE()	164	&DATE	REXX
DESC()	136	&DESC	NetView
DISC()	159	&DISC	NetView
DISTAUTO()	159	&DISTAUTO	NetView
DOMAIN()	160	&DOMAIN	NetView
DOMAIN('x')	160	None	NetView
| ENVDATA('x')	160	None	NetView
EVENT()	129	None	NetView

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions (continued)

REXX Function	Described on	NetView Control Variable	REXX Function Provided By
FNDMBR(...)	126	None	NetView
HCOPY()	163	&HCOPY	NetView
HDRMTYPE()	129	&HDRMTYPE	NetView
HIER(*n*)	144, 66	&HIER	NetView
HMASPRID()	144	None	NetView
HMBLKACT()	145	None	NetView
HMCPLINK()	145	None	NetView
HMEPNAU()	146	None	NetView
HMEPNET()	146	None	NetView
HMEPNETV()	147	None	NetView
HMEVTYPE()	147	None	NetView
HMFWDED()	148	None	NetView
HMFWDSNA()	148	None	NetView
HMGENCAU()	149	None	NetView
HMONMSU()	149	None	NetView
HMORIGIN()	150	None	NetView
HMSECREC()	150	None	NetView
HMSPECAU()	151	None	NetView
HMUSRDAT()	152	None	NetView
IFRAUGMT()	130	&IFRAUGMT	NetView
IFRAUIND()	130	&IFRAUIND	NetView
IFRAUIN3()	130	&IFRAUIN3	NetView
IFRAUI3X()	130	&IFRAUI3X	NetView
IFRAUSDR()	130	&IFRAUSDR	NetView
IFRAUSRB()	131	&IFRAUSRB	NetView
IFRAUSB2()	130	&IFRAUSB2	NetView
IFRAUSRC()	131	&IFRAUSRC	NetView
IFRAUSC2()	130	&IFRAUSC2	NetView
IFRAUTA1()	131	&IFRAUTA1	NetView
IFRAUWF1()	131	&IFRAUWF1	NetView
JOBNAME()	136	&JOBNAME	NetView
JOBNUM()	136	&JOBNUM	NetView
KEY()	136	&KEY	NetView
LENGTH(...)	69	&LENGTH	REXX
LINETYPE()	132	&LINETYPE	NetView
LU()	164	&LU	NetView
MCSFLAG()	137	&MCSFLAG	NetView
MSGASID()	137	&MSGASID	NetView

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions (continued)

REXX Function	Described on	NetView Control Variable	REXX Function Provided By
MSGAUTH()	137	&MSGAUTH	NetView
MSGCATTR()	137	&MSGCATTR	NetView
MSGCMISC()	137	&MSGCMISC	NetView
MSGCMLVL()	138	&MSGCMLVL	NetView
MSGCMSGT()	138	&MSGCMSGT	NetView
MSGCNT()	138	&MSGCNT	NetView
MSGCOJBN()	138	&MSGCOJBN	NetView
MSGCPROD()	139	&MSGCPROD	NetView
MSGCSPLX()	139	&MSGCSPLX	NetView
MSGCSYID()	139	&MSGCSYID	NetView
MSGDOMFL()	139	&MSGDOMFL	NetView
MSGGBGPA()	139	&MSGGBGPA	NetView
MSGGDATE()	140	&MSGGDATE	NetView
MSGGFGPA()	140	&MSGGFGPA	NetView
MSGGMFLG()	140	&MSGGMFLG	NetView
MSGGMID()	140	&MSGGMID	NetView
MSGGSEQ()	140	&MSGGSEQ	NetView
MSGGSYID()	140	&MSGGSYID	NetView
MSGGTIME()	141	&MSGGTIME	NetView
MSGID()	132	&MSGID	NetView
MSGORIGN()	133	&MSGORIGIN	NetView
MSGSRCNM()	141	&MSGSRCNM	NetView
MSGSTR()	134	&MSGSTR	NetView
MSGTOKEN()	141	&MSGTOKEN	NetView
MSGTSTMP()	134	&MSGTSTMP	NetView
MSGTYP()	141	&MSGTYP	NetView
MSGVAR()	135	None	NetView
MSGVAR(*number*)	135	&1 - &31	NetView
MSUSEG(...)	152, 70	&MSUSEG	NetView
MVSLEVEL()	160	&MVSLEVEL	NetView
NVCNT()	125	&NCCFCNT	NetView
NVDELID()	135	&NVDELID	NetView
NVID(*n*)	126	&NCCFID	NetView
NVSTAT(*name*)	126	&NCCFSTAT	NetView
NETID()	160	&NETID	NetView
NETVIEW()	160	&NETVIEW	NetView
NETVIEW('x')	160	None	NetView
OPID()	158	&OPID	NetView

Table 20. Comparison of REXX Functions and NetView Command List Language Control Variables and Functions (continued)

