IBM Tivoli NetView for z/OS
Version 6 Release 1

Automation Guide

SC27-2846-01

IBM Tivoli NetView for z/OS
Version 6 Release 1

Automation Guide

SC27-2846-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 615.

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

This edition replaces SC27-2846-00.

Contents

Figures

About this publication

The IBM® Tivoli® NetView® for z/OS® product provides advanced capabilities that you can use to maintain the highest degree of availability of your complex, multi-platform, multi-vendor networks and systems from a single point of control. This publication, the *IBM Tivoli NetView for z/OS Automation Guide*, provides information about planning for automated operations. You can use the automation capabilities of the NetView program to improve system and network efficiency, and operator productivity. NetView automation can eliminate or simplify much of the routine work that operators perform.

Intended audience

This publication is for system programmers who want to use the NetView program for system automation, network automation, or both. The publication is both for those who are new to automation and for those who have existing automation projects that they want to update or expand.

Publications

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

IBM Tivoli NetView for z/OS library

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

- *Administration Reference*, SC27-2869, describes the NetView program definition statements required for system administration.
- *Application Programmer's Guide*, SC27-2870, describes the NetView program-to-program interface (PPI) and how to use the NetView application programming interfaces (APIs).
- *Automation Guide*, SC27-2846, describes how to use automated operations to improve system and network efficiency and operator productivity.
- *Command Reference Volume 1 (A-N)*, SC27-2847, and *Command Reference Volume 2 (O-Z)*, SC27-2848, describe the NetView commands, which can be used for network and system operation and in command lists and command procedures.
- *Customization Guide*, SC27-2849, describes how to customize the NetView product and points to sources of related information.
- *Data Model Reference*, SC27-2850, provides information about the Graphic Monitor Facility host subsystem (GMFHS), SNA topology manager, and MultiSystem Manager data models.
- *Installation: Configuring Additional Components*, GC27-2851, describes how to configure NetView functions beyond the base functions.
- *Installation: Configuring Graphical Components*, GC27-2852, describes how to install and configure the NetView graphics components.
- *Installation: Configuring the GDPS Active/Active Continuous Availability Solution*, SC14-7477, describes how to configure the NetView functions that are used with the GDPS Active/Active Continuous Availability solution.

- *Installation: Configuring the NetView Enterprise Management Agent*, GC27-2853, describes how to install and configure the NetView for z/OS Enterprise Management Agent.
- *Installation: Getting Started*, GI11-9443, describes how to install and configure the base NetView functions.
- *Installation: Migration Guide*, GC27-2854, describes the new functions that are provided by the current release of the NetView product and the migration of the base functions from a previous release.
- *IP Management*, SC27-2855, describes how to use the NetView product to manage IP networks.
- *Messages and Codes Volume 1 (AAU-DSI)*, GC27-2856, and *Messages and Codes Volume 2 (DUI-IHS)*, GC27-2857, describe the messages for the NetView product, the NetView abend codes, the sense codes that are included in NetView messages, and generic alert code points.
- *Programming: Assembler*, SC27-2858, describes how to write exit routines, command processors, and subtasks for the NetView product using assembler language.
- *Programming: Pipes*, SC27-2859, describes how to use the NetView pipelines to customize a NetView installation.
- *Programming: PL/I and C*, SC27-2860, describes how to write command processors and installation exit routines for the NetView product using PL/I or C.
- *Programming: REXX and the NetView Command List Language*, SC27-2861, describes how to write command lists for the NetView product using the Restructured Extended Executor language (REXX) or the NetView command list language.
- *Resource Object Data Manager and GMFHS Programmer's Guide*, SC27-2862, describes the NetView Resource Object Data Manager (RODM), including how to define your non-SNA network to RODM and use RODM for network automation and for application programming.
- *Security Reference*, SC27-2863, describes how to implement authorization checking for the NetView environment.
- *SNA Topology Manager Implementation Guide*, SC27-2864, describes planning for and implementing the NetView SNA topology manager, which can be used to manage subarea, Advanced Peer-to-Peer Networking, and TN3270 resources.
- *Troubleshooting Guide*, GC27-2865, provides information about documenting, diagnosing, and solving problems that occur in the NetView product.
- *Tuning Guide*, SC27-2874, provides tuning information to help achieve certain performance goals for the NetView product and the network environment.
- *User's Guide: Automated Operations Network*, SC27-2866, describes how to use the NetView Automated Operations Network (AON) component, which provides event-driven network automation, to improve system and network efficiency. It also describes how to tailor and extend the automated operations capabilities of the AON component.
- *User's Guide: NetView*, SC27-2867, describes how to use the NetView product to manage complex, multivendor networks and systems from a single point.
- *User's Guide: NetView Enterprise Management Agent*, SC27-2876, describes how to use the NetView Enterprise Management Agent.
- *User's Guide: NetView Management Console*, SC27-2868, provides information about the NetView management console interface of the NetView product.
- *Licensed Program Specifications*, GC31-8848, provides the license information for the NetView product.

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

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

Related publications

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

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

Accessing terminology online

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

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

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

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

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

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

MVS™ For z/OS operating systems

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

VTAM®
For Communications Server - SNA Services

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

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

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

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

Using NetView for z/OS online help

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

Using LookAt to look up message explanations

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

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

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

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

Accessing publications online

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

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

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

Ordering publications

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

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

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

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

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

Accessibility

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

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

Tivoli technical training

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

Tivoli user groups

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

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

Downloads

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

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

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

These applications can help with the following tasks:

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

Support information

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

Online

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

IBM Support Assistant

The IBM Support Assistant is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The Support Assistant provides quick access to support-related information and serviceability tools for problem determination. To install the Support Assistant software, go to http://www.ibm.com/software/support/isa/.

Troubleshooting information

For more information about resolving problems with the NetView for z/OS product, see the *IBM Tivoli NetView for z/OS Troubleshooting Guide*. Additional support for the NetView for z/OS product is available through

the NetView user group on Yahoo at http://groups.yahoo.com/group/NetView/. This support is for NetView for z/OS customers only, and registration is required. This forum is monitored by NetView developers who answer questions and provide guidance. When a problem with the code is found, you are asked to open an official problem management record (PMR) to obtain resolution.

Conventions used in this publication

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

Typeface conventions

This publication uses the following typeface conventions:

Bold

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

Italic

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

`Monospace`

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

Operating system-dependent variables and paths

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

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

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

Syntax diagrams

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

Symbols

The following symbols are used in syntax diagrams:

►► Marks the beginning of the command syntax.

► Indicates that the command syntax is continued.

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

►◄ Marks the end of the command syntax.

Parameters

The following types of parameters are used in syntax diagrams:

Required Required parameters are shown on the main path.

Optional Optional parameters are shown below the main path.

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

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

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

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

Punctuation and parentheses

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

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

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

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

```
COMMAND_NAME opt_variable_1,,opt_variable_3
```

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

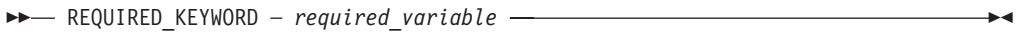
Abbreviations

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

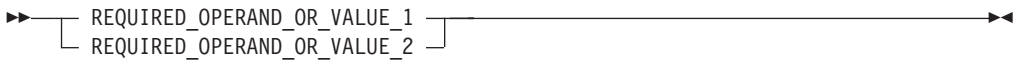
Syntax examples

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

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

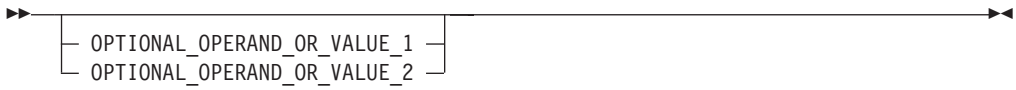


A required choice (two or more items) is shown in a vertical stack on the main path. The items are shown in alphanumeric order.



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

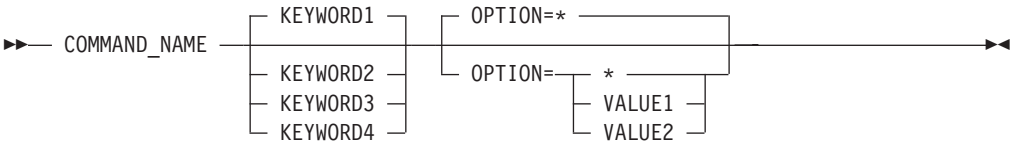
A required choice (two or more items) is shown in a vertical stack below the main path. The items are shown in alphanumeric order.



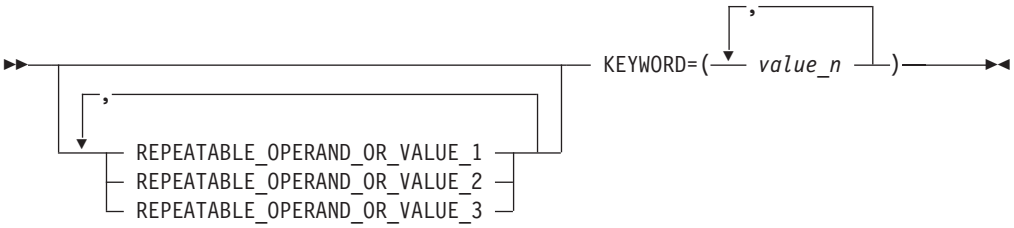
Default keywords and values: Default keywords and values are shown above the main syntax line in one of the following ways:

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

main syntax line is used. The following syntax example shows the default values for operand OPTION=* above and below the main syntax line.



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



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

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

Fragment1

Fragment2

Part 1. Introducing Automation

Chapter 1. Introducing NetView Automation

This chapter introduces NetView automation by describing:
- What the term *NetView automation* means
- Benefits of automation
- Classes of automation
- Stages of automation

What Does NetView Automation Mean?

NetView automation means using the NetView program (and some associated products) to automate many of the information-system and network operations that usually require human intervention. The NetView program provides specialized services to assist in system and network automation. Through these services, the NetView program can perform many routine operator tasks.

For an overview of specific functions and facilities of the NetView program and other products that contribute to NetView automation, see Chapter 2, "Overview of Automation Products," on page 21.

Benefits of Automation

NetView automation offers system-wide and network-wide benefits by simplifying your operating environment. You can reduce the amount of manual intervention required to manage operating systems, subsystems, application programs, network devices, and many other products.

The need to simplify operations increases as you add hardware and software products to your data center, data centers to your network, and personnel to your data-processing staff. By simplifying your operations, NetView automation can help you meet required service levels, contain costs, and make efficient use of your operation staff.

NetView automation helps you:
- Improve system and network availability
- Remove constraints to growth
- Increase operator productivity
- Ensure more consistent operating procedures

Improving System and Network Availability

Automation can improve the availability of your system and network. Automated operations can quickly and accurately respond to unexpected events. When outages do occur, whether planned or unplanned, automation can reduce your recovery time.

Automation decreases the chances for operator errors. Some operator errors can cause failures and lengthen recovery times. For example, an operator might fail to see a message or might type a command incorrectly. Also, an operator might have to type long sequences of commands, remembering the command syntaxes of several programs or components (or take the time to look them up). There are many opportunities for operator error.

With automation, you substitute automatic responses for operator-typed commands. If operator intervention is required, automation procedures can simplify the tasks, reduce the chances of mistakes, and ensure similar responses to similar events. Automation also expedites shutdown, initialization, and recovery procedures, reducing downtime.

Removing Constraints to Growth

Automation can help you remove constraints on system and network growth. For example, ever-increasing data rates might constrain your growth.

As you add faster systems and larger networks to the environment, your operators can receive more messages and alerts. Under normal operating conditions, most operators can read and comprehend each message but might have difficulty reacting to all of them. Automation can reduce the number of messages and alerts that are displayed by:

- Suppressing routine messages
- Blocking routine alerts
- Responding to messages and alerts automatically

Increasing the number of consoles also constrains growth. New products can add consoles that you need to manage. Regardless of your operating system, having many systems requires many consoles. Automation consolidates consoles on individual systems and helps you operate many systems from one centralized point.

Another constraint is the increasing complexity of networks. Interconnected networks often include large numbers of resources, many product types, and combinations of TCP/IP and Systems Network Architecture (SNA) resources. Finding experienced operators to manage all of them can be difficult and costly.

Automation reduces the complexity of the operator's task by managing complex networks according to rules that you specify. Therefore, automation can help you to manage system and network growth.

Increasing Operator Productivity

With automation, operator productivity can increase because the operators spend less time reading messages and alerts and performing repetitive tasks. The operators have more time to concentrate on the tasks that require operator intervention, such as resolution of a new problem.

Ensuring Consistent Operating Procedures

By writing automation procedures and documenting them, you can structure your operations and enable effective reviews. Automation provides a basis for ensuring consistent operating procedures across your organization. Using automation, you can implement new operating procedures quickly and consistently, and you can manage changes more easily and efficiently.

Classes of Automation

With NetView, you can establish a wide variety of automated environments. This section describes several classes of automated operations and the terminology used in this book to describe each one. These classes include:

- "System Automation" on page 5

System and Network Automation

Besides automating its own internal processing, the NetView program can accomplish both system and network automation. *System automation* is the automated operation of the operating system, subsystems, and application programs. *Network automation* is the automated operation of network resources through a communication program, such as VTAM.

You can use the NetView program to implement system automation, network automation, or both. If you combine system and network automation in a single design, you can develop integrated, comprehensive automation. You can also give operators a unified view of information to help them perform problem determination on all system hardware, system software, and network devices that might contribute to a problem.

The content of messages, management services units (MSUs), and system commands are examined by the automation table. Based on that content, the automation table issues appropriate commands to control your system and network. *MSUs* are data structures that carry alert major vectors and other management-services data.

For more information about MSUs in NetView, see Chapter 9, "NetView Program Information Routing for Automation," on page 81. For more information about MSUs in SNA, refer to the *SNA Management Services Reference* and *Systems Network Architecture Formats*.

System Automation

System automation means automatically responding to system messages and MSUs, and automatically issuing system commands. The system commands can be issued either at scheduled times or in response to a system message or MSU.

For information about how the NetView program accomplishes system automation with the help of operating-system facilities, see "Operating-System Automation Facilities and Interactions with the NetView Program" on page 27.

From the perspective of the NetView program, system messages and MSUs are the messages and MSUs that an operating system, subsystem, or application program issues. The operating system message types include:

- Write-to-operator (WTO)
- Write-to-operator-with-reply (WTOR)

The NetView program can alter or redirect these before they are presented to consoles or system logging.

System MSUs are MSUs that come across the NetView program-to-program interface or through LU 6.2 sessions from other programs on the system. NetView automation can suppress or automatically respond to system messages and MSUs.

System commands are the commands that operators can issue to systems, subsystems, and application programs. In an automated environment, NetView

operators and automation routines can use all system commands. For example, you might issue a system command automatically at specified intervals or in response to a particular system message.

Network Automation

Network automation means automatically responding to network messages and MSUs, and automatically issuing network commands. The network commands can be issued either at scheduled times or in response to a network message or MSU.

The NetView program provides you with extensive, policy-based automation for your network resources. AON provides automation of the following network resources:

- VTAM SNA
- TCP/IP

The NetView program accomplishes network automation through interaction with other communication software, typically VTAM. If you are already using the NetView program for network management, you can progress to network automation by having the program do much of the work that operators now do. Network automation, unlike system automation, does not use the operating system's message-processing facilities.

Network messages and *network MSUs* are those messages and MSUs that come from or go through the VTAM program, directly or indirectly. They include:

- VTAM messages sent to the NetView program across the program operator interface
- MSUs sent to the NetView program to report hardware and software problems in the network

The NetView program can suppress or automatically respond to network messages and MSUs.

Network commands are any commands that operators can issue to VTAM or through it to network devices. NetView automation facilities use many of these commands.