REXX Function	Described on	NetView Control Variable	REXX Function Provided By
OPID('x')	158	None	NetView
OPSYSTEM()	160	&OPSYSTEM	NetView
PANEL()	160	None	NetView
PARMCNT()	125	&PARMCNT	NetView
ARG(1)	125	&PARMSTR	REXX
PARTID()	160	&PARTID	NetView
PRTY()	141	&PRTY	NetView
REPLYID()	135	&REPLYID	NetView
RC	125	&RETCODE	REXX
ROUTCDE()	142	&ROUTCDE	NetView
RXDEFENV()	163	&RXDEFENV	NetView
RXDEFSTR()	163	&RXDEFSTR	NetView
RXNUMENV()	163	&RXNUMENV	NetView
RXOVRENV()	163	&RXOVRENV	NetView
RXOVRSTR()	163	&RXOVRSTR	NetView
SESSID()	135	&SESSID	NetView
SMSGID()	142	&SMSGID	NetView
STCKGMT()	161	&STCKGMT	NetView
SUBSTR(...)	73	&SUBSTR	REXX
SUBSYM(...)	120	None	NetView
SUPPCHAR()	161	&SUPPCHAR	NetView
SYSCONID()	142	&SYSCONID	NetView
SYSID()	142	&SYSID	NetView
SYSPLEX()	161	&SYSPLEX	NetView
TASK()	161	&TASK	NetView
TGLOBAL(*name*)	128, 96	&TGLOBAL	NetView
TIME()	164	&TIME	REXX
TOWER(...)	162	None	NetView
TYPE()	162	None	NetView
VTAM()	162	&VTAM	NetView
VTCOMPID()	162	&VTCOMPID	NetView
WEEKDAYN()	163	&WEEKDAYN	NetView
WTOREPLY	142	&WTOREPLY	NetView

Commands Used in Command Lists

The following is a list of NetView commands in the *Tivoli NetView for z/OS Command Reference* that are for use in command lists. With the exception of FLUSHQ, MSGREAD, TRAP, and WAIT, you can use these commands in command lists written in REXX or in the NetView command list language.

Command List Commands

- DOM
- FLUSHQ
- GETMPRES
- GETMSIZE
- GETMTFLG
- GETMTYPE
- GLOBALV
- MSGREAD
- MSGROUTE
- PARSEL2R
- SDOMAIN (with QUIET option)
- TRAP
- WAIT
- WTO
- WTOR

Note: FLUSHQ, MSGREAD, TRAP, and WAIT can be used only in REXX command lists.

When using the commands in a REXX command list, enclose in single quotes the parts of the command on which you do not want variable substitution to take place.

Appendix B. 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 alphabetical 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 57 on page 177
ACTLU	Use this command list to activate a VTAM node.	Figure 58 on page 179
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 59 on page 180
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 60 on page 182
DSPRSTAT	This command list can be used by an operator station task (OST) operator to display the results of several executions of the CHKRSTAT command list for a specific resource. Use DSPRSTAT as an aid when you need to determine how often a resource is active, based on the intervals in which it was checked by the CHKRSTAT command list.	Figure 61 on page 184
GETCG	The GETCG command list gets the value of a common global variable and displays it to the requesting task.	Figure 62 on page 185
GREETING	This command list shows an example of waiting and trapping using the DATE command.	Figure 63 on page 185
LISTVAR	Refer to the NetView online help for a functional description of this command list.	Figure 64 on page 186
PRINT	This command list prints members of a data set to a system print file.	Figure 65 on page 187
TYPE	This command list displays members of a data set one line at a time at the invoking user's terminal.	Figure 66 on page 188
TYPEIT	This command list displays members of a data set one line at a time at the invoking user's terminal.	Figure 67 on page 189

NetView Command List Language Examples

Table 22 on page 176 lists the NetView command list language command list examples shown in this book.

NetView Command List Language Examples Index

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 23 on page 92
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 25 on page 98
GLOBVAR1	The GLOBVAR1 command list illustrates the scope of user variables, task global variables, and common global variables within individual command lists.	Figure 27 on page 100
PATH	This command list uses the &WRITE control statement and a VTAM command.	"&WRITE Control Statement" on page 58
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 26 on page 99

Appendix C. 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 executing 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 57. 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 57. 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 58. ACTLU Example

CHKOPNUM Example

Figure 59 on page 180 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 59. 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 59. CHKOPNUM Example (Part 2 of 2)

CHKRSTAT Example

Figure 60 on page 182 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 60. 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 60. CHKRSTAT Example (Part 2 of 2)

DSPRSTAT Example

Figure 61 on page 184 is an example of a command list that uses the same type of function as Figure 60 on page 182.

```
/*********************************************************************/
/*                                                                   */
/*  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 61. 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 62. 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 63. GREETING Example

LISTVAR Example

```
/*********************************************************************/
/*                                                                   */
/* THE LISTVAR COMMAND LIST WRITTEN IN REXX                          */
/*                                                                   */
/*********************************************************************/
 SELECT
   WHEN MSGVAR(1)='?' THEN               /* HELP REQUESTED ?          */
     'HELP LISTVAR'                      /* GO GET HELP               */
   WHEN MSGVAR(1)¬='' THEN               /* ANY PARMS SPECIFIED ?     */
     'MESSAGE 306E,LISTVAR' MSGVAR(1)    /* NO PARMS ALLOWED          */
   OTHERWISE                             /* ALL OK, LIST OUT VARIABLES*/
     DO
       MYSYS = SUBSTR(OPSYSTEM(),1,3)
       SAY "CNM353I LISTVAR : 'OPSYSTEM' = "OPSYSTEM()
       IF MYSYS = 'VSE' THEN
       SAY "CNM353I LISTVAR : 'CURPART'  = "CURPART()
       IF MYSYS = 'MVS' THEN
       SAY "CNM353I LISTVAR : 'MVSLEVEL' = "MVSLEVEL()
       IF MYSYS = 'MVS' THEN
       SAY "CNM353I LISTVAR : 'CURSYS'   = "CURSYS()
       SAY "CNM353I LISTVAR : 'VTAMLVL'  = "VTAM()
       SAY "CNM353I LISTVAR : 'VTCOMPID' = "VTCOMPID()
       TEMP = NETVIEW()
       IF TEMP = 'NV33' THEN TEMP = 'Tivoli V1R3'
       SAY "CNM353I LISTVAR : 'NETVIEW'  = "TEMP
       SAY "CNM353I LISTVAR : 'NETVIEW'  = "NETVIEW()
       SAY "CNM353I LISTVAR : 'NETID'    = "NETID()
       SAY "CNM353I LISTVAR : 'DOMAIN'   = "DOMAIN()
       SAY "CNM353I LISTVAR : 'APPLID'   = "APPLID()
       SAY "CNM353I LISTVAR : 'OPID'     = "OPID()
       SAY "CNM353I LISTVAR : 'LU'       = "LU()
       SAY "CNM353I LISTVAR : 'TASK'     = "TASK()
       SAY "CNM353I LISTVAR : 'NVCNT'    = "NVCNT()
       SAY "CNM353I LISTVAR : 'HCOPY'    = "HCOPY()
       IF MYSYS = 'MVS' THEN
       SAY "CNM353I LISTVAR : 'CURCONID' = "CURCONID()
       IF MYSYS = 'MVS' &AUTOTASK() = '1' THEN
       SAY "CNM353I LISTVAR : 'AUTCONID' = "AUTCONID()
       IF VTAM() = '' THEN               /* IS VTAM ACTIVE ?          */
         SAY 'CNM386I LISTVAR : VTAM IS NOT ACTIVE AT THIS TIME'
     END                                 /* OF THE LIST VARIABLES     */
   END                                   /* OF SELECT                 */
 RETURN                                  /* RETURN TO CALLER          */
EXIT
```

Figure 64. LISTVAR Example

PRINT Example

Figure 65 on page 187 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 65. PRINT Example

TYPE Example

Figure 66 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 66. TYPE Example

TYPEIT Example

Figure 67 on page 189 is an example of a command list that does essentially the same thing as the example in Figure 66, 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 67. TYPEIT Example

Index

Special Characters

/*%DATA directive 21
/*%LOGIC directive 22
&1 - &31 parameter variables 87
&ACTIONDL control variable 129
&ACTIONMG control variable 129
&APPLID control variable 158
&AREAID control variable 136
&ASID control variable 158
&ATTENDED control variable 159
&ATTNID control variable 129
&AUTCONID control variable 159
&AUTOTASK control variable 159
&AUTOTOKE control variable 136
&BEGWRITE control statement 59
&BITAND built-in function 64
&BITOR built-in function 64
&BITXOR built-in function 65
&CART control variable 136
&CGLOBAL control statement 97
&CONCAT built-in function 66
&CONTROL control statement 56
&CURCONID control variable 159
&CURSYS control variable 159
&DATE control variable 164
&DESC control variable 136
&DISC control variable 159
&DISTAUTO control variable 159
&DOMAIN control variable 160
&EXIT control statement 79
&GOTO control statement 79
&HCOPY control variable 163
&HDRMTYPE control variable 129
&IF control statement 77
&IFRAUGMT control variable 130
&IFRAUI3X control variable 130
&IFRAUIN3 control variable 130
&IFRAUIND control variable 130
&IFRAUSB2 control variable 130
&IFRAUSC2 control variable 130
&IFRAUSDR control variable 130
&IFRAUSRB control variable 131
&IFRAUSRC control variable 131
&IFRAUTA1 control variable 131
&IFRAUWF1 control variable 131
&JOBNAME control variable 136
&JOBNUM control variable 136
&KEY control variable 136
&LENGTH built-in function 69
&LINETYPE control variable 132
&LU control variable 164
&MCSFLAG control variable 137
&MSGASID control variable 137
&MSGAUTH control variable 137
&MSGCATTR control variable 137
&MSGCMISC control variable 137
&MSGCMLVL control variable 138
&MSGCMSGT control variable 138
&MSGCNT control variable 138
&MSGCOJBN control variable 138
&MSGCPROD control variable 139
&MSGCSPLX control variable 139
&MSGCSYID control variable 139
&MSGDOMFL control variable 139
&MSGGBGPA control variable 139
&MSGGDATE control variable 140
&MSGGFGPA control variable 140
&MSGGMFLG control variable 140
&MSGGMID control variable 140
&MSGGSEQ control variable 140
&MSGGSYID control variable 140
&MSGGTIME control variable 141
&MSGID control variable 132
&MSGORIGIN control variable 133
&MSGSRCNM control variable 141
&MSGSTR control variable 134
&MSGTOKEN control variable 141
&MSGTSTMP control variable 134
&MSGTYP control variable 141
&MSUSEG built-in function 70
&MVSLEVEL control variable 160
&MVSRTAIN variable 135
&NCCFCNT control variable 125
&NCCFID built-in function 71, 126
&NCCFSTAT built-in function 72, 126
&NETVIEW control variable 160