Single-System or Multiple-System Automation

You can also choose between single-system and multiple-system automation. When beginning new automation, start with single-system automation. That is, automate as many operations locally (at each system) as possible before moving to multiple-system automation. You thereby reduce the number of interactions needed with other systems to achieve full multiple-system automation. You also avoid overtaxing the communication facilities, focal-point systems, and telecommunication lines.

For descriptions of the stages of single-system and multiple-system automation, see “Stages of Automation” on page 7.

Single-System Automation

In *single-system automation*, the automation of each host system is self-contained. You can automate the system, its subsystems and application programs, and the network devices in the domain of that system's VTAM program. However, in single-system automation, the NetView program cannot automate any devices outside its own VTAM domain. Operators handle those tasks that cannot be automated locally, such as recovery of an operating system or an initial program load.

Multiple-System Automation

In *multiple-system automation*, you coordinate automation across two or more host systems. The coordination enables you to automate the operation of resources that you cannot automate locally on a single system. Multiple-system automation is either *single-site* or *multi-site*, depending on whether the coordination unites a single data center or spans several data centers at remote locations.

With multiple-system automation, you can establish remote operations, called *centralized operations*, in which many of your systems have no operators present and do not need full operator interfaces. You operate the unattended systems remotely. You forward information about the conditions of the unattended systems to the central system, along with any problem reports that you cannot automate locally.

Stages of Automation

NetView automation encompasses a broad selection of techniques. These techniques can be divided into those used:

- On a single system
- In a multiple-system environment
- Specifically for non-NetView systems or non-SNA devices

For an example of the stages of automation, see "Example of a Staged Approach" on page 18.

Single-System Automation Stages

This section introduces the primary techniques of automating system and network management on a single system. These techniques are grouped into seven stages, according to the approximate order that you might implement them.

Figure 1 is the first of several illustrations in this chapter that show the staged introduction of automation to your system or systems. Later illustrations show the expansion of automation to multiple systems.

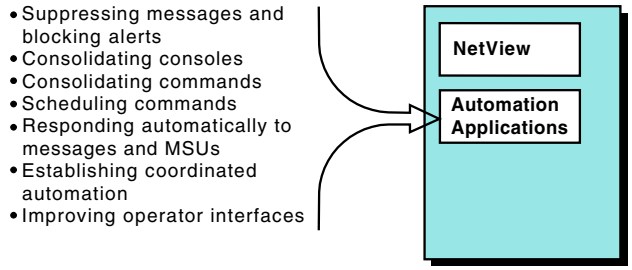


Figure 1. Adding Automation, with the NetView program on a Single System

The first three stages of automating a single system use NetView automation to increase the speed and accuracy with which operators process information as follows:

- Suppressing messages and blocking alerts
- Consolidating consoles
- Consolidating commands

The next three stages further reduce the workload of operators by having the NetView program automatically perform the following management tasks:
- Schedule commands
- Respond to messages and MSUs
- Establish coordinated automation

The final stage, improving operator interfaces, adapts your operator interfaces to the new environment and the reduced workload.

Suppressing or Revising Messages and Blocking Alerts

Even in a small data processing center, you probably receive many informational messages and alerts that operators simply ignore. In a larger center, you might receive hundreds of messages and alerts per second, only a small fraction of which contain data that operators use to make decisions.

A first step toward automated operations is to suppress or block routine messages and alerts. In this way, you can decrease the unneeded information your operators receive. They can then concentrate on important information.

Decreasing the number of messages can also decrease the load on the system. The system can then process important messages efficiently. You can continue to log the messages you suppress, keeping them available for debugging applications, auditing your automation, and similar activities.

You can suppress or revise messages by using the combined capabilities of NetView.

Use of the z/OS message processing facility (MPF) is still supported but many of its functions are superseded by the NetView revision table or the NetView automation table facilities. Use the revision table to suppress or revise most system messages. Only messages that require more complex actions, such as initiating an automation command list, must be passed to the NetView address space for automation table processing. You can also use the automation table to suppress unneeded NetView messages and VTAM messages received directly by NetView.

You can block unneeded alerts by first determining which problem records become events and which events become alerts. You can then set recording filters for the hardware monitor with the SRFILTER command. For more information about filtering commands with the SRFILTER command, refer to the NetView online help.

You can also add filtering statements to the NetView automation table. The NetView automation table contains processing options and automatic responses for incoming messages and MSUs. Automation-table statements can override recording filters for the hardware monitor.

Consolidating Consoles

After you reduce the flow of messages, you might be able to combine some consoles. For more information, see “Improving Operator Interfaces” on page 12 and Chapter 26, “Centralized Operations,” on page 373.

After suppressing unneeded messages, you can route the remaining messages to one or two consoles. You can display messages to operators in several ways.

Reducing Consoles: You can decrease the number of NetView consoles your operators monitor by moving information from the hardware monitor to another

interface. You can decrease or eliminate use of the hardware monitor by displaying alert information in other forms. For example, alerts that cannot be handled with an automatic response might be converted into messages or displayed on a full-screen panel. The automation table can initiate this process.

You can decrease the number of NetView consoles your operators monitor by decreasing or eliminating messages to operators and by increasing your reliance on the hardware monitor. When problems occur that automation cannot handle, you can generate hardware monitor alerts to inform your operators. You can then use the hardware monitor to display information that helps operators solve problems.

Consolidating Consoles through Message Collection: You can consolidate consoles by having the NetView program collect messages from a variety of sources such as:

- The operating system
- Master operator consoles of the Information Management System (IMS™) program or the Customer Information Control System (CICS®) program through the NetView terminal access facility
- Other subsystems and application programs
- Processor hardware consoles, through the Processor Operations component of the Tivoli System Automation for z/OS product
- VTAM application programs
- VTAM

After consolidation, you might have a few consoles close together in a central operating area or you might have just a single console. Then, a few operators or one operator can receive all of the messages that are essential for controlling the system and network. If you have more than one operator, you can display all of the messages that a specific operator needs, and no others, on one console for that operator. An operator does not have to watch several consoles at once or sift through another operator's messages.

Dedicating a NetView Console: You can consolidate the consoles used to manage the system and the network.

You can dedicate one NetView console to manage the system (using the command facility) and another console to manage the network (using the hardware monitor).

You can customize the NetView console, enabling operators who use other consoles to easily adapt to using NetView. In some cases, this ability to customize the NetView console depends upon the facilities of the operating system. Examples of console customization include coloring messages and changing message prefixes. (You can also use the revision table to perform console customization).

For information about console customization, refer to the *IBM Tivoli NetView for z/OS Customization Guide*.

Consolidating Commands

You can use simple command procedures to improve operations. Learn the sequence of commands your operators most commonly issue and write short programs (called *command procedures*) to issue those sequences automatically. An operator can enter the name of the command procedure, and the NetView program issues all of the commands in the sequence.

For example, you might create a simple command procedure to perform any of the following actions:

- Bringing a bank of direct access storage devices (DASD) online and mounting each volume with the correct attributes
- Re-establishing a set of telecommunication lines after repair
- Initializing a simple application, including verification of required DASD
- Dumping a filled system management facilities (SMF) data set or a dump data set
- Monitoring an operation checklist

Writing command procedures for your operators decreases the typing each operator must do. Operator productivity rises, and the chance of an error because of typing a command incorrectly decreases. The command procedures also provide a base for later automation, because you can use the NetView automation table or a timer command to automatically invoke some of the same procedures.

Scheduling Commands

If you want to issue a command at a particular time or issue a given command periodically, you can use command scheduling and the NetView timer commands. For example, you might need to shut down your applications at 5:00 p.m. to free processor capacity for a special activity, such as tape transfer, or you might want to check the status of certain tasks every 3 minutes.

The command that is issued can be a command procedure. Suppose you have written a simple command procedure that initializes an application program. If you want to initialize the program every day at 6:00 a.m., you can run your command procedure daily at that time.

By scheduling commands, you relieve your operators of the need to issue the commands manually. You can also perform actions when your operators are unavailable or repeat certain commands at a frequency that is impractical for human operators.

Responding Automatically to Messages and MSUs

Responding to event notifications, such as messages and MSUs, often consumes much of an operator's time. In many cases, the NetView program can automatically issue the operator's responses.

The NetView program provides an automation table that examines incoming messages and MSUs and responds to them with various actions. The NetView automation table can initiate any reaction you specify to a message or MSU, such as issuing a command. For example, the NetView program can automatically respond to all IOS150I messages, which indicate that a failed device is now available. The NetView automation table can issue an MVS VARY ONLINE command to bring the device back online.

When you have programmed the NetView program to reply automatically to the most common messages and alerts, you can suppress those messages and alerts from being displayed, eliminating the need for operators to view notifications for problems that automation is solving.

Establishing Coordinated Automation

Coordinated automation represents an advanced stage of automated operations. In coordinated automation, the NetView program continually tracks the preferred

state of each data-processing resource and the actual state. If the actual state differs from the preferred state, automation takes corrective action.

Programmers or operators set the preferred state of each resource. Resources include hardware components, such as channels, and software components, such as data sets or the address space for the MVS time sharing option (TSO). You can write command procedures to help operators examine and change the preferred state of a resource.

To determine the actual state of each resource, your automation can employ passive and active monitoring. *Passive monitoring* means waiting for messages and alerts that indicate status changes. *Active monitoring* means issuing commands to solicit status information. For example, you might set up a command procedure to run every 10 minutes and issue commands to check the states of important resources. By combining passive and active monitoring, you can ensure that automation has reliable, up-to-date information.

When your automation application program receives information about the state of a resource, it records that information, perhaps by updating a global variable. For example, if the IMS program fails, the value of a global variable that represents the state of the IMS program can be changed to DOWN.

When a preferred state or actual state of a resource changes, automation determines whether corrective action is needed. If so, automation can issue a command or command procedure to remedy the situation. It can also notify operators of the change of state.

Besides tracking preferred and actual states, you can track other information. For example, you might use a variable to indicate the automated action being taken for each resource. You can also specify the resources for which automation is responsible. Automation still monitors all resources, but attempts problem resolution only for those that you specify. With this technique, you can return to the manual control of any resource by changing a variable to stop part of your automation.

Automation samples are included with NetView. These samples demonstrate coordinated automation using NetView global variables. Before implementing coordinated automation, study the samples.

For information about the sample set, see Appendix I, “The Sample Set for Automation,” on page 577

Consolidating Automation with RODM

In addition to the techniques previously mentioned, you can consolidate automation using some of the capabilities provided by the Resource Object Data Manager (RODM) component of NetView. These RODM capabilities can help track resource information and help automate the resolution of problems. RODM can retain various types of information about resources, events, and the relationships among them. Because you specify complex relationships among pieces of information in RODM, the NetView program can determine interactions between multiple events and use them in analyzing and resolving problems.

Improving Operator Interfaces

Automated operations reduce the amount of human involvement needed to run a data-processing environment. Nevertheless, operators still need to be able to monitor the environment, examine the status of resources, and verify that automation is functioning correctly.

Furthermore, you need a mechanism for exception notification. *Exception notification* is the process of informing operators when automation routines encounter an event you have not yet automated or when the routines fail to resolve a problem.

Therefore, plan interfaces that give operators the information they need. You can present information to operators in the following forms:

- Messages from the command facility
- Alerts from the hardware monitor
- Status information from the status monitor and the NetView management console (NMC)
- Full-screen displays and help panels displayed with the VIEW command processor

Presenting Information in Messages

Messages are displayed on the NetView console to provide information about the NetView program and the products that the program is managing. The command facility, operated from the NetView console, displays messages. NetView operators monitor this facility most often in many environments that are not automated.

Automated operations can improve your use of the command facility. Message suppression decreases the number of messages displayed, making it easier to read the remaining messages. You can use the command facility for exception notification by creating a message whenever automation routines encounter a problem.

Console consolidation enables an operator to monitor more than one product, such as your operating system and NetView, from a single screen. In addition, you can use the automation table to hold important messages on the screen or to reissue messages with modified text.

Automation can also control the way messages are displayed to help the operator quickly recognize the importance of specific types of messages. For example, the system can present different classes of messages with different colors or highlighting. Also, different groups of messages can be formatted with different arrangements of information. You can make these and other changes in the appearance of the display by using a screen format member.

For information about the screen format member, refer to the *IBM Tivoli NetView for z/OS Customization Guide*.

The NetView program can store a specified limit of messages for display. If this number is exceeded, some of the oldest messages are discarded, but automation based on messages still continues, and all messages are logged.

Presenting Information in Hardware Monitor Alerts

The hardware monitor receives information in the form of events and alerts, and displays the information. The events and alerts are MSUs and other data structures that flow into NetView. Alerts primarily indicate that network hardware is experiencing problems.

Note: The hardware monitor submits only unsolicited MSUs to the automation table.

You can continue to use the hardware monitor in conjunction with other facilities that provide resource information for display, just as you would in an unautomated environment. To do so, you can have one or more consoles present alerts to operators from the hardware monitor. The operators can use the alerts to manage network problems. You can display automation status and other information on a separate console, in another form, such as messages or full-screen panels.

Deciding How to Use the Hardware Monitor: Operators can display problem descriptions, lists of probable causes, and lists of suggested actions. The hardware monitor also:

- Maintains a history of reported problems
- Provides viewing filters that determine which operators see which alerts
- Enables you to send information to the Information/Management program, to a user-defined external log, or to a system management facilities (SMF) external log

Generating Alerts: To generate your own alerts, use the GENALERT command, the program-to-program interface, or the management services (MS) transport of NetView. After suppressing or automating the majority of the messages you receive, use alerts to notify operators of the remaining messages and of any problems that your automation encounters.

You can issue the GENALERT command from the automation table, when certain messages are received, or from command procedures. You control the contents of the alerts you generate, including descriptions, suggested actions, telephone numbers of people to contact, and other information that fits your environment.

You can also write a REXX command that formats the alert and sends it by way of the program-to-program interface (PPI) PIPE stage.

Presenting Information in Beeper/E-mail Actions

Using the INFORM command and its associated policy definitions, you can generate beeper or email actions to notify appropriate personnel of key events or actions. For example, you can use beeper or email actions for off-shift hours or for support of remote locations.

For more information, see the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)*.

Presenting Status Information

The status monitor and the NetView management console (NMC) can track network status. You can determine status without remembering past sequences of messages or issuing query commands. The status monitor provides status information for display in text form on the status monitor panel. The NetView management console (NMC) provides status information in graphic displays of your network on the screen of a workstation. While your automation is responding to events and keeping resources active, operators can efficiently monitor the network with status displays.

The NetView management console (NMC) can display information on a workstation attached to an MVS system. You can run the facility on a single

system, but it is most useful in a multiple-system environment. To display information about other systems graphically, you can forward status information to an MVS system.

For a description of forwarding status information, see Chapter 26, “Centralized Operations,” on page 373.

Displaying Information on Full-Screen Panels

For greater flexibility in designing interfaces, you can create full-screen panels that are displayed from a command procedure. Full-screen panels provide many color and highlighting options, which can be used for displaying status information, exception notification, or both.

The NetView program automatically defers displaying messages during the display of full-screen panels. However, automation and message logging continue while the panels are displayed.

You can create full-screen panels with a standard editor, such as the Interactive System Productivity Facility (ISPF).

After you have created a panel, you can use the VIEW command to display it from a command procedure. In addition to displaying data with the color and highlighting options you specified, the VIEW command can accept input in fields you have designated. This input is passed back to the command procedure, enabling your automation routines to communicate with the operator, interactively.

For examples of how to display full-screen panels, use a standard editor to review NetView command lists that are using the VIEW command. Such command lists include BROWSE, TUTOR, and DISG.

The HELP command also uses the VIEW command; therefore, you can create help panels or modify existing NetView help panels. You can display information that documents the automation you create, assists operators in using your command procedures, and presents customized information that reflects your network environment.