&NVDELID control variable 135
&OPID control variable 158
&OPSYSTEM control variable 160
&PARMCNT control variable 125
&PARMSTR control variable 125
&PARTID control variable 160
&PAUSE control statement 61
&PRTY control variable 141
&REPLYID control variable 135
&RETCODE control variable 125
&RXDEFENV control variable 163
&RXDEFSTR control variable 163
&RXNUMENV control variable 163
&RXOVRENV control variable 163
&RXOVRSTR control variable 163
&SESSID control variable 135
&SMSGID control variable 142
&STCKGMT control variable 161
&SUBSTR built-in function 73
&SUPPCHAR control variable 161
&SYSCONID control variable 142
&SYSID control variable 142
&SYSPLEX control variable 161
&TASK control variable 161
&TGLOBAL control statement 96
&THEN clause, &IF control statement 78
&TIME control variable 164
&VTAM control variable 162
&VTCOMPID control variable 162
&WAIT control statement 81
&WEEKDAYN control variable 163
&WRITE control statement 58
&WTOREPLY control variable 142

A

accessibility information ix
ACTAPPLS example
  REXX 177
ACTIONDL
  NetView command list language control variable 129
  REXX function 129
ACTIONMG
  NetView command list language control variable 129
  REXX function 129
activating command lists 4
ACTLU example
  REXX 179
ADDRESS instruction 28
AFTER command, scheduling command lists 9
ALL, &CONTROL operand 57
ALLOCATE, NetView command 30
allocating data sets, NetView 30
APPLID
  NetView command list language control variable 158
  REXX function 158
AREAID
  NetView command list language control variable 136
  REXX function 136
  WTO command 136
arithmetic operations, assignment statements 55
ASID
  NetView command list language control variable 158
  REXX function 158
Assembler command processors, nesting REXX command lists 32
assignment
  clauses, REXX 19
  statements, NetView command list language 54
AT command, scheduling command lists 9
ATTENDED
  NetView command list language control variable 159
  REXX function 159
ATTNID
  NetView command list language control variable 129
  REXX function 129
AUTBYPAS
  REXX function 120
AUTCONID
  NetView command list language control variable 159
  REXX function 159
AUTHCHK
  REXX function 121

AUTHCHKX
  REXX function 123
AUTODROP command, REXX 7
automation, message 105
automation task 16
AUTOTASK
  NetView command list language control variable 159
  OST restrictions 16
  REXX function 159
AUTOTBL command 105
AUTOTOKE
  NetView command list language control variable 136
  REXX function 136
AUTOWRAP setting 14

B

BEGWRITE control statement 59
BGNSESS command 12
blanks stripping 164
books
  feedback viii
  online viii
  ordering viii
built-in function
  NetView command list language
    assignment statement 56
built-in function, &HIER 66
built-in functions
  NetView command list language
    &BITAND 64
    &BITOR 64
    &BITXOR 65
    &CONCAT 66
    &HIER 66
    &LENGTH 69
    &MSUSEG 70
    &NCCFID 71, 126
    &NCCFSTAT 72, 126
    &SUBSTR 73
    definition 63
  REXX 19

C

C command procedures, nesting REXX command lists 32
CALL instruction, using 27
calling another command list 10
CART
  NetView command list language control variable 136
  REXX function 136
  WTO, WTOR commands 136
CGI
  REXX function 159
CGLOBAL
  control statement 97
  REXX function 128
CHKOPNUM, example command list 179
CHKRSTAT, example command list 181
CHRON command, scheduling command lists 9
clauses, REXX 19
CLEAR command 59
CMD
  &CONTROL operand 57
  command 13
CMDNAME
  REXX function 124
CMDSYN statement 5
code points, translating 118
coding conventions
  NetView command list language
    continuation statements 44
    double-byte character text 45
    suppression character conventions 46
    syntax 43
  REXX
    coding non-REXX commands, REXX command list 23
    record size 23
    suppressing display, non-REXX command 25
    syntax 23
command list information
  NetView command list language control variables 120
  REXX functions 120
command list language, NetView 3
command lists
  activating 4
  commands used in 173
  compiling command lists 20
  creating 5
  creating data sets, MVS 5
  definition 3
  display, controlling during execution 16
  loading, main storage 7
  long-running commands, using 12
  message driven 105
  naming 5
  nested 10
  network commands, using 11
  Network Control Program, activating 4
  overview 3
  restarting 9
  restrictions under PPT 15
  routing messages 105
  running 8
  SHOWCODE 14
  startup, examples 4
  stopping 9
  suspending 9
  system commands, using 12
  updating 5
  using 3
commands
  AFTER 9
  ALLOCATE 30
  AT 9
  AUTODROP 7
  AUTOTBL 105
  BGNSESS 12
  CHRON 9
  CMD 13
commands *(continued)*
  common operations services
    LINKDATA 110
    LINKPD 111
    LINKTEST 110
    RUNCMD 112
  DEFAULTS 8
  EVERY 9
  EXECIO 29
  FREE 30
  full-screen 14
  GO 9, 86
  hardware monitor, using command lists 12
  long-running
    major 12
    minor 13
  MSGROUTE 105
  network 11
  operator
    AFTER 9
    AT 9
    CHRON 9
    EVERY 9
    running command lists 9
  OVERRIDE 8
  RESET 9, 86
  RETURN 12
  session monitor, using command lists 12
  STACK 9, 86
  status monitor, using command lists 12
  system 12
  TRACE END (TE) 33
  TRACE START (TS) 32
  UNIQUE 13, 14
  UNSTACK 9, 86
  UPPER 14
  VIEW 14
  VTAM 11
comments
  NetView command list language 54
  REXX 19