Propagating Single-System Automation

The first stage of multiple-system, network-wide automation is to propagate single-system automation to all of your NetView systems (see Figure 2 on page 15). You might need to design new automation for each system because different applications or devices can be installed on each system in the network. However, if you have implemented single-system automation on one system, you might be able to propagate much of that automation onto other systems.

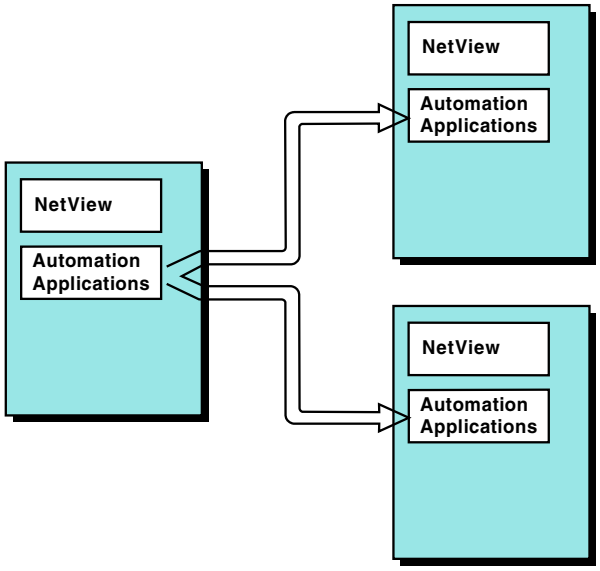


Figure 2. Propagating Automation to Additional Systems

If you customize the copied automation for the new systems, the number of changes needed depends on how different the new environment is from the one on which you developed the automation.

Use a flexible design for propagation. See "Designing for Expansion and Propagation" on page 52 for information about how you can design portable automation.

Centralizing Operations

In a centralized operation that results from single-system consolidation, you can route information from many systems, spread across the network, to a single console or set of consoles. Operators no longer need to run each system from separate consoles.

To avoid overburdening the communication between systems, do not send problems to another system until you have locally automated responses to as many problems as possible. Forward only two types of information:

- Information about the condition of the individual systems (for display to operators)
- Information about problems that the individual systems cannot automatically resolve without assistance from another system

These problems include those that require operator attention and those that require restarting the processor, the operating system, or NetView.

As shown in Figure 3 on page 16, you can designate one system as the focal point for receiving forwarded exceptions from distributed data systems. By logging on to the NetView program at the focal-point system, operators can manage a group of systems, an entire data center, or several data centers.

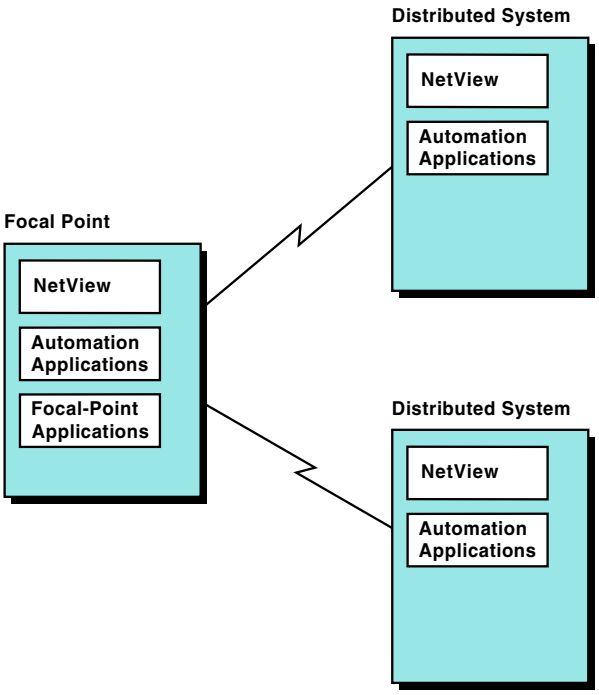


Figure 3. Forwarding Exceptions that Local Automation Cannot Handle

Use of Focal Points in Centralized Operations

Whether you perform single-site or multi-site automation, the focal-point system performs two sets of actions:

- The focal-point system automates its own system and network management. For this, implement the same types of single-system automation that you are using on other systems.
- The focal-point system automates information that comes from the systems that report to it, which are known as ***distributed, target,*** or ***entry point*** systems.

Information that cannot be automated by either the target systems or their focal point is presented to operators at the focal point system.

With an arrangement of focal-point and distributed data systems, you might not staff certain data centers during off shifts and remotely operate the data centers. During those shifts, you can forward information from the distributed systems at unattended data centers to a focal-point system at an attended data center. However, running an automated data center unattended might still require some manual intervention for such tasks as mounting tapes and handling printers.

See “Automation-Related Functions and Services” on page 34 for ways to reduce the need for manual intervention.

You can use the NetView program to forward messages, alerts, and the status information used by the NetView management console (NMC). By tracking the focal points of the application programs, the NetView program can also assist in information forwarding for application programs that use the management services transport.

Because an outage in the focal-point system can interrupt the management of many other systems, select a reliable system for your focal point. You can also designate a backup focal point to take control in the event of a planned or unplanned outage.

See "Choosing Focal Points" on page 57 for criteria to use in selecting a reliable system for your focal point. For information about selecting a backup focal point, see "Using a Backup Focal Point" on page 58.

Establishing Remote Operation

When you implement the stages previously described, your distributed systems can automate most operations. Information about the remaining operating activities is forwarded to a focal point, where automation and your centralized operations staff handle situations that do not require manual intervention at a remote location.

You can complete multiple-system automation by automating actions that involve the hardware and system consoles of the target processors. Actions that involve these consoles include initialization, configuration, and shutdown of target processors. You can use IBM System Automation for z/OS to accomplish these actions for most IBM processors. Use of System Automation for z/OS to remotely initialize target systems is shown in Figure 4.

See "System Automation/390 Programs" on page 33 for an overview of System Automation for z/OS capabilities.

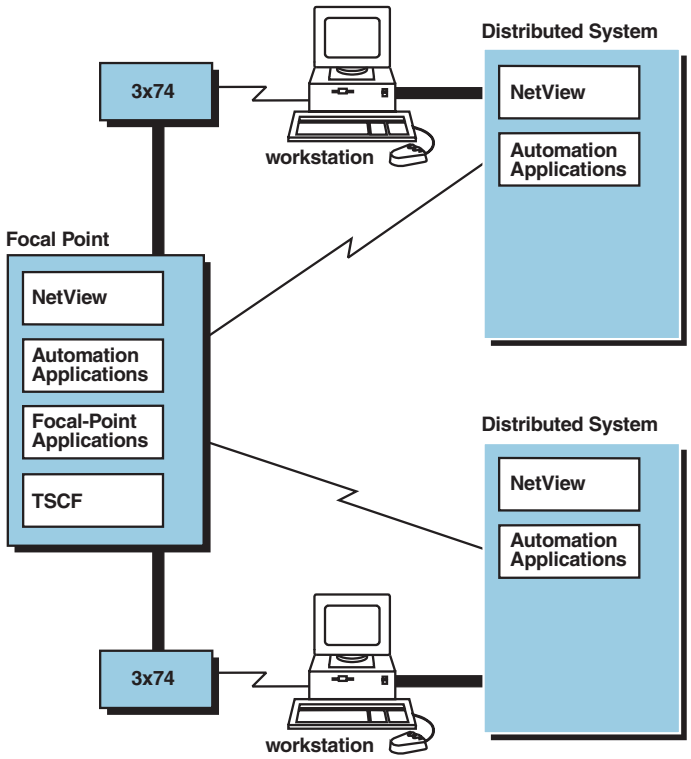


Figure 4. Remotely Initializing Target Systems

System Automation for z/OS can control Enterprise System/4381, Enterprise System/3080, Enterprise System/3090, and most Enterprise System/9000 (ES/9000) processors, but cannot remotely initialize 9370 processors.

However, the Automated Power Control (APC) feature of the 9370 enables you to automate initial program loads (IPLs). You can set a timer to turn on power to the 9370, which then performs an IPL and starts the operating system.

The operating system can start NetView, which then establishes your system and network automation. APC also enables you to turn on power remotely through a modem or other RS-232 device for initialization in recovery situations.

The 9370 system also offers a Remote Operator Facility (ROF). This facility gives you a remote-console capability and enables you to control distributed 9370s from your central site. ROF runs on a workstation and enables operators at the central site to control the hardware and operating system of the remote 9370 service processor through a dialed connection.

Note: System Automation for z/OS does not support the rack-mounted ES/9000 processors (models 120, 130, 150, and 170). You can initialize these processors remotely with the NetView RUNCMD command by sending initialization commands to the processor console of the ES/9000. By writing command procedures to send these initialization commands, you can ensure correct entry of the RUNCMD command.

For information about ES/9000 processors, refer to *Enterprise System/9000 Models 120, 130, 150, and 170: Managing Your Distributed Processors*.

Automating Non-NetView Systems and Non-SNA Devices

You can use the NetView program to automate many target systems, even though the target systems are not running the NetView program. You can also use the NetView program to automate many network devices, even though the devices do not use SNA protocols or report to VTAM.

NetView automation capabilities for a non-NetView system or non-SNA device depend on the capabilities of the system or device. The system or device must be able to send problem reports and other information in a form that NetView can interpret (such as messages or MSUs), and the system or device must be able to receive commands from the NetView program.

You can directly automate some products using the NetView program and indirectly automate other products by using an existing NetView interface or by writing your own interface. The NetView program interfaces with the AIX NetView Service Point program and with the Tivoli NetView program, which is used with the AIX NetView Service Point program. See “Examples of Using NetView Program Interfaces” on page 33 for descriptions of AIX and other NetView interfaces.

Example of a Staged Approach

In a typical environment in which operators manage systems by monitoring a steady stream of event notifications such as messages and alerts, operators observe each event and respond if the event indicates a problem. This operating technique can be described as an event-monitoring environment.

In this example, the following sequence describes a staged approach for automating the systems in your enterprise. This approach moves from an event-monitoring environment to an exception-monitoring environment, and from there to a centralized-operations environment. In the centralized-operations environment, automation responds to the majority of events and problems.

For the few that remain, notifications are sent to a single focal-point system. The NetView system, as the focal point, describes the problems using efficient interfaces, enabling operators to understand the situation quickly and to take appropriate action.

To teach your operators about the new environment, document the way your network is automated; then update your procedures or run books.

Stage 1: Suppress Messages and Filter Alerts

Block out unneeded notifications. Allow time after setting up this stage for operators to become accustomed to monitoring the environment with limited notifications. Notify your operators before this procedure takes place.

Stage 2: Consolidate Consoles

Fewer consoles are needed for monitoring messages, and the message rate for each console diminishes. Forward unsuppressed messages from your operating system to NetView.

Stage 3: Consolidate Commands

Consult operators and other sources of information to identify the procedures and sets of commands that operators most commonly use to perform their tasks. Then, write simple command procedures that enable operators to efficiently perform their tasks.

Stage 4: Schedule Commands

Using command scheduling, issue timer commands to perform repetitive operator tasks.

Stage 5: Create Automated Responses to Messages and MSUs

Use the NetView automation table to issue automated responses to common messages and MSUs. This can reduce the rate of messages and alerts displayed to operators and diminish the role of the operators in minute-by-minute system and network operations.

Stage 6: Coordinate Monitoring and Reactivating

Create a coordinated system to monitor and reactivate the products that your operators have been managing. In this stage:

1. Track the state of each program or resource using, for example, global variables or RODM.
2. Monitor messages and alerts to determine in what state each resource is.
3. Issue command procedures to resolve any differences.

Because this stage eliminates the last of the repetitive, mechanical tasks that operators were performing, you have now moved from event monitoring to exception monitoring. Operators no longer view a continuous stream of messages and alerts. Instead, they view only summarized status information and notifications of exceptional problems that automation cannot handle.

Stage 7: Improve Operator Interfaces

Operators no longer continuously monitor the command facility and the hardware monitor for messages and alerts. Instead, employ alternative interfaces that are more suited to status display and exception notification, such as full-screen panels displayed with the VIEW command.

Stage 8: Implement Multiple-System Automation

Go from single-system automation to multiple-system automation. To automate a multiple-system enterprise, first ensure that you propagate single-system automation to every NetView system.

Stage 9: Centralize Operations

Choose one system to be the focal point. Then, forward exception notifications from other systems to your focal point. Begin operating all of your systems from the single focal point, eliminating the need for operators at the other systems. If your enterprise is spread across several data centers or several sites, you also perform remote initialization.

Stage 10: Extend Automation to Additional Machines and Devices

With the Automated Operations Network (AON) component of NetView, you can manage almost any data-processing equipment, including non-IBM systems and non-SNA devices. See Chapter 31, “Using Automated Operations Network,” on page 441 for specific information.

Chapter 2. Overview of Automation Products

This chapter describes the major products used in NetView program automation, their roles in an automated environment, and how they relate to one another. Specifically, this chapter includes overview information about:

- NetView Program Automation Facilities
- "Operating-System Automation Facilities and Interactions with the NetView Program" on page 27
- Other IBM programs that provide automation
- "Examples of Using NetView Program Interfaces" on page 33
- "Automation-Related Functions and Services" on page 34

NetView Program Automation Facilities

The NetView program is central to automated operations. It can receive information from the other products in your enterprise, process that information in ways you specify, and issue automatic responses.

Several NetView program facilities are important to automation, whether you are automating a system, a network, or multiple enterprises. These facilities enable you to customize and use the NetView program to perform the types of automation described in Chapter 1, "Introducing NetView Automation," on page 3. The NetView program provides the following major facilities for creating your own automation applications:

- "Command Lists and Command Processors"
- "Timer Commands" on page 23
- Automated tasks ("Autotasks" on page 23)
- "Automation Table" on page 24
- "Message Revision Table" on page 25
- "Resource Object Data Manager" on page 25 (RODM)
- "Installation Exits" on page 25 for automation
- "MVS Command Revision" on page 26
- "Automated Operations Network (AON)" on page 27
- "Status Monitor" on page 27

Command Lists and Command Processors

With the NetView product, you can write programs and use them as if they were NetView commands. These programs are classified according to the language in which you write them.

Command lists are sets of commands and special instructions that you write in the Restructured Extended Executor (REXX) language or the NetView command list language.

Command processors are assembled or compiled modules that you write in assembler, PL/I, or C language. Command lists and command processors are used extensively in automation.

A command list or command processor can either assist an operator with a task or perform a procedure without operator intervention. When you write a command

list that performs the tasks of several NetView commands, operators can accomplish a complex task with a single command.

To perform a procedure without operator intervention, use the NetView automation facilities to start a command list or command processor. For example, the automation table or a timer command can start a command.

Choosing a Language

In planning for automated operations, choose a language or set of languages for writing your command procedures. For a description of the capabilities of each language, refer to the *IBM Tivoli NetView for z/OS Customization Guide*.

Because only assembly language gives you access to NetView control blocks, you must use assembly language for any intricate automation that examines or modifies control-block information. However, most other automation routines are easier to write in the other four languages. The other four languages provide several functions that are of special value to automation, as described in "Automating with Command Procedures."

Automating with Command Procedures

A *command procedure* is a command list, or a command processor written in PL/I or C language. This section summarizes automation functions available to command procedures.

Obtaining Message and Management Services Unit (MSU) Information: The automation table can respond to a message MSU by calling a command procedure. The automation table can extract information about the message or MSU to be passed to the command procedure in the form of parameters. For example, the automation table might capture the MVS system ID or job name of a message and pass it to a command procedure for use in the response.

Alternatively, the command procedure itself can extract information about the message or MSU. A command procedure issued from the automation table (or a command procedure issued because an MSU was received on the management services transport) can obtain the contents of the message or MSU that caused it to be issued.

Using Global Variables: Automation often requires cooperation among many command procedures and coordination with the automation table. Global variables provide a convenient way to transmit information from one command procedure to another and to the automation table.