common global variables
  NetView command list language 97
common operations services (COS) commands 109
comparing NetView command list language, REXX 169
CONCAT built-in function 66
constants, assignment statements 54
continuation statements 44
CONTINUE, &WAIT operand 90
CONTROL control statement 56
control statements, NetView command list language
  &BEGWRITE 59
  &CGLOBAL 97
  &CONTROL 56
  &EXIT 79
  &GOTO 79
  &IF 77
  &PAUSE 61
  &TGLOBAL 96
  &WAIT 81
  &WRITE 58

control statements, NetView command list language *(continued)*
  comparison, REXX instructions 169
  definition 56
control variables, NetView command list language
  &1 – &31 87
  &ACTIONDL 129
  &ACTIONMG 129
  &APPLID 158
  &AREAID 136
  &ASID 158
  &ATTENDED 159
  &ATTNID 129
  &AUTCONID 159
  &AUTOTASK 159
  &AUTOTOKE 136
  &CART 136
  &CURCONID 159
  &CURSYS 159
  &DATE 164
  &DESC 136
  &DISC 159
  &DISTAUTO 159
  &DOMAIN 160
  &HCOPY 163
  &HDRMTYPE 129
  &IFRAUGMT 130
  &IFRAUI3X 130
  &IFRAUIN3 130
  &IFRAUIND 130
  &IFRAUSB2 130
  &IFRAUSC2 130
  &IFRAUSDR 130
  &IFRAUSRB 131
  &IFRAUSRC 131
  &IFRAUTA1 131
  &IFRAUWF1 131
  &JOBNAME 136
  &JOBNUM 136
  &KEY 136
  &LINETYPE 132
  &LU 164
  &MCSFLAG 137
  &MSGASID 137
  &MSGAUTH 137
  &MSGCATTR 137
  &MSGCMISC 137
  &MSGCMLVL 138
  &MSGCMSGT 138
  &MSGCNT 138
  &MSGCOJBN 138
  &MSGCPROD 139
  &MSGCSPLX 139
  &MSGCSYID 139
  &MSGDOMFL 139
  &MSGGBGPA 139
  &MSGGDATE 140
  &MSGGFGPA 140
  &MSGGMFLG 140
  &MSGGMID 140
  &MSGGSEQ 140
  &MSGGSYID 140
  &MSGGTIME 141
  &MSGID 132
  &MSGORIGIN 133
  &MSGSRCNM 141
control variables, NetView command list language *(continued)*
  &MSGSTR 134
  &MSGTOKEN 141
  &MSGTSTMP 134
  &MSGTYP 141
  &MVSLEVEL 160
  &MVSRTAIN 135
  &NCCFCNT 125
  &NETID 160
  &NETVIEW 160
  &NVDELID 135
  &OPID 158
  &OPSYSTEM 160
  &PARMCNT 125
  &PARMSTR 125
  &PARTID 160
  &PRTY 141
  &REPLYID 135
  &RETCODE 125
  &RXDEFENV 163
  &RXDEFSTR 163
  &RXNUMENV 163
  &RXOVRENV 163
  &RXOVRSTR 163
  &SESSID 135
  &SMSGID 142
  &STCKGMT 161
  &SUPPCHAR 161
  &SYSCONID 142
  &SYSID 142
  &SYSPLEX 161
  &TASK 161
  &TIME 164
  &VTAM 162
  &VTCOMPID 162
  &WAIT, using 87
  &WEEKDAYN 163
  &WTOREPLY 142
  comparing REXX functions 170
  definition 52
CONTWAIT, &WAIT operand 89
COS return codes 110
creating command lists 5
CURCONID
  NetView command list language control variable 159
  REXX function 159
CURSYS
  NetView command list language control variable 159
  REXX function 159
Customer Support ix

D

DATA (%DATA) directive 21
Data REXX 20
  /*%DATA directive 21
  /*%LOGIC directive 22
  host command environment 29
data set, information functions 126
data set, MVS 5
data set information functions 126
DATE control variable 164
DBCS 45
deallocating data sets, NetView 30
DEFAULTS command 8
DESC
  NetView command list language control variable 136
  REXX function 136
  WTO, WTOR commands 136
disability information ix
DISC
  NetView command list language control variable 159
  REXX function 159
DISPLAY
  &WAIT operand 89
display, controlling 16
displaying panels 14
DISTAUTO
  NetView command list language control variable 159
  REXX function 159
DOMAIN
  NetView command list language control variable 160
  REXX function 160
domain information, REXX functions 125
double-byte character set characters (DBCS)
  &CONCAT, using 66
  &SUBSTR, using 74
  coding conventions, NetView command list language 45
  PPT, command list 16
DOUBLESUPP character 46
DSIPUSH macro 13
DSPRSTAT, example command list 183

E

e-mail contact ix
editing facilities, updating command lists 5
ENDWAIT, &WAIT operand 85, 89
ENVDATA
  REXX function 160
environment addressed by REXX, changing 28
ERR, &CONTROL operand 57
ERROR, &WAIT operand 85
errors, handling
  NetView command list language 86
  REXX 34
EVENT
  REXX function 129
event=-label pairs, &WAIT control statement 82, 86
EVERY command, scheduling command lists 9
examples
  ACTAPPLS
    REXX 177
  ACTLU
    REXX 179
  command list using
    &WAIT 92
  command lists
    reference, define, update task global variables 98

examples *(continued)*
GETCG
REXX 184
GLOBVAR1 100
GREETING
REXX 185
LISTVAR
REXX 186
MSUSEG 157
REXX command lists, NetView
CHKOPNUM 179
CHKRSTAT 181
DSPRSTAT 183
PRINT 186
TYPE 188
TYPEIT 188
startup command lists 4
EXECIO command, REXX command list 29
EXIT control statement 79
expressions
NetView command list language 54
REXX 19

F

feedback about publications ix
FNDMBR, REXX function 126
FREE, NetView command 30
full-screen commands, using 14
function packages, REXX, writing 28
functions
built-in
NetView command list language 63
REXX 19
REXX
ACTIONDL() 129
ACTIONMG() 129
APPLID() 158
AREAID() 136
ASID() 158
ATTENDED() 159
ATTNID() 129
AUTBYPAS 120
AUTCONID() 159
AUTHCHK() 121
AUTHCHKX() 123
AUTOTASK() 159
AUTOTOKE() 136
CART() 136
CGI() 159
CGLOBAL() 128
CMDNAME() 124
CODE2TXT() 118
comparison, NetView command list language control variables 170
CURCONID() 159
CURSYS() 159
DESC() 136
DISC() 159
DISTAUTO() 159
DOMAIN() 160
ENVDATA() 160
EVENT() 129
FNDMBR() 126

functions *(continued)*
REXX *(continued)*
HCOPY() 163
HDRMTYPE() 129
HIER(n) 144
HMEPNAU 146
HMEPNET 146
HMEPNETV 147
HMFWDSNA 148
IFRAUGMT() 130
IFRAUI3X() 130
IFRAUIN3() 130
IFRAUIND() 130
IFRAUSB2() 130
IFRAUSC2() 130
IFRAUSDR() 130
IFRAUSRB() 131
IFRAUSRC() 131
IFRAUTA1() 131
IFRAUWF1() 131
JOBNAME() 136
JOBNUM() 136
KEY() 136
LINESIZE() 28
LINETYPE() 132
LU() 164
MCSFLAG() 137
MSGASID() 137
MSGAUTH() 137
MSGCATTR() 137
MSGCMISC() 137
MSGCMLVL() 138
MSGCMSGT() 138
MSGCNT() 138
MSGCOJBN() 138
MSGCPROD() 139
MSGCSPLX() 139
MSGCSYID() 139
MSGDOMFL() 139
MSGGBGPA() 139
MSGGDATE() 140
MSGGFGPA() 140
MSGGMFLG() 140
MSGGMID() 140
MSGGSEQ() 140
MSGGSYID() 140
MSGGTIME() 141
MSGID() 132
MSGORIGN() 133
MSGREAD 39
MSGSRCNM() 141
MSGSTR() 134
MSGTOKEN() 141
MSGTSTMP() 134
MSGTYP() 141
MSGVAR(n) 135
MSUSEG() 152
MVSLEVEL 160
MVSRTAIN() 135
NETID() 160
NETVIEW() 160
NVCNT() 125
NVDELID() 135
NVID(n) 126
NVSTAT() 126
OPID() 158
OPSYSTEM() 160

functions *(continued)*
REXX *(continued)*
PANEL() 160
PARMCNT() 125
PARTID() 160
PRTY() 141
REPLYID() 135
restrictions 27
ROUTCDE() 142
RXDEFENV() 163
RXDEFSTR() 163
RXNUMENV() 163
RXOVRENV() 163
RXOVRSTR() 163
SESSID() 135
SMSGID() 142
STCKGMT() 161
STORAGE() 28
SUBSYM() 120
SUPPCHAR() 161
SYSCONID() 142
SYSID() 142
SYSPLEX() 161
TASK() 161
TGLOBAL() 128
TOWER() 162
TYPE() 162
VTAM() 162
VTCOMPID() 162
WEEKDAYN() 163
WTOREPLY 142
functions, data set information 126

G

GETCG example
REXX 184
global variable information functions 128
global variables
common
NetView command list language 97
task
NetView command list language 96
GLOBALV
instruction
NetView command list language 102
GLOBVAR1 example 100
GO command 9, 86
GOTO control statement 79
GREETING example
REXX 185

H

hardware monitor commands, using 12
HCOPY
NetView command list language control variable 163
REXX function 163
HDRMTYPE
NetView command list language control variable 129

HDRMTYPE *(continued)*
REXX function 129
hexadecimal notation (NetView command list language) 53
HIER function
NetView command list language 66
REXX 144
HMEPNAU function 146
HMEPNET function 146
HMEPNETV function 147
HMFWDSNA function 148

I

IF control statement 77
IFRAUGMT
NetView command list language control variable 130
REXX function 130
IFRAUI3X
NetView command list language control variable 130
REXX function 130
IFRAUIN3
NetView command list language control variable 130
REXX function 130
IFRAUIND
NetView command list language control variable 130
REXX function 130
IFRAUSB2
NetView command list language control variable 130
REXX function 130
IFRAUSC2
NetView command list language control variable 130
REXX function 130
IFRAUSDR
NetView command list language control variable 130
REXX function 130
IFRAUSRB
NetView command list language control variable 131
REXX function 131
IFRAUSRC
NetView command list language control variable 131
REXX function 131
IFRAUTA1
NetView command list language control variable 131
REXX function 131
IFRAUWF1
NetView command list language control variable 131
REXX function 131
information, accessibility ix
information, disability ix
information function, MSU 143
initialization, running command lists 8
instructions, REXX
ADDRESS 28
CALL 27
instructions, REXX *(continued)*
comparison, NetView command list language control statements 169
definition 19
PARSE 26
restrictions 26
SAY 23, 26
SIGNAL 34
TRACE END 32
TRACE START 32

J

JOBNAME
NetView command list language control variable 136
REXX function 136
JOBNUM
NetView command list language control variable 136
REXX function 136

K

KEY
NetView command list language control variable 136
REXX function 136
keyboard, shortcut keys ix

L

labels
NetView command list language 46
REXX 19
LENGTH built-in function 69
LINESIZE function 28
LINETYPE
NetView command list language control variable 132
REXX function 132
LINKDATA command 110
LINKPD command 111
LINKTEST command 110
LISTVAR example
REXX 186
loading command lists, storage 7
LOGIC (%LOGIC) directive 22
logon, operator, automatically running command lists 8
long-running commands
major 12
minor 13
queuing 13
LU
NetView command list language control variable 164
REXX function 164

M

major long-running commands 12
management services unit (MSU) functions
&HIER 66
management services unit (MSU) functions *(continued)*
&MSUSEG 70
HIER() 144
MSUSEG() 152
NetView command list language 63, 66, 70
REXX 66, 69, 70
management services units information function 143
manuals
feedback viii
online viii
ordering viii
MCSFLAG
NetView command list language control variable 137
REXX function 137
message
=-label pairs, coding 82, 86
automating responses 105
multiline, working 84
processing information
REXX functions 128
routing, command list 105
sending, operators 57
waiting, command lists
NetView command list language 89
message automation
command lists
defining 105
error handling 107
running 9
testing 106
implementing 105
midnight 164
minor long-running commands 13
MSGASID
NetView command list language control variable 137
REXX function 137
MSGAUTH
NetView command list language control variable 137
REXX function 137
MSGCATTR
NetView command list language control variable 137
REXX function 137
MSGCMISC
NetView command list language control variable 137
REXX function 137
MSGCMLVL
NetView command list language control variable 138
REXX function 138
MSGCMSGT
NetView command list language control variable 138
REXX function 138
MSGCNT
NetView command list language control statement 132, 138
REXX function 138

MSGCOJBN
  NetView command list language control variable 138
  REXX function 138
MSGCPROD
  NetView command list language control variable 139
  REXX function 139