Global variables are variables that retain their values between uses of command procedures. You can use them to share information between command procedures running on one task (*task global variables*) or on different tasks (*common global variables*). The automation table also can read global-variable values. To change a value, the table must call a command procedure.

The NetView program gives you the option of saving global variables to an external database. Saving variables can help recovery from any outage because you can restore the variables when you restart the NetView program.

Accepting Parameters: Command procedures can also accept parameters. For example, operators can enter parameter information after the name of the command procedure when using a command procedure from a terminal.

Automation facilities, such as other command procedures or the automation table, can also specify parameters when using your command procedure. For example, you can write a recovery command list that uses parameter variables to accept the name of the application program to restart, the start command for the product, and the amount of time to wait for the application program to initialize.

Obtaining Environment Information: Your command procedures can get information about the system and the operating environment. For example, a command procedure can obtain such data as:

- Operating system in use
- Domain ID
- Current date and time
- Type of task that is running the procedure

Interacting with the System and Network: Command procedures can pass commands and messages to the operating system, enabling you to perform system automation. For information about how command procedures pass commands and messages to the operating system, see "Operating-System Automation Facilities and Interactions with the NetView Program" on page 27.

Command procedures can also pass commands to the VTAM program to control the network.

Waiting: Command procedures can issue commands to solicit information and wait for the responses before taking further action. For example, an automation procedure that restarts a failed application program might issue a query command afterward and wait for verification that application-program cleanup is complete.

Timer Commands

You can use timer commands to initiate automated actions. Both operators and automation procedures can issue timer commands to schedule other commands, command lists, and command processors. The NetView program provides the following timer commands:

- The AT command schedules another command for execution at a specified time.
- The AFTER command schedules a command for execution after a specified delay.
- The EVERY command schedules a command to be issued repeatedly after specified intervals.
- The CHRON command enables you to perform complex timer automation functions.
- The LIST TIMER and PURGE TIMER commands enable you to examine or cancel commands that you have scheduled.
- The TIMER command enables you to add, change, and delete timers using full screen panels.

For information about the using the timer commands, see Chapter 11, "Timer Commands," on page 115 or the NetView program online help.

Autotasks

An *autotask* is an operator station task (OST) that does not require a terminal or an operator. Like other OSTs, autotasks can receive messages and issue commands. Autotasks are limited only by the fact that they cannot run full-screen applications. Unlike other OSTs, autotasks can run without the VTAM program being active.

This ability, along with the fact that autotasks can do most of the tasks you can do from an operator's OST, makes autotasks useful for automation.

You can define one or more autotasks for automation and have them started during NetView initialization. Then the automation table, command lists, command processors, and timer commands can all issue commands under your autotasks. The autotasks can receive messages and present them to the automation table or to installation-exit routines. Thus, many of the other facilities for automation can use autotasks.

Autotasks are the preferred task for a wide variety of automation purposes. When you route work to an autotask, you can avoid problems that might occur if you used an operator's OST. For example, the operator might be logged off or using the OST for other work.

Automation Table

The NetView automation table enables you to specify processing options, for incoming messages and MSUs, and to issue automatic responses. The table contains a sequence of statements that define the actions that the NetView program can take in various circumstances.

To determine the automated actions that the program can take, your automation statements can examine any field in an MSU and any part of message text. (In multiline messages, only the ACQUIRE condition can examine lines after the first line.) Statements can also examine IDs of messages, resource hierarchies of MSUs, domain IDs of either messages or MSUs, and many other attributes, such as occurrence thresholds. Operands for AND and OR are recognized, so you can specify several comparisons in any combination.

You can specify any number of actions for the NetView program to take when an incoming message or MSU matches your conditions. Actions can be commands, command lists, and command processors. For simple responses, a single command might be sufficient, such as a NetView command, a VTAM command, or a system or subsystem command. For more complex responses, you can write command lists or command processors. The automation table specifies the task under which the action is performed, enabling you to run automation procedures under an autotask.

Actions also include setting message-processing and MSU-processing options. For any particular message, you can use message-processing options to specify such things as whether:

- The message should be suppressed (and if not, to which operator it should be displayed)
- A message should be held on the operator's display (messages requiring operator attention)
- Automation should process the deletion request for a specific action message
- The message should be logged in the system, network, or hardcopy log
- An audible alarm should sound to call attention to the message

MSU processing options apply to MSUs that are directed to the hardware monitor. These options enable you to override recording filters. For any particular MSU, you can use MSU processing options to specify such things as whether

- The hardware monitor records the MSU in the event database
- The hardware monitor records the event in the alert database

- The NetView program forwards the alert to a focal point

You can also specify highlighting options, such as color and underlining, to help focus operator attention.

Use the AUTOCNT command to generate automation table usage reports for your system. You can use the reports to analyze automation table statements to see the matching frequency. You can move frequently matched statements toward the top of the table so that less checking of unmatched criteria takes place.

You can also determine whether unmatched statements must be deleted from the table or changed because of logic errors. Automation table usage reports enable you to determine the level of automation taking place on your system. These statistics can be useful in reports for management purposes.

You can use the AUTOTEST command to test an automation table. You can perform this test using either current messages and MSUs or prerecorded messages and MSUs. For more information, see Chapter 15, "The Automation Table," on page 147.

You can use the AUTOMAN command to manage your automation tables. Using this function, you can enable or disable automation table statements, load and unload automation tables, and display their status. For more information, see "Managing Multiple Automation Tables" on page 248.

Message Revision Table

You can use the message revision table (MRT) to examine messages flowing in the system and make changes to certain aspects of the messages. The MRT is active as long as the SSI address space is active, even when the NetView program is not active. See Chapter 13, "The Message Revision Table," on page 127 for more information about using this function.

Resource Object Data Manager

The NetView program can use the Resource Object Data Manager (RODM) to hold many types of information about network and system resources. RODM keeps this information in high-speed storage so the information can be retrieved and updated quickly. For automation, you can use the information in RODM in conjunction with other automation facilities to assist in determining the appropriate responses to messages, MSUs, and status changes.

RODM uses small programs, called *method procedures* (or *methods*), to perform many functions that retrieve, update, and manipulate information within RODM. An application program interface (API) is also provided by RODM so that application programs can gain access to the information in RODM. Through this API and the method procedures, the NetView program can retrieve and update the resource information in RODM, as needed.

For information about how RODM can be used in automation, see Chapter 8, "Automation with the Resource Object Data Manager," on page 77.

Installation Exits

In the NetView program, you can write routines that take control of processing at certain points. These points, called *installation exits*, enable you to alter the normal course of NetView program processing. Installation exits that are important to automation are:

- DSIEX02A
- DSIEX16
- DSIEX16B
- DSIEX17
- XITCI

For details about writing installation-exit routines in assembly language, refer to *IBM Tivoli NetView for z/OS Programming: Assembler*. For details about writing installation-exit routines in PL/I and C languages, refer to *IBM Tivoli NetView for z/OS Programming: PL/I and C*.

Using DSIEX02A

If you write a routine for DSIEX02A, the routine receives control just before a message goes to the automation table. The routine can alter, replace, or delete the message. If you alter or replace the message, the new version of the message goes to the automation table. To increase processing speed, write this installation-exit routine in assembly language. You can also use PL/I or C language.

Using DSIEX16 or DSIEX16B

You can use the exits to modify message processing options, reformat messages, and alter information in MSUs. Both of these installation-exit routines must be written in assembly language.

Using DSIEX17

A routine written by you for DSIEX17 that receives control as soon as a message or delete operator message (DOM) is received from the MVS system. Your routine also receives control when a message or DOM is received from user calls to assembly language service DSIMMDB or to PL/I and C language service CNMPMDB.

Refer to *IBM Tivoli NetView for z/OS Programming: Assembler* for information about DSIMMDB. Refer to *IBM Tivoli NetView for z/OS Programming: PL/I and C* for information about CNMPMDB.

Your routine can delete a message or DOM, or can modify the text and attributes of a message. If you write a routine for this installation exit, use only assembly language.

Your routine can also be used to mark messages that were issued as action messages from MVS for which no DOM is expected.

Using XITCI

If you write a routine for exit XITCI for the hardware monitor, your routine receives control when the BNJDSERV task receives data. With XITCI, you can modify any data entering the hardware monitor. The XITCI exit routine can be written in PL/I, C, or assembly language.

MVS Command Revision

The NetView MVS Command Revision function enables you to examine, modify, or reject an MVS command. For more information see Chapter 14, “The Command Revision Table,” on page 135.

Automated Operations Network (AON)

You can use the Automated Operations Network (AON) component of the NetView program to provide policy-based network automation for VTAM SNA, and TCP/IP resources.

AON components intercept alerts and messages that indicate problems with network resources. AON can recover failed resources and monitor resources until they recover. AON can keep a record of resource failures to track recurring network problems.

AON uses most of the functions described in this manual to provide drop-in, policy-based automation.

For more information, see Chapter 31, “Using Automated Operations Network,” on page 441.

Status Monitor

You can use the NetView status monitor to automatically reactivate failing network nodes. The MONON and MONOFF commands start and stop this form of automation. You can enter MONON and MONOFF from a terminal or have your automation application program issue them. Use statements in the VTAMLST data set members to control which resources the status monitor automates.

If you want to do your own automation when a node changes status, you can add a SENDMSG statement to DSICNM (CNMS5001). Thereafter, a change in the node status generates a CNM094I message, which you can process with the automation table. For details about SENDMSG, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Operating-System Automation Facilities and Interactions with the NetView Program

In system automation, the operating system provides some automation facilities and can interact with the NetView program for additional automation. The NetView program receives information from the operating system, processes that information with the NetView automation facilities, and sends responses to the operating system as commands. Also, in some interactions not directly related to automation, operator commands can be sent between the operating system and the NetView program.

Automation on MVS Systems

The NetView program can automate responses to messages and MSUs from the operating system and from MVS application programs. The operating system performs its automation tasks before it sends messages to the NetView program for further automation. Also, NetView commands can be sent from system operators to the NetView program, and MVS commands can be sent from the NetView program to the operating system.

System messages that you can direct to the NetView program (either through the subsystem interface or to the NetView program's extended multiple console support consoles) include write-to-operator (WTO) and write-to-operator-with-reply (WTOR) messages. Some messages issued by application programs (such as CICS and IMS programs) to their consoles are not available through the subsystem

interface or extended multiple console support (EMCS) consoles. To automate responses to such messages, you can use the NetView program's terminal access facility.

Automating Responses to Messages

To suppress or revise system messages, use the NetView message revision table. To automate responses to messages, you can mark the messages in the NetView message revision table for delivery to the NetView program or for "NetView only" (NETVONLY).

Solicited messages (command responses) are sent to the NetView program through extended MCS consoles. Unsolicited messages are sent to the NetView program through the subsystem interface (SSI). Messages marked as NETVONLY are always sent to the NetView program through the SSI.

Figure 5 on page 29 shows message flow between the z/OS system and the NetView program when the subsystem interface is used. Figure 6 on page 30 shows the command flow.

As indicated in Figure 5 on page 29, messages first flow to the Message Processing Facility (MPF), which you can use to set several processing options. Next, the messages are sent through multiple console support. Messages destined for most subsystems are broadcast to the subsystems through the subsystem interface.

Messages are processed by the Message Revision Table (MRT) even if they are not destined for the NetView program. Other subsystem interface (SSI) programs can also examine and alter messages.

The NetView program compares each message that it receives to entries in its automation table and issues any automated response that you have specified.

Note: You can use the endcmd.close.leeway statement in the CNMSTYLE member to specify how long commands can run after a CLOSE IMMED, CLOSE STOP, or an MVS STOP (P) command is entered for the NetView program. During the period defined by the endcmd.close.leeway statement, message automation remains active, but no new commands are queued. If a CLOSE STOP command is issued and message queuing is enabled for the SSI, then MVS message queuing ends as soon as the CLOSE STOP command is recognized.

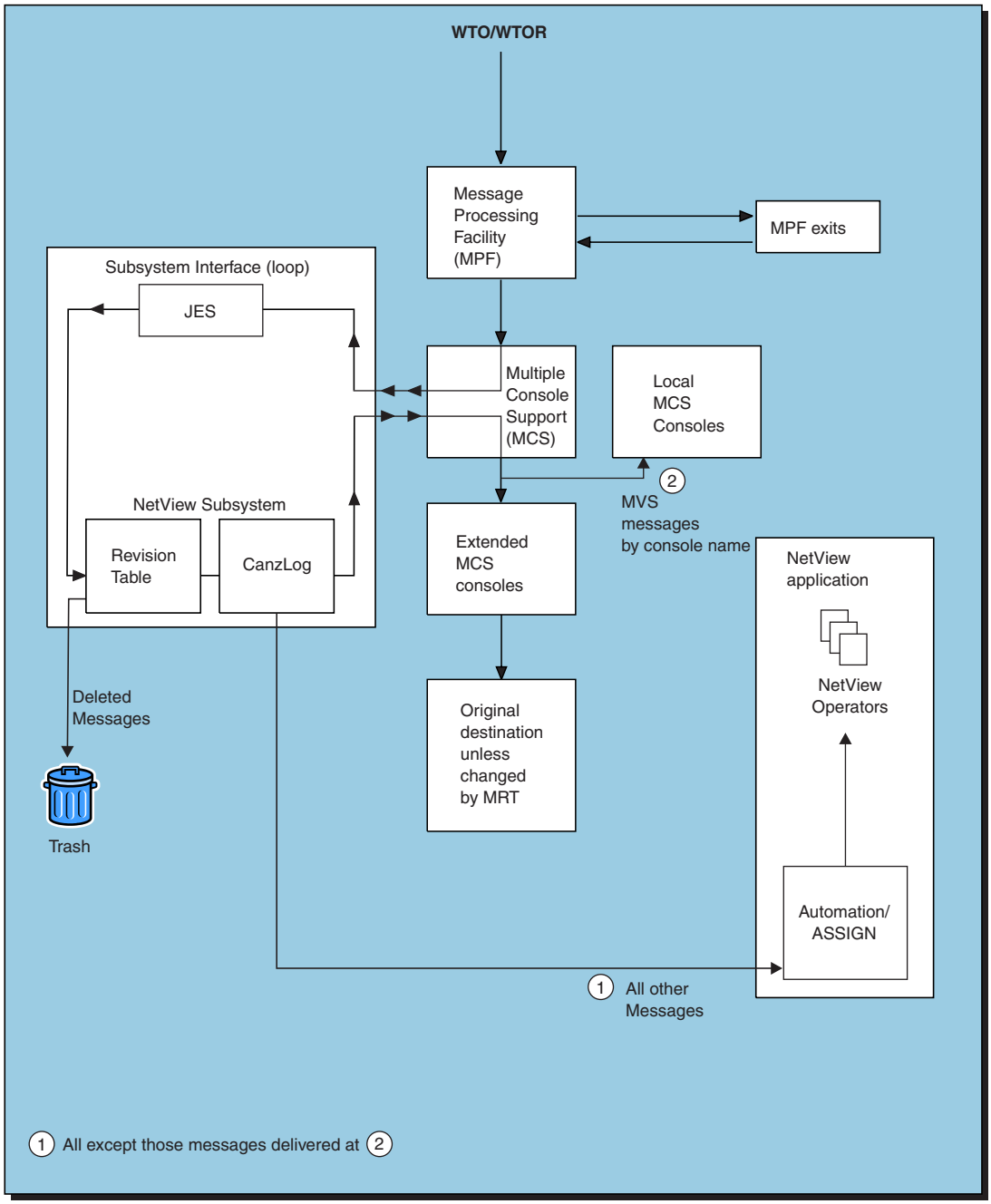


Figure 5. Message Flow between the z/OS System and the NetView Program through the Subsystem Interface

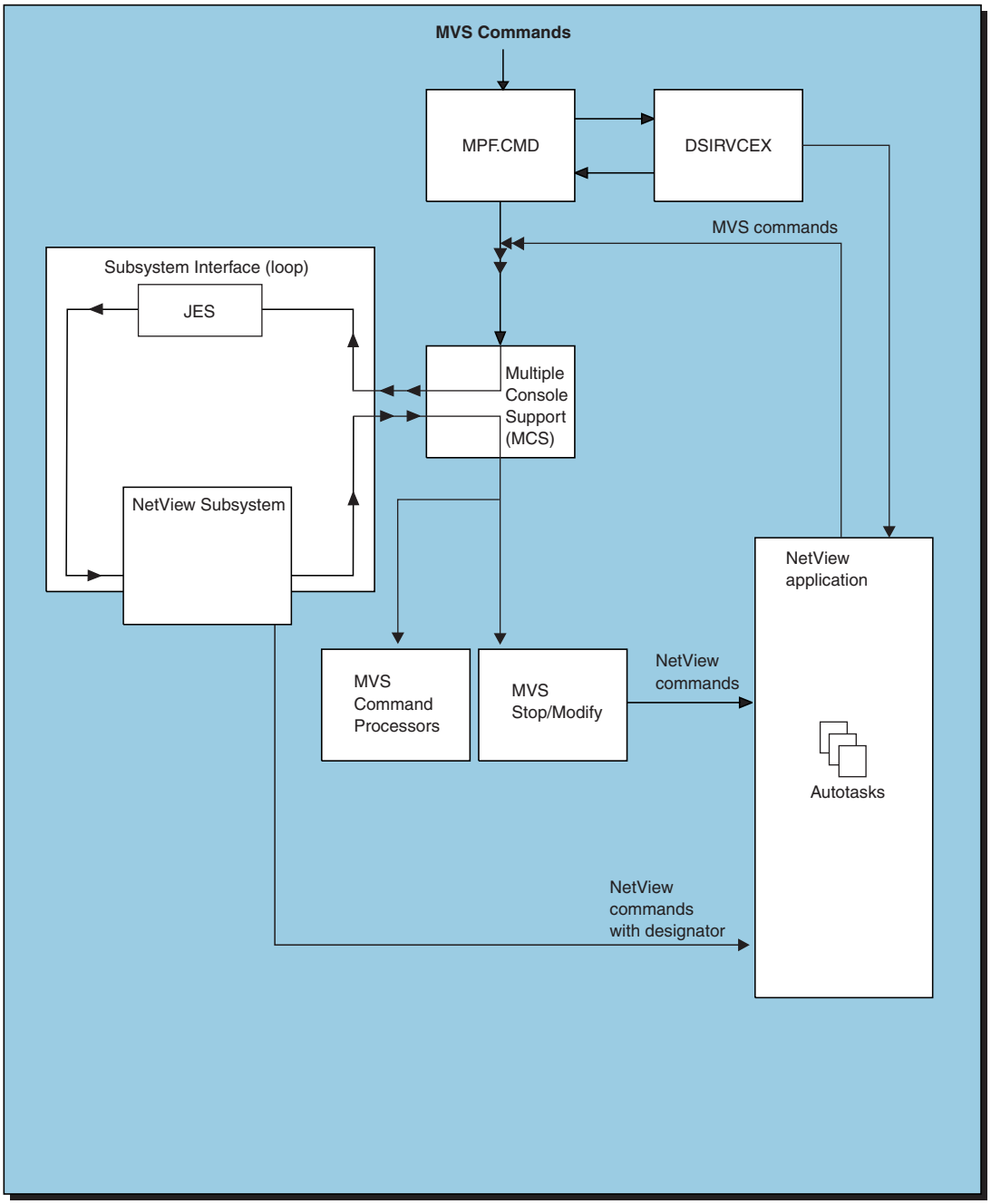


Figure 6. Command Flow between the z/OS System and the NetView program

Setting Options for Automating with either the Message Processing Facility (MPF) or the Message Revision Table (MRT)

To automate responses to messages, MPF can be used to set options, such as whether a given message is displayed to operators, suppressed, or marked as eligible for automation. In the NetView for z/OS Version 5, Release 2 product, the message revision table can also be used to provide these functions and more.

By default, the NetView program receives each message that you mark eligible for automation and sends it through the automation table. You can save processing time by marking as eligible only those messages that you want to be automated.

Messages leaving MPF or the message revision table can flow to the NetView program through the subsystem interface.

After passing through MPF and the NetView MRT, messages can be directed to the NetView program. If there is a NETVONLY action specified in the MRT, the message is sent directly to the NetView program. In this case, there is no further system action on the message. Also, if automation was requested in either the MPF or the MRT, a copy of the message is routed to the NetView program.

When a message flows to EMCS through its EMCS console, additional information is available, usually in a Message Data Block (MDB) which is attached to a system message at IFRAUVPT. For example, the originating SYSPLEX name can be found in this MDB. The NetView program preserves any color information that was set by an MPF or Message Revision in system messages, regardless of how the messages are delivered.

See Chapter 6, "Automation Using MVS Extended Multiple Console Support Consoles," on page 65 for information about extended multiple console support consoles.

Automating a Sysplex

In addition to providing automation for a single MVS system, the NetView program can provide automation for MVS systems that are interconnected in a sysplex configuration. An MVS *sysplex* configuration consists of multiple MVS systems working as a single system by sharing functions and programs.

If the NetView program is operating in a sysplex environment, the NetView program automates only those messages that are directed to it by console name or route code. See "DoForeignFrom Statement" on page 129 for disposition of messages from outside the local MVS image.

See Chapter 7, "Automation in an MVS Sysplex," on page 73 for more information about sysplex automation.

Automating Responses to MSUs

To automate responses to MSUs from another MVS application program, you can send the MSUs to the NetView program through the NetView-to-program interface and the management services (MS) LU 6.2 transport. The program-to-program interface can receive both network management vector transports (NMVTs) and control point management services units (CP-MSUs). The NetView automation table can automate responses to both types of MSUs.

Entering NetView Commands from MVS Consoles

There are three ways to issue NetView commands from an MVS system:

- The MVS MODIFY command
- The NetView subsystem designator character
- Through the Command Revision Table (CRT)

For more information, see "Issuing NetView Commands with the MVS MODIFY Command," "Issuing NetView Commands with the Designator Character" on page 32, and "Issuing NetView commands through the Command Revision Table (CRT)" on page 32.

Issuing NetView Commands with the MVS MODIFY Command: If you have an autotask associated with the system console, you can enter NetView commands from the console using the MVS MODIFY command. To do this, enter:

```
f procname,command
```

Where *procname* is the name that your system programmer assigned to the cataloged procedure for the NetView program such as CNMCNETV, and *command* is the NetView command you want to issue. For example, to display the MVS console names and IDs used by the NetView program, enter:

```
f procname,disconid
```

Issuing NetView Commands with the Designator Character: To enable system operators to issue commands to the NetView program, you can associate multiple console support consoles with NetView autotasks. Refer to the AUTOTASK command in NetView online help for information about associating multiple console support consoles with autotasks.

As indicated in Figure 5 on page 29, operator commands issued from multiple console support consoles flow to subsystems through multiple console support and the subsystem interface. A subsystem processes only those commands that are preceded by its assigned character. For example, JES2 typically processes all commands that are preceded by a dollar ($) symbol.

The NetView program processes all commands that are preceded by a designator character string. If you are using the NetView program's CNMSTYLE processing to start your SSI, use the MVSPARM.Cmd.Designator statement to set this. You can also see sample CNMSJ010 for other methods. If you are using more than one NetView program on a system, and these NetView programs are to process NetView commands entered at a multiple console support console, assign a different designator character string for each NetView program on the system. The sample uses the subsystem name as the designator character string. The default is the percent (%) character.

If a NetView autotask is associated with a multiple console support console and a NetView command is issued from that console, the command is called by the NetView autotask associated with the console. You can call NetView command procedures and commands from the multiple console support console.

Issuing NetView commands through the Command Revision Table (CRT): To enable system operators to issue commands to the NetView program using the Command Revision Table, prepare the CRT to recognize the commands to be routed and call a NETVONLY action for the command. The REXX procedure that is invoked can examine the SAF user name (and group) for verification and can take any action wanted. For example, consider identifying the LINKNV command to use the NETVONLY action, calling a REXX procedure that invokes an AUTOTASK command to enable the use of the MODIFY command or the designator character for the particular console from which the LINKNV command was issued.

Issuing MVS Commands from the NetView program

You can issue MVS commands from the NetView program to the MVS system by preceding each MVS command with the NetView command MVS. Either a NetView operator or an autotask can issue the NetView MVS command.

In addition to preceding an MVS command with the NetView command MVS, you can define command definitions for individual command verbs. For more information about defining command definition statements for MVS, see the CNMS6401 sample.

To protect against the unauthorized use of MVS commands you can use the command authorization function of the NetView program. Also, you can use the OPERCMDS class of the IBM Resource Access Control Facility (RACF®) or a compatible security product to protect system commands. For more information about system command security, refer to the *IBM Tivoli NetView for z/OS Security Reference*.

Automating MVS Commands

You can automate MVS and subsystem commands entered from any MVS console or console interface. To do this, you must install a load module as an MVS command exit, add a .CMD statement in one of the `MPFLSTxx` members, and issue a `SET MPF=`*xx* command to activate the exit. Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for more information.

Issuing MVS System Messages and Delete Operator Messages (DOMs)

You can use the NetView WTO and WTOR commands to issue MVS system messages and the NetView DOM command to issue MVS DOMs.

For more information about the DOM, WTO, and WTOR commands, refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

System Automation/390 Programs

You can speed up the automation process by incorporating System Automation for OS/390 into your design. The System Automation for OS/390 licensed program is a NetView-based application which runs on z/OS and MVS/ESA Version 5. It is designed to provide a single point of control for a full range of system management functions.

Examples of Using NetView Program Interfaces

You can use the NetView program to automate the management of any product that sends messages or MSUs to the NetView program and receives commands from the NetView program. For some of these products, you need to use an interconnecting product as an interface to the NetView program.

By using interconnecting products, you can manage non-SNA networks and devices. The majority of these non-SNA networks and devices use a NetView service point, such as the UNIX NetView Service Point program, as an interface to the NetView program.

This section describes a few examples of interconnecting products that can be used as interface programs.

NetView Program Service Points

The UNIX NetView Service Point licensed program enables you to add to the list of products managed by the NetView program. You can obtain or write service point application programs that enable management and automation of many non-SNA networks and devices. A service point application program for the UNIX NetView Service Point program can monitor a non-SNA network, report network-management data to the NetView program, and pass commands from the NetView program to devices in the non-SNA network. Therefore, you can use the service point application program to expand the scope of NetView automation. The UNIX NetView Service Point program runs under the UNIX operating system.

You need the UNIX NetView Service Point program as an interface for communication between the Tivoli NetView program and IBM Tivoli NetView for z/OS. However, the Tivoli NetView program can operate as a standalone program that provides network management services without communicating with the IBM Tivoli NetView for z/OS product.

For information about the UNIX NetViewProgram Service Point program, refer to the *AIX NetView Service Point Installation, Operation, and Programming Guide*.

Distributed Networks

You can automate the handling of events that occur in distributed networks by using the NetView program with the Event/Automation Service. The Event/Automation Service provides a gateway between the NetView program and the distributed networks for network events that originate in either environment. The Event/Automation Service communicates with the NetView program using the NetView subsystem PPI interface, and communicates with event servers using the TCP/IP protocol.

The Event/Automation Service can translate and forward either NetView alerts or messages into Event Integration Facility (EIF) events and can also translate and forward these events into NetView alerts. These alerts can then be used with automation to start automatic responses.

IP Networks Using SNMP

The Event/Automation Service can manage event data between the NetView program and SNMP agents and SNMP managers. NetView alerts can be converted into SNMP traps before being forwarded to an SNMP manager. Traps that arrive from an SNMP agent can be converted into SNA alerts which can then be forwarded to the NetView program hardware monitor. There, these alerts are filtered and routed to the NetView automation table.

For more information about SNMP traps, see “Event/Automation Service” on page 404.

Non-IBM Networks

The NetView program can manage other types of networks (for example, DECnet).

Automation-Related Functions and Services

This entire book describes automation primarily from the perspective of system and network console automation. The book also explains how you can use automation facilities to assist or replace operator action in responding to messages and MSUs and issuing commands on the consoles of system and network software.

Other functions and services closely related to automation are also available for systems and networks. For more information, investigate the following automation-related topics:

- “Managing Workload” on page 35
- “Managing Network Performance” on page 35
- “Managing Input/Output” on page 35
- “Managing Storage” on page 36
- “Management Reporting” on page 36

Managing Workload

Automating the management of production batch jobs offers advantages in availability, improved control, and reduced operator involvement.

IBM offers the Operations Planning and Control/Enterprise Systems Architecture (OPC/ESA) licensed program for workload management. The OPC/ESA program can plan, control, and automate your MVS batch production workload. This program plans and schedules your workload processing and monitors and controls the flow of work through your entire data-processing environment, both local and remote. It reduces the human intervention needed while letting you retain manual control of important processes and decisions.

Using the OPC/ESA program for workload management complements NetView automation. The OPC/ESA program does the job scheduling. If a failure in a scheduled job requires operator action, NetView program automation can supply that action.

For more information about the OPC/ESA program, refer to *Operations/Planning and Control/Enterprise Systems Architecture General Information*.

Managing Network Performance

You can use NetView Performance Monitor (NPM) to give your automation application programs an increased spectrum of performance data. The NPM program can also be valuable for centralized operations, because the program can help you monitor the speed with which your central system communicates with distributed systems. Operators using the NPM program on a central system can view data collected at other systems.

NPM communicates with the NetView program through the NetView program-to-program interface and several other interfaces. By issuing commands to NPM through the operating system, automation routines can request data about specific network resources. For example, you can request data about communication controllers, lines, logical units, and physical units in SNA, local area, and X.25 networks.

NPM can also send unsolicited data. For example, if performance for a critical network resource falls below a threshold you define, the NPM program can send an alert to the NetView program. You can use the alert to inform automation routines of the performance problem before the problem affects users. You also have the option of sending resolutions to the NetView program to inform your automation routines when a problem is resolved.

Managing Input/Output

Input/output (I/O) management involves controlling the flow of data into and out of a data processing complex. In an automated environment, you might want to change your approach to I/O activities that previously required manual intervention, such as tape and printer management.

One approach to tape management is to avoid it by converting from tapes to direct access storage devices (DASD). DASD does not require the manual intervention that tapes do. You might want to compare, on a case-by-case basis, the cost of DASD to the cost of tapes plus the cost of people to handle the tapes. Consider the higher reliability and manageability of DASD. Also, you can institute periodic

reviews of your I/O rules and policies. Determine how effective your policies are and how consistently your application programmers are applying them.

For less frequently used data, you might find that it is still appropriate to rely on tape devices. The cartridge of the IBM 3480 Magnetic Tape Subsystem is extensively used for its size and reliability advantages over other tape devices. The Automatic Cartridge Loader function is available to assist with cartridge handling and scratch tape mounts with minimal human intervention and minimal delay.

Sometimes, the best approach to printer handling is to place responsibility in the hands of the users. You might be able to reduce the volume of printing. Programs such as IBM's Report Management and Distribution System (RMDS) program can help. The RMDS program enables you to present report data to users online from a central library archive. Users can view data online, printing only the portion of the information that they need to have on paper. The RMDS program eliminates the need for printing large volumes of report data on a regular basis and distributing them to users who often want only a fraction of a report.

For more information about the RMDS program, refer to *Report Management and Distribution System: General Information*.

Managing Storage

Storage management involves maintaining the integrity and availability of data that you keep on auxiliary storage devices such as tapes or DASD. Previously, users had to be aware of the characteristics of each device within the pool of storage devices on which their data sets can reside.