MSGCSPLX
  NetView command list language control variable 139
  REXX function 139
MSGCSYID
  NetView command list language control variable 139
  REXX function 139
MSGDOMFL
  NetView command list language control variable 139
  REXX function 139
MSGGBGPA
  NetView command list language control variable 139
  REXX function 139
MSGGDATE
  NetView command list language control variable 140
  REXX function 140
MSGGFGPA
  NetView command list language control variable 140
  REXX function 140
MSGGMFLG
  NetView command list language control variable 140
  REXX function 140
MSGGMID
  NetView command list language control variable 140
  REXX function 140
MSGGSEQ
  NetView command list language control variable 140
  REXX function 140
MSGGSYID
  NetView command list language control variable 140
  REXX function 140
MSGGTIME
  NetView command list language control variable 141
  REXX function 141
MSGID
  REXX function 132
MSGORIGIN, NetView command list language control variable 133
MSGORIGN, REXX function 133
MSGREAD
  setting functions 39
MSGROUTE command 105
MSGSRCNM
  NetView command list language control variable 141
  REXX function 141
MSGSTR
  NetView command list language control variable 134
MSGSTR *(continued)*
  REXX function 134
MSGTOKEN
  NetView command list language control variable 141
  REXX function 141
MSGTSTMP
  NetView command list language control variable 134
  REXX function 134
MSGTYP
  NetView command list language control variable 141
  REXX function 141
MSGVAR, REXX function 135
MSU (management services unit) functions
  NetView command list language (built-in functions) 63
  REXX 69
MSU information function 143
MSUSEG function 70, 152
  REXX usage examples 156
multiline messages, using 84
MVS
  command 12
  creating data sets 5
  message processing information 135
MVSLEVEL
  NetView command list language control variable 160
  REXX function 160
MVSRTAIN
  NetView command list language control variable 135
  REXX function 135

N

NCCFCNT NetView command list language control variable 125
NCCFID
  built-in function 71, 126
NCCFSTAT built-in function 72, 126
nested command lists
  definition 10
  NetView command list language, using &WAIT 88
  REXX
    Assembler, C, PL/I command procedures 32
    using MSGREAD 39
    using TRAP 38
    using WAIT 38
  testing 11
NETVIEW
  NetView command list language control variable 160
  REXX function 160
NetView command list language
  coding conventions 43
  comments 54
  comparison, REXX 169
  features 43
  functions, built-in 56
  labels 46
  null statements 54
NetView command list language *(continued)*
  variables 47
NetView commands
  using &PAUSE 62
  using &WAIT 86
network commands, using 11
network control program, activating, command lists 4
NOINPUT, &PAUSE operand 61
NOSUB, &BEGWRITE operand 60
null statements 54
nulls stripping 164
NVCNT function 125
NVDELID
  NetView command list language control variable 135
  REXX function 135
NVID function 126
NVSTAT function 126

O

online publications viii
operands
  ALL, &CONTROL control statement 57
  CMD, &CONTROL control statement 57
  CONTINUE, &WAIT control statement 90
  CONTWAIT, &WAIT control statement 89
  DISPLAY
    &WAIT control statement 89
  ENDWAIT, &WAIT control statement 85, 89
  ERR, &CONTROL control statement 57
  ERROR, &WAIT control statement 85
  NOINPUT, &PAUSE control statement 61
  NOSUB, &BEGWRITE control statement 60
  STRING, &PAUSE control statement 61
  SUB, &BEGWRITE control statement 60
  SUPPCHAR 161
  SUPPCHAR, in CNMSTYLEt 46
  SUPPRESS
    &WAIT control statement 89
  VARS, &PAUSE control statement 62
operator
  command, running command lists 9
  information
    NetView command list language control variables 158
    REXX functions 158
  input, REXX command list 26
  logon, command list 8
  sending messages 57
OPID
  NetView command list language control variable 158
  REXX function 158

OPSYSTEM
  NetView command list language control variable 160
  REXX function 160
ordering publications viii
OST, autotask, restrictions 16
OVERRIDE command 8

P

PANEL
  REXX function 160
panels, displaying 14
parameter variables, NetView command list language
  &WAIT, using 87
  characteristics 48
  nested command lists, using 50
  null 51
  passing, command list 49
  quoted strings, using 51
  special characters, using 51
PARMCNT
  NetView command list language control variable 125
  REXX function 125
PARMSTR control variable 125
PARSE instruction 26
parsing
  REXX command lists 32
PARTID
  NetView command list language control variable 160
  REXX function 160
PAUSE control statement, using NetView commands 61, 62
pausing, REXX command list 26
PL/I command procedures, nesting REXX commands 32
PPT restrictions 15
PRINT, example command list 186
PROFILE statement 8
PRTY
  NetView command list language control variable 141
  REXX function 141
publications
  feedback viii
  online viii
  ordering viii

Q

queuing long-running commands 13
quotes, REXX command lists or Data REXX files 23

R

record size
  NetView command list language 44
  REXX 23
REPLYID
  NetView command list language control variable 135
  REXX function 135
RESET command 9, 86
Restructured Extended Executor (REXX) language
  command lists
    coding non-REXX commands 25
    compiling 19
    environment, changing 28
    environment functions 163