With the introduction of the Storage Management Subsystem (SMS) using the MVS Data Facility Storage Management Subsystem (DFSMS) family of products, storage administrators rather than users can manage DASD storage. Storage administrators establish policy statements in the form of storage classes and management classes, defining and managing the way storage is allocated on the basis of these classes. The user, allocating storage in terms of these policy statements, no longer needs to use device and configuration specifics such as UNIT and VOLSER.

Use of SMS decreases the number of program abends caused by out-of-space conditions that plague production job streams, because jobs need not be sensitive to configuration details. You can use storage management with workload-management products, such as the OPC/ESA program, that offer automated job recovery facilities. The result is production streams that run consistently and finish within their scheduled windows with minimal human intervention.

For more information about the DFSMS family of products, refer to the *MVS Storage Management library.*

Management Reporting

As you move toward an automated environment, include a strong management-reporting system in your automation design. As automation handles more and more of your operations, you might need to identify things that need management attention or that necessitate resource changes. To capture information from logs and summarize it for presentation to management, you can use the Information/Management and Service Level Reporter (SLR) products.

For information about Information/Family and SLR products, refer to *Introducing the Information/Family for MVS* and *Service Level Reporter General Information*.

Part 2. Achieving an Automated Environment

Chapter 3. Defining an Automation Project

This chapter describes the project definition tasks and phase of an automation project. In this phase, you assemble a planning team, investigate how automation can improve your operations, and set goals and objectives for the project.

Project Definition Tasks

The project definition phase focuses on:

Automation often works best as an integrated, company-wide effort that coordinates many separate departments and groups. Automation can change organizational and working relationships in the following ways:

- Operation organizations might be restructured.
- Operator roles might change.
- Working relationships among operators, technical support personnel, and system programmers might change.

Because automation can require considerable coordination or produce widespread changes, it is important to have the commitment of the whole organization, including upper-level management. Management must provide the resources necessary to achieve your automation goals.

An integrated approach helps to avoid duplication of effort. A fragmented approach, with each group or location choosing small and unrelated projects, can lead to wasted time, inappropriate approaches, or automation applications that cannot work together.

During the implementation phase, you can create your automation a small piece at a time. This is also an excellent time to look at the automation process as a whole. By developing an enterprise-wide approach from the start, you avoid the risk of having to redesign the project later.

At the beginning of the project, it is important to identify your goals, such as the following examples.

- How can automation best support your business objectives?
- How can it improve your data-processing operations?

While identifying the benefits of automation, you can also estimate the costs. By doing so, you can determine the types of automation that provide the greatest return for your investment.

Assembling an Automation Team

The first step is to assemble a team or teams to analyze and implement your automation. You might already know who in your organization is doing the automation. If not, ask these questions:

- Where in the organization does the responsibility for automation planning fall?
- What skills does the automation team need, and who can provide those skills?

Choosing an Approach

You can use any of several approaches, depending on the resources available and the schedule you require. One approach is to assign a project leader who works on the project full-time, calling on the support of other organizations as needed. Another approach, if more resources are available, is to form a temporary project team. In this case, several people work on automation full-time.

A third and more lasting approach is to create a permanent automation department. Also, consider whether you need separate teams for different stages or phases of the project. Many organizations start with a temporary planning team but establish a permanent department as their automation develops.

It is a good idea for an automation planning team to include people from all the organizations affected by automation. You might include:

- One or more operators
- A member of the technical-support staff for system management
- Another member for network management (if applicable)
- System programmers who support your major subsystems and applications
- Network user representatives

You can also include your Tivoli branch system engineer on the planning team. The branch system engineer can provide information about automation products or about the experiences of other customers who have successfully planned and implemented automation.

Involving Operation Groups

To achieve success, involve your operation groups in every phase of automation, from project definition through design, implementation, testing, and production. Members of the operation groups understand today's environments and can identify procedures that are appropriate to automate. They also are the ones who have to live with the results of automation. Involving them in the design of each automated procedure helps to ensure that the procedure matches their needs.

For example, both system and network operators in an unautomated environment usually rely on a constant flow of messages to know that things are running smoothly and that expected events are happening as anticipated. If you automate a specific procedure (system initialization) and suddenly no messages are displayed, the operators might have difficulty assessing whether things are going as anticipated. Involving the operation groups helps ensure that operator interfaces are adequate.

Creating a Project Plan

To manage your automation project, use a project plan that lists the steps you need to take in every phase, identifies the person or group responsible for each step, and assigns a target date for completion of the step. The project plan becomes a vehicle for managing the project and keeping track of your success in meeting the schedule.

The project plan can evolve over time. If you are not yet able to fill in complete details, you can, nevertheless, start a plan by setting down the tasks, responsible parties, and target dates that you already anticipate. You can fill in the details as the project evolves.

See Appendix B, "Sample Project Plan," on page 513 for a plan that identifies representative tasks for all phases of a project: definition, design, implementation, and production. This plan, of course, is just an example; the plan for your project might look substantially different.

Identifying the Goals of Your Organization

Another task of the planning team is to identify automation goals. Clear goals enable you to focus your project and measure your results. They can also help you to complete planning documents such as business proposals or the automation project plan.

Identifying Business Goals

Your corporation or organization probably has several business goals. They might be something like the following goals:

- Increase total business volume over the next 2 years by 40 percent
- Increase net profit for each of the next 5 years by 10 percent
- Increase profit margin by 5 percent next year by containing costs and increasing productivity

Different areas of the organization might have different business goals that you need to consider. By clearly understanding the goals of your organization, you can decide how automation can contribute to their achievement.

Identifying Data-Processing Requirements

To support overall business goals, data-processing departments typically have requirements of their own. The requirements might be objectives like these:

- Increase system availability by 10 percent over the next 2 years.
- Accommodate 12 percent growth capacity (in millions of instructions per second, or MIPS) and network resources over the next 2 years with no increase in operation staff.
- Improve system performance by 15 percent each year for the next 5 years.

Data-processing requirements typically fall into two classes: system-oriented and user-oriented. *System-oriented* requirements measure the amount of information that your systems process. These requirements include:

- Expected batch throughputs
- Workloads on each system
- Interactive transaction rates
- The number of concurrent users that you can support

By contrast, *user-oriented requirements* measure the impact of data-processing services on the user. Examples are expected response times for interactive work and expected turnaround times for batch work.

Service-level agreements reflect these expectations of performance. A service-level agreement resembles a contract between the data-processing department and that department's users. A service-level agreement might specify the services you provide, the hours you provide them, and various agreed measures of availability and performance. Whereas other requirements often represent goals that you want to accomplish, service-level agreements state minimums that you must accomplish.

If your organization does not use service-level agreements, or if your service-level agreements do not accurately reflect your goals, consider establishing agreements that are based on your goals. Service-level agreements can help you measure the improvements in service to users that automation provides. They can also help you identify problem areas of your operation that might benefit from automation.

Understanding Your Operating Environment

Questions you can ask might be:

- How do your operators spend their time?
- What routine and repetitive tasks can you automate to increase productivity?
- What unscheduled events require operator action?
- For each unscheduled event requiring operator action, how severe are the results of delayed or incorrect action?
- What events of any kind have a significant impact on your operations?

In summary, what are the most important problems and challenges in your operating environment today, and where can you gain the greatest return from automation?

After investigating your present environment, you can consider the future:

- What changes do you expect in your environment in the next year or the next several years?
- Do you plan to add hardware to your systems or network?
- Do you plan to add new applications that you must manage?
- Will the number of users relying on you for service increase?
- Will you be under pressure to accommodate growth without increasing your operations staff?
- Do you plan to add data centers?

Factors such as these contribute to your automation strategy and goals. With a good understanding of where you are and where you are going, you can devise a comprehensive strategy that makes full use of automation.

Start the process of identifying operating requirements for automation by working with the operations staff. Operators can identify the procedures they perform regularly, those they perform on a scheduled basis, and those that involve predictable responses or repetitive tasks. With this information, you can choose and prioritize the procedures you automate.

The NetView program's Message Revision table can help you analyze your system message traffic. By including UPON statements for MSID, PREFIX, or JOBNAME

(with or without any other statements), your revision report shows the numbers of messages matching each condition. See "Message Revision Table" on page 25 for more on the MRT.

MVS System and Network Logs

Analyze your MVS system and network logs for information about the number of messages that operators view each day. This information can help you assess the benefits of message suppression. On most MVS systems, message suppression yields impressive results.

You can write simple application programs to help you process logs. For an example of an application program used to process logs, see Appendix I, "The Sample Set for Automation," on page 577. The example program analyzes SYSLOG (the JES2 log) or DLOG (the JES3 log). For other logs, you can modify the example program or write one of your own.

The example analysis program illustrates several things your program can do:

- Record each unique message ID received and the number of times messages with that ID occurred.
- Provide a list of unique message IDs received, sorted by frequency of occurrence.
- Accept input that specifies such things as time limits for the analysis and any messages that can be ignored.

The list of sorted message IDs indicates where you might concentrate your efforts. For each message ID, compute the percentage that it contributes to total message volume. Usually, a small number of message IDs account for most of the message traffic. As few as 10 message IDs can cause 90 percent of the traffic. Therefore, you need to suppress or automate only a few message IDs to produce significant savings.

The logs also show the commands that operators have entered and help you to identify operating problems. For example, suppose you find that operators are entering many JES commands that start and stop job queues, alter job classes, and reset job priorities. The indication is that operators are spending a lot of time controlling the flow of work. You might, therefore, introduce a job scheduling program such as OPC/ESA. Or, perhaps a large proportion of the commands issued are responses to a frequently recurring situation such as the loss of a CICS terminal. By noting the frequency with which different commands are issued, you can identify the procedures that offer the greatest return on your automation effort.

Operation Procedure Books

Operation procedure books, or run books, are good sources of information for automation. When you identify your requirements and decide which procedures to automate, you can turn to the operator procedure books for a step-by-step guide to how automation can perform those procedures.

Problem-Management Reports

Problem-management reports track hardware and software problems and outline the actions taken to solve each problem. They can help you identify frequently recurring problems that are consuming resources, and they can help you identify procedures for responding to those problems.

Look for outages that were prolonged, either because the problem was not detected immediately or because the resources necessary to correct the problem were not available. Decide whether an automated process could have detected the problem and notified the correct people more quickly, or solved the problem.

Help-Desk Logs

Help-desk logs are another source of problem descriptions. Like problem-management logs, help-desk logs help you identify recurring situations and situations for which established procedures are inadequate.

Service-Level Agreements

By reviewing service-level agreements and measurements taken to confirm compliance, you can identify areas where you are having difficulty meeting commitments to users. These areas clearly represent problems. Automation might be a way to solve them.

Users

Users can inform you of possible problems in the environment, including problems that they have not reported to operators or that are not tracked by problem management.

Other Data-Processing Plans

Examine the changes you anticipate in your environment over your planning period. Start with documentation of your current system and network configurations, both hardware and software, and then examine your plans for the future. Document configurations for all data centers that you plan to automate, including those you plan to operate from a central focal-point system. Your automation plan must reflect anticipated changes.

For example, if your organization is adding a large new system, message suppression and console consolidation might be major requirements. If you are adding data centers or moving toward distributed networks, network and multiple-system automation might be major requirements. If you are supporting a growing number of users, adding hardware and software to your systems and networks without adding operators might be your primary requirement.

Interpreting the Information

After you review these sources of information, you should know:

- How operators spend their time
- The benefits of message suppression
- Which procedures you want to standardize and document
- Which procedures offer the greatest return for your automation effort
- What problems your users are experiencing

All of the information you gather contributes to defining your requirements for automation.

Developing Goals and Objectives for Automation

By developing goals and measurable objectives for your automation project, you can determine the project's contributions to your business and data-processing requirements and improve your operating environment.

Developing Goals for Automation

Developing goals is an essential part of the planning process. With your knowledge of business and data-processing requirements and your list of operating problem areas, you can develop appropriate long-term automation goals.

See “Benefits of Automation” on page 3 for categories of possible automation benefits.

You might want to review these categories and decide which are the most important to your organization. You can also choose goals of your own that reflect your own needs and environment. Choosing three or four of the most important benefits you expect from automation and making them your long-term goals provides a focus for the automation project.

Developing Measurable Objectives

Use measurable objectives to determine the progress you are making toward your automation goals. Identify one or more specific measurements or indicators for each long-term goal.

Measurements and projections play an important role in assessing the costs and benefits of the automation project. If greater system availability is one of your goals, you should know your current availability levels and the levels you expect to attain. You can evaluate whether certain portions of the project require more resources, whether others should be discontinued or expanded in scope, and the extent to which automation is achieving your goals.

Table 1 on page 48 shows a worksheet with examples of major measurements. The worksheet, which covers a 5-year period, uses goals derived from the automation benefits in “Benefits of Automation” on page 3. You must decide on major measurements that reflect your automation goals and suit your situation.

For information about calculating benefits for the measurements listed in Table 1 on page 48, see “Quantifying Costs and Benefits.” For additional examples of indicators that you can use to measure progress toward a number of goals, see Appendix C, “Sample Progress Measurements,” on page 521.

Quantifying Costs and Benefits

After identifying indicators you can use, their current measurements, and the measurements you expect after automation, you can compute monetary values. Calculating monetary values gives you further information about the types of automation that can yield the greatest benefit. Calculating monetary values can also help you determine the level of resources you must allocate to each form of automated operations.

The projected costs for an automation project derive from assessment of the human and system resources that implementation requires. The projected benefits derive from the measurements and projections you have established for each of your automation goals.

If you have created a project plan, the plan shows many of the steps you expect to take to plan, design, and implement automation. See Appendix B, “Sample Project Plan,” on page 513 for a sample automation plan.

You can use your plan as a basis for determining the resources that each step requires. Identify the human and system resources you need for each of the remaining phases of the automation effort.

Next, calculate benefits. Using the measurements and projections you developed for your automation goals, you can quantify the savings achieved by moving from manual to automated operations. The savings represent the financial benefits of automating. Table 1 shows an example of a benefits worksheet.

For example, if one of your goals is to avoid adding operators as your network expands, your measurable objectives must specify how many operators you expect to add if you continue manual operations. Similarly, you must project how many fewer operators you need to add if you implement automation and simplify operator tasks. Calculate the money you can save to estimate the value of automation in this area.

Improved availability can be an important benefit. To calculate the value of CICS availability, for example, you might use the following steps:

1. Calculate the amount of yearly downtime per user for CICS without automation and subtract the projected amount of downtime per user with automation.
2. Multiply the difference in downtime by the total of each class of CICS user, such as operator or programmer.
3. Multiply the result by the chance (in percent) that each user will need CICS during downtime.
4. Multiply this result by the monetary value for the user's time.

Some measurements might overlap. For example, a measurement of the personnel savings per data center might overlap with a measurement of the personnel savings per application. If you have overlapping measurements, ensure that you do not include both of them in the total savings.

Table 1. Example of a Financial-Benefit Worksheet

Area	Without Automation	With Automation	Savings per Year	5-Year Total
System and Network Availability				
NetView program				
CICS program				
IMS program				
TSO program				
VTAM program				
Communication controllers				
NCP programs				
Growth-Constraint Removal				
Maximum capacity				
Operator Productivity				
Number of personnel				
Today				

Table 1. Example of a Financial-Benefit Worksheet (continued)

Area	Without Automation	With Automation	Savings per Year	5-Year Total
First Year				
Second year				
Third year				
Fourth year				
Fifth year				
Consistent Operations				
Operator-caused failures				
Operator turnover				
Totals				

Securing Commitment

Your investigation of requirements, goals, costs, and benefits can assist you in obtaining the commitment of management and of your whole organization for proceeding with the automation project.

It is important to obtain commitment and support from each department or group that automation affects. The affected groups might include system and network operations, system programming, technical support, users, and others. You need the cooperation of these groups to successfully design and implement automated operations. Therefore, ensure that each group understands your goals and the benefits that you expect.

Chapter 4. Designing an Automation Project

This chapter describes the design phase of an automation project. In this phase, identify specific procedures to automate and the work required to automate them. Define the scope of the project and the order in which procedures are to be automated. From this information, determine a structure for your automation. Lay the groundwork for implementation by establishing common practices and rules for all of your automation application programs.

After introducing the project design tasks, this chapter describes several guidelines that can direct your design efforts.

Project Design Tasks

After reviewing your preliminary planning decisions, you are ready to begin the design tasks.

Identify Procedures and Functions to Automate

You might have already identified many procedures and functions to automate during initial planning. Talking to operators, examining system and network logs, and returning to the information sources described in "Understanding Your Operating Environment" on page 44 can help you find additional candidates for automation. Good candidates for automation are:

- Procedures that consume operators' time
- Events that demand prompt and accurate responses
- Repetitive procedures that can be performed mechanically

Prioritize Procedures and Functions

After choosing the automation procedures and functions necessary to achieve your automation goals, you can create a schedule. The schedule can prioritize the procedures and functions, giving preference to the changes that offer the greatest return for the least effort. The schedule reflects the speed with which you expect your organization to implement and assimilate automation.

Schedule Stages for Implementation

When you schedule the implementation of the automation procedures and functions, consider dividing the project into stages. By doing so, you give yourself sufficient time to test, tune, and absorb each change in the environment. See "Stages of Automation" on page 7 for an example of a sequence of stages.

You can devise a sequence to reflect your goals and objectives.

Establish Standards

Besides creating schedules, the design team can establish standards and choose general automation techniques. For example, you might decide on any of the following:

- An approach to security issues for all routines
- A format for writing messages to the logs for all routines
- A common way of notifying operators when there are problems

- A common protocol for using global variables to share information between routines

By deciding these things in advance, you ensure a unified automation approach that makes maintenance easier and enhances accurate communication among all parts of the automation application. “Design Guidelines” describes many of the issues that you must consider. These include not only programming issues but also the impact that operational changes might have on your environment and your organization.

Design Guidelines

Consider the following principles, suggestions, and guidelines when creating your design:

- Design for easy expansion and propagation - see “Designing for Expansion and Propagation”
- Design for audibility - see “Designing for Auditability” on page 53
- Design for security - see “Designing Automation Security” on page 53
- Design for availability - see “Designing for Auditability” on page 53
- Automate an event close to its source - see “Automating Close to the Source” on page 54
- Choose whether to use more than one NetView program per system - see “Using Multiple NetView Programs on a Single System” on page 54
- Provide effective operator interfaces - see “Providing Operator Interfaces” on page 55
- Educate your staff for automation - see “Educating Your Staff” on page 56
- Anticipate changing staff roles - see “Anticipating Changing Staff Roles” on page 56
- Provide for automation testing - see “Providing for Testing” on page 56
- Provide for problem and change management - see “Designing for Auditability” on page 53t
- Choose reliable focal points - see “Choosing Focal Points” on page 57
- Consider using backup focal points - see “Using a Backup Focal Point” on page 58
- Define operator sphere-of-control - see “Defining Operator Sphere-of-Control” on page 59

Designing for Expansion and Propagation

To save time and effort when adding new software or new equipment to your automated environment, design your automation for expansion. If you plan to automate more than one system, ensure that the routines you write for one system can be easily copied onto other systems.

One way to design for expansion and propagation is to use global variables for system and resource names and other important information rather than hard-coding them into command procedures. You can then use a single set of automation routines and adapt them to new equipment, new software, or new systems by redefining your global variables.

In a parallel sysplex environment, a copy of NetView might be running on multiple MVS images in that environment. Because of this, data set names, partitioned data set member names, and the contents of these members might need to be unique for each MVS image. Cloning support decreases the amount of maintenance required by permitting this type of data to be shared across a parallel sysplex while retaining the uniqueness of each MVS image. It might no longer be

necessary to maintain separate NetView partitioned data sets with unique member data. To take full advantage of this function, an MVS system of Version 5 Release 2 or later is required.

See the automation samples documented in Appendix I, "The Sample Set for Automation," on page 577 for an example of using global variables. You can also store variable information in a control file rather than using global variables.

Another way to design for expansion and propagation is to use the Resource Object Data Manager (RODM) for retaining system and resource names and other important information, rather than hard-coding the information in command procedures. See "Resource Object Data Manager" on page 25 and Chapter 8, "Automation with the Resource Object Data Manager," on page 77 for more information about using RODM.

If you plan to propagate automation to more than one operating system, consider writing your command procedures in a language that runs on all of them. See "Choosing a Language" on page 22 for information about which languages run on which operating systems.

Designing for Auditability

In any automated operation, a good audit trail is vital. The NetView program records system messages in the Canzlog log. The Canzlog log contains a rich set of recorded attributes for each message and a filtering scheme that shows only the messages of interest. Optionally, the NetView program can record messages in the network log and the system log or both the network log and the system log. You can also create sequential logs and sort information into different logs based on any criteria that you choose. For example, you can log messages coming from different subsystems in separate files, establish separate logs for each NetView operator and autotask, or set up a separate log to track automation-related activity.

Several basic principles apply to designing for a good audit trail:

- Log each action that automation initiates.
- Flag each automation event that you log so it can be identified as an automation event. For example, you might precede all automation-related log entries with a greater-than symbol (>).
- Log as much relevant information as possible. For example, you might log the reason that you issue an automation procedure and the names of global variables that you update as a result.
- Log each occasion when automation solves a problem.
- Log each occasion when automation fails to solve a problem. You can use this information to upgrade and improve your automation.

Designing Automation Security

Design your automated environment so that only authorized operators have access to system and network control facilities. You can control operator access through passwords, restricting data set access, using command authorization, with span of control, and other techniques. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for more information.

For instance, you can protect which commands can be issued by operator or automation tasks using either the NetView command authorization table or a system authorization facility (SAF) product such as RACF (Resource Access Control Facility).

Ensure that a command procedure issued from the automation table validates the source of a message or MSU before responding with any potentially disruptive commands. For example, you can ensure that a message came from the expected system, job, or operating-system component. Because people can enter both system and network commands from NetView consoles in an automated environment, it becomes especially important to control access to NetView consoles.

Designing for Availability

Because you are entrusting your system or network to automation, you need to ensure that the automation application is continuously functioning and available. Think of ways to reduce the number of planned outages and to recover from unplanned outages. The approach you take in designing for availability might vary, depending on whether you are automating a single system or a multiple-system network.

Among the approaches take to provide for automated recovery in a single system is to run two NetView programs on the system, as described in "Using Multiple NetView Programs on a Single System." The advantage arises because the two NetView programs can then monitor each other. If one fails, the other can restart it.

In a multiple-system network, have NetView programs on separate systems monitor each other and initiate recovery when necessary. However, this approach depends on having reliable links between your systems.

If you are using a focal-point system to automate several distributed systems, establish a backup for the focal-point system. This ensures that distributed systems can continue to forward information that requires external automation or operator action, even if the primary focal point becomes unavailable.

In any case, your automation applications within NetView can monitor each other, ensuring that autotasks and the automation table function continuously. For example, one autotask can monitor another by sending it messages and checking for timely responses. You can use the EVERY command to perform this sort of query on a regular basis.

Automating Close to the Source

A guiding rule for automation is to automate an event as close to its source as possible. If you intend to operate several distributed systems from a focal-point system, automate everything you can on the distributed systems themselves. Forward to the focal point only those problems that the distributed systems cannot handle. Automating close to the source maximizes both performance and reliability.

Similarly, you can suppress unwanted system messages with an MRT action in the address space where it originates. This saves considerable processing both in the system and in NetView.

Using Multiple NetView Programs on a Single System

A single NetView program on a system can accomplish all NetView functions, including network management, network automation, and system automation. However, some organizations choose to divide these functions among two or more NetView programs. For example, one NetView program on each system might perform network-management operations, such as network problem determination, and another might perform automation. Or, one NetView program might perform all network functions and another might perform all system functions.

A NetView program for automation can run at a dispatching priority higher than the tasks that it automates. A NetView program for network management can be set to a lower priority so it does not interfere with automation and other tasks.

Installing more than one NetView program on a system can help groups within your organization independently use the NetView functions they need. For example, a system-operation group and a network-management group can have separate NetView programs.

However, there are drawbacks to running more than one NetView program per system, including increased complexity. Running two NetView programs means maintaining two copies of libraries and logs. You must be careful to avoid endless loops, in which two NetView programs continually send messages back and forth to each other. Also, storage requirements are greater with two programs.

If you choose to run NetView Message Revision table from two Net Views in the same system, be sure to examine the order of SSI invocations (this can be done by the command `D SSI`). The later MRT override settings made in an earlier MRT.

For more information about running multiple NetView programs, see Chapter 32, "Running Multiple NetView Programs Per System," on page 455.

Providing Operator Interfaces

The operator interface is critical to the design of your automation scheme. Ensure that operators are receiving the information they require to operate the system or network, to influence the operation of the automation application program, and to monitor the automation.

See "Improving Operator Interfaces" on page 12 for options offered by NetView for monitoring in the automated environment. For example, depending on which NetView feature is installed, operators can monitor information provided by:

- The command facility
- The hardware monitor
- The status monitor
- NetView management console
- The Automated Operations Network (AON)
- Full-screen panels and help panels
- The Canzlog log

In most environments, operators are accustomed to using messages to judge how well the system is working. Therefore, operators need to be involved in deciding which messages must be suppressed and what automated actions must be taken. Ensure that you still provide operators with the information they need to verify that the system is functioning correctly.

As you begin automation, you can inform operators of everything, from events that require action to the issuing of automated command procedures. Eventually, as automation becomes the standard mode of operation and the operation staff becomes comfortable with automation, you can curtail notification and inform operators only when their action is required. However, continue to log messages that indicate when automation activity occurs in the system or network. These messages can assist in problem determination if automation fails.

Educating Your Staff

The people who design and implement your automation application programs need to be adequately trained before they begin. They need to understand both the requirements of your organization and the products you are using for automation. If you divide the duties, you might need different training for different groups. For example, one group might create automation procedures and another might create automation displays.

Furthermore, operators need to be informed of the changes in their operating environment at every stage of automation. They must understand the new operator interfaces and the changes to their responsibilities. Education is an ongoing requirement for ensuring the success of automation.

You can continue to train operators to run systems manually, ensuring that they can resume responsibility for operations if automation fails. However, it is usually more efficient to train your people to resume automation. Rather than expend the effort teaching manual operating techniques, you can test your automation and implement backup and recovery plans to avoid failure. Document your automation so that operators and programmers can use the documentation to perform procedures manually if necessary.

Anticipating Changing Staff Roles

Automation can change the roles and interactions of data-processing staff members. Ensure that you consider these changes and how automation affects your employees and your organization.

For example, if you are combining system and network automation, you can also combine system and network operation staffs. Because you are using a common design for system and network automation, the people who are to resolve problems that automation cannot handle need to understand both system and network resources.

Another example of a change in roles is a possible change in operator career paths. As automation takes over system and network monitoring and routine, repetitive tasks, operators might spend a greater proportion of their time making decisions, solving unique problems, and working with the automation application itself. One way to accommodate these changes is to create a new job category for operators, such as automation specialist. The specialist must understand system and network operations, as well as the automation applications used to run them. Operators who create or help with automation procedures can gain automation skills and learn to operate the environments of the future.

Providing for Testing

As with any new product or application, plan to test your automated procedures before placing them in the production environment. Each stage of your implementation requires thorough testing. In addition, you can do regression testing of your automation applications when your system or network changes, ensuring that your routines work with new releases of operating systems and application programs, and with new hardware.

Providing for Problem and Change Management

Problem management is an important part of automated operations. By logging problem records before automation takes any recovery action, you can minimize the risk of losing your record of system and network problems that require attention.

Implementing automation also affects change management. With automation, it is helpful to track all changes to the operating environment, possibly in more detail than you have before. Even slight changes to a message format, for example, can affect your operations if the message is triggering automation. Keep a list of messages, alerts, and other data records that are triggering automation. For each message, record whether you use just the message ID or use other parts of the message as well. When you learn of changes to a message or alert, compare them to the list to see whether you need to update your automation.

Choosing Focal Points

In a multiple-system environment, you can perform many automation tasks with single-system automation running independently on each system. For tasks that you do not automate locally, you can forward the associated data to a designated focal-point system. Then you can automate responses to the data with automation on the focal-point system, or you can display the data for operators.

Before choosing a focal point, consider the kinds of tasks that you want the focal point to perform. The way you intend to use the focal point influences your choice of a focal-point system. The following are some considerations that can affect your decision:

- The focal point can perform automation activities that require coordination among two or more systems.
- The focal point can monitor your automation facilities in other systems and recover those facilities if they fail.
- The focal point can monitor the hardware and software of other systems and recover the hardware or software if it fails.
- Operators at the focal point can respond to exception conditions that automation cannot handle.

Choose a stable and reliable system for a focal point. In general, avoid choosing a system that is already heavily used. Also, avoid a system that you use for developing application programs, installing and testing new products, or other testing.

The focal-point system must have an information management product installed, enabling it to log problems that occur in other systems. You might also need system-management application programs, such as programs for problem management, change management, and reporting.

If you have a communication management configuration (CMC) system, the CMC system might have the highest availability of your systems. Therefore, you might want to use your CMC system as the focal point. Examine the capacity of the CMC system to ensure that the system can handle the combined processing load of CMC and automation duties.

Figure 7 on page 58 shows a focal-point system that manages distributed systems. As shown, the distributed systems can be at more than one site.

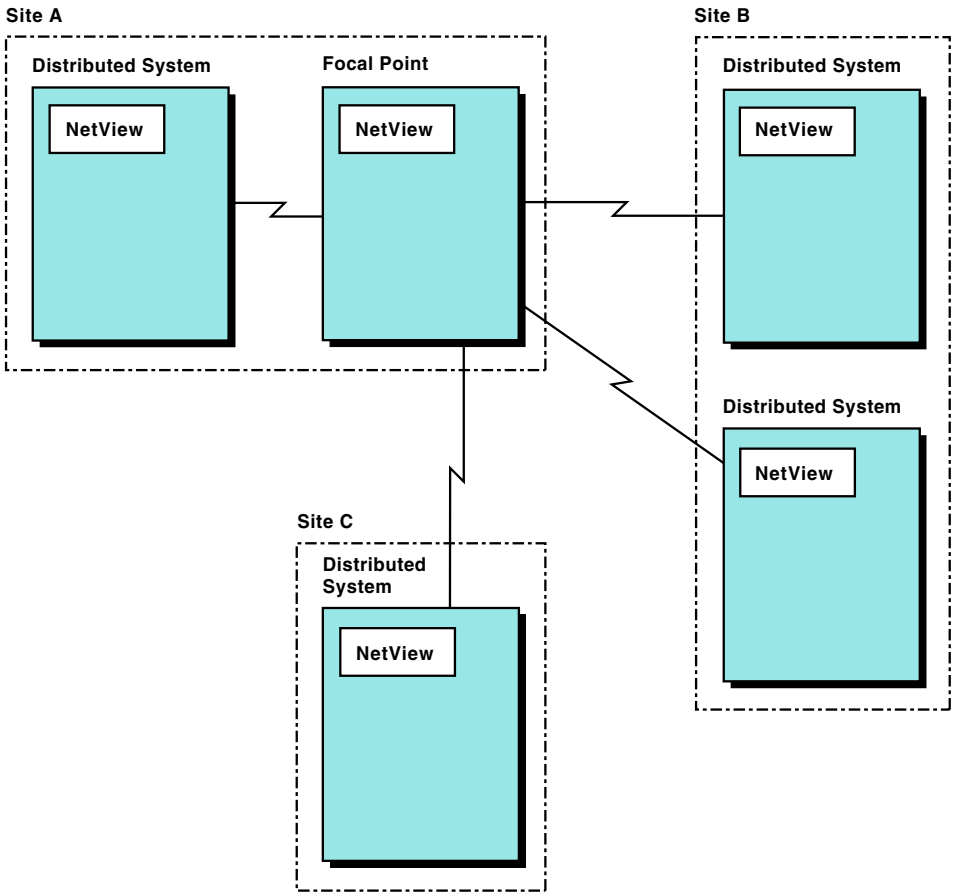


Figure 7. Example of a Multisite Configuration

You can forward many types of data from a distributed system to a focal point, including messages, alerts, status information, and user-defined classes of information for the LU 6.2 transports.