    errors, recovering 34
    examples 177
    EXECIO command, using 29
    nesting assembler, C, PL/I command procedures 32
    operator input, pausing 26
    parsing 32
    restrictions 26
    SAY instruction, using 23
    suppressing non-REXX commands 25
    tracing 32
    TSO/E environments 31
  command lists and data REXX files
    SAY instruction, using 26
  command lists and Data REXX files
    coding conventions 23
    return codes 33
  command lists or Data REXX files
    CALL instruction, using 27
    LINESIZE function, using 28
    record size 23
    restrictions 27
    STORAGE function, using 28
    trailing blanks 23
  comparison, NetView command list language 169
  function packages, writing 28
  introduction 19
RETCODE control variable 125
return codes
  COS 110
  NetView command list language 125
  REXX 33
RETURN command, REXX restrictions 12
ROUTCDE
  NetView command list language control variable 142
  REXX function 142
RUNCMD command 112
running command lists
  another command list 10
  NetView is started 8
  NetView receives messages 9
  operator command 9
  operator logon 8
  specified time 9
  user-written command procedure 11
RXDEFENV
  NetView command list language control variable 163
  REXX function 163
RXDEFSTR
  NetView command list language control variable 163
  REXX function 163
RXNUMENV
  NetView command list language control variable 163
  REXX function 163
RXOVRENV
  NetView command list language control variable 163
  REXX function 163
RXOVRSTR
  NetView command list language control variable 163
  REXX function 163

S

SAY instruction, using 23, 26
scope checking
  variables, NetView command list language 99
SECURITY
  controlling access, command lists 6
  running command lists when NetView is started 8
  using network commands in command lists 11
SESSID
  NetView command list language control variable 135
  REXX function 135
session
  information
    NetView command list language control variables 158
    REXX functions 158
  monitor commands, command list 12
  TAF example 4
shortcut keys, keyboard ix
SHOWCODE command list 14
SIGNAL instruction 34
SMSGID
  NetView command list language control variable 142
  REXX function 142
STACK command 9, 86
status monitor commands, command list 12
STCKGMT
  NetView command list language control variable 161
  REXX function 161
STIMER, MVS and VM GCS macro 16
STORAGE function 28
STRING, &PAUSE operand 61
stripping, nulls and blanks 164
SUB, &BEGWRITE operand 60
SUBSTR built-in function 73
SUBSYM, REXX function 120
SUPPCHAR 46, 161
SUPPCHAR control variable 161
SUPPRESS
  &WAIT operand 89
suppressing
  messages 106
  non-REXX commands, REXX command lists 25
suppression characters 46

SYSCONID
  NetView command list language control variable 142
  REXX function 142
SYSID
  NetView command list language control variable 142
  REXX function 142
SYSPLEX
  NetView command list language control variable 161
  REXX function 161
system commands, using 12

T

TAF
  command output 13
  session example 4
TASK
  NetView command list language control variable 161
  REXX function 161
task global variables
  command list examples, reference, define, update 98
  NetView command list language 96
TE command 33
terminal information
  NetView command list language control variables 163
  REXX functions 163
TGLOBAL
  control statement 96
  REXX function 128
THEN clause, &IF control statement 78
TIME control variable 164
time intervals, running command lists 9
Tivoli Customer Support ix
tokens, message
  NetView command list language 83
TOWER
  REXX function 162
TRACE END command 33
TRACE START command 32
tracing, REXX command lists 32
translating code points 118
translation functions
  code-to-text function (CODE2TXT) 118
translation tables, code-to-text 118
TRAP
  REXX command list 38
TS command 32
TSO/E environment 31
TSO/E EXECIO command 29
TYPE
  REXX function 162
TYPE, command list example 188
TYPEIT, command list example 188

U

UNIQUE command 13, 14
UNSTACK command 9, 86
UPPER command 14
user variables 52
user-written command procedure, activating command lists 11

V

variables
  command list information
    NetView command list language 120
    REXX 120
  operator information
    NetView command list language 158
    REXX 158
  session information
    NetView command list language 158
    REXX 158
  substitution order 47
  terminal information
    REXX 163
  user 52
VARS, &PAUSE operand 62
VIEW command 14
VIEWAID control variable 14
VIEWCURCOL control variable 14
VIEWCURROW control variable 14
VIEWICCOL control variable 14
VIEWICROW control variable 14
VTAM
  commands 11
  NetView command list language control variable 162
  REXX function 162
VTCOMPID
  NetView command list language control variable 162
  REXX function 162

W

WAIT
  NetView command list language control statement
    coding suggestions 91
    control and parameter variables 87
    customizing 89
    ending 86, 91
    general 81
    nested command lists, using 88
    NetView commands, using 86
    sample using 92
  REXX instruction
    nested REXX command lists, using 38
WEEKDAYN
  NetView command list language control variable 163
  REXX function 163
WRITE control statement 58
WTOREPLY 142

File Number: S370/4300/30XX-50
Program Number: 5697–ENV

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-8862-00