For an overview of the forwarding capabilities of the NetView program for each type of data, refer to Chapter 26, "Centralized Operations," on page 373. Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for information about how to set up message, alert, and status forwarding. See the *IBM Tivoli NetView for z/OS Application Programmer's Guide* for more information.

You can have a single focal point or several. However, if you have more than one focal point, each distributed system usually sends all types of data to a single focal point. That is, any alerts, messages, status information, and operations management information forwarded from a given system can all go to the same focal point. With this type of design, operators and automation at the focal point can monitor all types of data from one location.

Using a Backup Focal Point

You can define as many as eight backup focal points. If you intend to have a focal-point system manage many other systems, you can use a backup to ensure that a focal-point failure does not disrupt your automation. The backup focal point can be one of your distributed systems or a dedicated backup system. This system must be available to take over for the focal point if any outages occur.

For many types of data, you can establish NetView-to-NetView sessions between the backup focal point and the distributed systems automatically if you lose communication with the primary focal point. You can do this without operator intervention. Only status forwarding does not support a backup focal point.

Other advantages and considerations for a backup focal point include:

- You can have primary and backup focal points monitor each other. A loss of communication can trigger recovery actions.
- The VTAM program and NCP can recover links in the network if link failures occur.
- You can establish multiple NetView-to-NetView sessions between the primary focal point and a distributed system. Ensure that the route used by each session is different.

Defining Operator Sphere-of-Control

Sphere-of-control enables an operator at a focal point to manage the relationships between that focal point and entry points (distributed nodes). Each entry point is categorized by type and state, which can be displayed by the focal point operator using the FOCALPT DISPSOC command.

In Figure 7 on page 58, the focal point is at Site A, and manages a sphere-of-control encompassing four distributed NetView systems. One entry point is at Site A, two are at Site B, and one is at Site C.

An operator at the focal point can manage a sphere-of-control through the sphere-of-control manager (SOC-MGR). The MS-CAPS application within the focal point or entry points is responsible for establishing and recovering the sphere-of-control relationship, and for providing status. The focal point operator can add and delete entry points and add information to the sphere-of-control configuration file. This file can be used during NetView initialization to set up sphere-of-control environments.

For information, see Chapter 26, “Centralized Operations,” on page 373.

Chapter 5. Implementing an Automation Project

This chapter describes the tasks involved in the implementation and production phases of an automation project.

If you envision an extensive automation project, divide it into stages as described in Chapter 4, "Designing an Automation Project," on page 51. You then have an implementation phase and a production phase for each stage of automation. Repeat the tasks in this chapter for each stage.

Implementation Tasks

In the design phase, you laid out a schedule for implementing various functions and procedures. Examine those functions one by one in the chosen order. For each function to be automated, use the following approach:

1. Analyze your manual method of operation. Often, you can best automate a function by having NetView facilities closely follow the sequence of steps that an operator usually takes. In any case, you must understand the manual method before devising an automated method.
2. Determine the best approach to automating the function.
3. In your development environment, install the products you plan to use for this function.
4. Develop application programs and command procedures that you plan to use for this function.
5. Install the application programs and command procedures in a test environment.
6. Test and debug these application programs and command procedures.
7. Measure the performance of the application programs and command procedures. Tailor and tune them for efficiency.

When you have thoroughly tested and tuned all automation products, functions, applications, and procedures, you are ready to go to the production phase.

Production Tasks

The production phase must begin with educating your operators on the changes you are about to make.

When you have educated your operators, begin installing the products, if any, that you are adding to the production systems to support automation. Test these products to ensure that they are running correctly on the production systems.

Next, install the automation functions and procedures that you have developed. Make necessary changes to adapt these functions to the production systems. If your design is for easy propagation, as described in Chapter 4, "Designing an Automation Project," on page 51, most of the necessary changes require only that you alter some global variables or data in a control file. Test your automation functions and make any necessary corrections or enhancements.

If you have divided your project into stages, go to the next stage in your sequence. See "Stages of Automation" on page 7 for a description of automation stages.

Continually re-examine and review the automation that you have put in place. Measure the results that you are achieving and compare them to the expected values you identified in the project-definition phase. For information about how measurements are used to track the results of automation, see “Developing Measurable Objectives” on page 47.

Look for ways to improve your automation. Perhaps there is another message that you can suppress or another MSU that can receive an automatic response. By aggressively tuning and enhancing your functions and procedures, you can realize the maximum benefit from automation.

Use the AUTOCNT command to generate automation table usage reports for your system. You can use the reports to analyze automation table statements to see how frequently they are matched. You can move frequently matched statements toward the top of the table so that less checking of unmatched criteria takes place. You can also determine whether to delete unmatched statements from the table or to delete statements changed because of logic errors.

Automation table usage reports also enable you to determine the level of automation taking place on your system. These statistics can be useful in reports for management purposes. For information about the AUTOCNT command and automation table usage reports, see Chapter 15, “The Automation Table,” on page 147.

Part 3. Planning for Automation in Selected Environments

Chapter 6. Automation Using MVS Extended Multiple Console Support Consoles

This chapter describes in more detail the information about extended multiple console support (EMCS) consoles that was given in "Automation on MVS Systems" on page 27. This chapter describes:

- Some of the advantages, implications, and planning considerations for using EMCS in NetView automation
- Some advantages for using EMCS instead of the MVS subsystem interface

Using EMCS Consoles with NetView

EMCS consoles enable an MVS application program to interact with the MVS system as if the application program were an operator at a terminal. Using extended multiple console support consoles, NetView automation can interact with the MVS system as if the NetView operator were an MVS operator.

Using extended multiple console support consoles enables NetView automation to interact with the MVS system without some of the restrictions imposed in other versions of the MVS system. For example, extended multiple console support consoles do not need to be defined in the CONSOL*xx* member of the PARMLIB data set.

NetView processes unsolicited MVS messages using the subsystem interface and processes solicited command responses using extended multiple console support consoles. This allows you to extract unsolicited messages earlier in the MVS process, while allowing operators the flexibility of EMCS.

For information about extended multiple console support consoles, refer to the MVS library. For more information about attributes for extended multiple console support consoles that NetView uses, refer to the *IBM Tivoli NetView for z/OS Security Reference*.

Advantages of Using EMCS Consoles with NetView

Some advantages for using EMCS consoles with NetView are:

- There is no defined limit on the number of MVS operator consoles that can be used.
- You can define MVS consoles dynamically for NetView operators. Note that operator commands and responses use EMCS consoles allocated for each operator.
- Information appearing on the NetView command facility screen can be made to look more like MVS operator consoles by using the SCRNFMT parameters of the DEFAULTS command.
- Consoles do not need to be defined in the CONSOL*xx* member of the PARMLIB data set.
- You have the option to have system messages delivered directly based on route codes. However, beware of duplicate automation when using this method, because more than one original message might be delivered to the NetView program.

- You can more easily define authority for your operators.

Usage Notes:

1. All cross-domain sessions must use the RMTCMD command to prevent loss of data. Otherwise, if the sessions are established between an operator station task (OST) and a NetView-NetView task (NNT), messages are sent without any appended message data block (MDB) data structures. Data structures contain special information about a message. Data structures also contain some deleted operator message (DOM) information associated with the message. Such information in the MDB data structures, therefore, is lost on the OST-NNT sessions.

   Sending a message without the MDB data structures provides compatibility for earlier levels of NetView that do not process the MDB information.
2. Change in the attributes for your extended multiple console support consoles might cause more than one console in NetView to solicit the same MVS system message. Such messages are considered solicited by the NetView program. Beware of duplicate automation.

Planning for Extended Multiple Console Support Consoles

This section describes points to consider as you plan for using extended multiple console support consoles in your NetView automation.

Enabling Extended Multiple Console Support Consoles

You can enable extended multiple console support consoles by specifying values in the MVSPARM statements in the CNMSTYLE member.

Developing Console Naming Conventions

Develop your naming conventions for consoles before you start to use extended multiple console support consoles.

Note: You can use the ConsMask keyword in your style specifications to simplify the task of choosing unique console names.

These are rules for developing console names:

- The length of each name must be 2 - 8 characters.
- The first character must be from the group of A–Z, @, #, and $.
- The remaining characters must be from the group of A–Z, 0–9, @, #, and $.

When using console naming conventions:

- Each name must be unique within a system and within all systems in a sysplex configuration.
- Console names that are defined in the CONSOL*xx* member of the PARMLIB data set are not available to be used as names of extended multiple console support consoles.
- Console names might be used by other application programs and must not be duplicated.

Acquiring Extended Multiple Console Support Consoles

You can acquire an EMCS console by using an MVS command or the NetView GETCONID command. If you issue an MVS command or the GETCONID without the CONSOLE keyword to acquire an EMCS console and your task has not already obtained a console, NetView determines the console name in the following order:

1. If a SETCONID command was used, the name specified by it is used.
2. If the ConsMask statement in the CNMSTUSR or C*xx*STGEN member that is included in the CNMSTYLE member is not defined as an asterisk (*), its value is used as a mask for determining the default console name. Refer to *IBM Tivoli NetView for z/OS Administration Reference* for more information.
3. If OPERSEC=SAFDEF was in effect when the operator logged on, NetView uses the value of CONSNAME specified in the NetView segment of the SAF product. If there is not a CONSNAME in the NetView segment, see Step 5.
4. If OPERSEC=SAFDEF was not in effect when the operator logged on, NetView uses the value of CONSNAME specified in the operator's profile in DSIPRF. If there is not a CONSNAME in the operator's profile, see Step 5.
5. If a CONSNAME was not specified in either the NetView segment or the operator's profile, NetView uses the operator task name as the console name. In this case, the operator ID must be greater than one character in length and abide by the same rules as for console names.

You can issue the GETCONID command to acquire an EMCS console with a name specified by the invoker as well as specifying other attributes for the console.

For information about the GETCONID and SETCONID commands, refer to the NetView online command help. For more information about attributes associated with extended multiple console support consoles, refer to the *IBM Tivoli NetView for z/OS Security Reference*.

Defining Consoles in Groups

If you want to use the RELCONID command SWITCH parameter to switch messages to an alternative console when your console is released, define your console to a group. For more information about console groups, refer to the MVS library.

Using the Message Revision Table (MRT) or the Message Processing Facility (MPF) Table to Direct Messages to NetView Automation

You can use the NetView Message Revision Table (MRT) or the message processing facility (MPF) to mark messages for automation. Information on directing a message to NetView automation can be found in "Subsystems in Message Processing" on page 524

You can also use the Message Revision table (MRT) to perform the functions provided by the message processing facility (MPF). Additional information about the Message Revision table can be found in "Message Revision Table" on page 25.

Using Attribute Values for Extended Multiple Console Support Consoles

Use specific attributes and their values for extended multiple console support consoles are provided as defaults. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for a chart of the full set of defaults.

Defaults for a Console Obtained by an Operator

For a console obtained for an operator, some defaults and their meanings are:

MSCOPE = *ALL
The console can receive messages from any member of a sysplex, and command responses can be received from all systems.

ROUTCODE = NONE
The console does not solicit system messages by route code.

Using Route Codes

If you decide to solicit messages for your extended multiple console support consoles by using route codes, be aware that you might create duplicate automation. When you set up an EMCS console to receive messages with a certain route code, a message with that route code is delivered to that console, as well as to any other console that solicited the message.

Some messages have more than one route code. When messages are solicited by route code, multiple instances of a message can be delivered to extended multiple console support consoles used by NetView. When setting console attributes, it is preferable to ensure that you do not solicit multiple instances of the same message. If you choose to solicit multiple instances of the same message, you can use the automation table to select which task is to process a message if two tasks receive the same message.

These examples illustrate cases in which duplicate message solicitation can cause NetView to produce duplicate automation.

Case 1

Consoles in use:

- The CON4 EMCS console is set up to receive messages with route code 4. This might have been set up with the MVS VARY command or the RACF OPERPARM segment.
- The CON6 EMCS console is set up to receive messages with route code 6.
- The CNMCSSIR task is receiving messages marked for automation, including any that are subject to a NETVONLY value that was set or REVISE('Y' AUTOMATE) revision table actions.

Event: The MVS system issues an IEE*xxxx* message with route code 4, and this message is marked for automation.

Result:
The CON4 console receives the message from the MVS system because the message is assigned route code 4. The CNMCSSIR task receives the message through the subsystem interface message interface system because the message is marked for Automation in the Revision AUTO(YES). Both tasks drive automation. Unless the automation table contains a statement to disregard one of the messages (for example, by operator ID), automation occurs twice because two identical messages are delivered to the NetView program.

Note: If the NETVONLY value is specified rather than REVISE('Y' AUTOMATE) in the NetView MRT, the duplicate automation is avoided.

Case 2

Consoles in use:
Same as for case 1.

Event: The MVS system issues message IEE*yyyy* with route codes 4 and 6, and this message is marked AUTO(NO) in the MPF table.

Result:
CON4 receives the message from the MVS system because the message is assigned route code 4. CON6 receives the message because it is assigned route code 6. Both tasks drive automation as in case 1.

Understanding Effects of Attributes

From the preceding examples, you must realize that the attributes set for extended multiple console support consoles affect the delivery of MVS system messages. For more information about the attributes of extended multiple console support consoles, see the *IBM Tivoli NetView for z/OS Security Reference*.

If you solicit messages by route code, be aware that some messages have no route codes. Therefore, a console defined to receive all messages with route codes does not receive all the messages in the system. For example, monitor type messages do not have route codes. Refer to the MVS library for a list of MVS system messages and their route codes.

Implementing Security Access

You can implement a security access facility product such as the Resource Access Control Facility (RACF) to provide security for NetView operator tasks and autotasks. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for information about using RACF to protect access to names of extended multiple console support consoles and about protecting system commands for operators and autotasks.

Avoiding Message Loss because of a Full MVS Message Data Space

Messages to be written to extended multiple console support consoles are temporarily stored in an MVS message data space until NetView retrieves them. If the maximum storage value set for the MVS message data space is exceeded during operation, message delivery is halted temporarily from the MVS system to the message data space for the extended multiple console support consoles that NetView uses. To avoid this problem, you can use the defaults that the NetView GETCONID command sets for the maximum data space for message transfer. This data space is managed by the MVS system and is used only as needed.

Note: These limits do not apply to unsolicited messages that are retrieved by the CNMCSSIR task for automation.

Avoiding Message Loss because of an Exceeded Queue Limit

Each EMCS console has an attribute called QLIMIT. This attribute defines the number of messages that can be queued at one time in the data space for this console. If the queue limit is reached, the MVS system temporarily stops delivering messages for this console, and these messages are lost.

Another attribute for each EMCS console is called ALERTPCT. You can use this attribute to help determine whether you are approaching the queue limit for a particular console. The ALERTPCT attribute defines the percentage of the queue limit that causes a warning message to be issued.

If the message queue limit for a console is reached, a task might not have enough time to process all the messages directed to it. Some tasks run at a lower priority than other tasks and do not get sufficient time for processing all messages.

Balancing MVS Message Storage and Message Queue Limit

You need to obtain a balance among the amount of storage reserved for the MVS message data space, the number of operators using extended multiple console support consoles, and the values defined for the message queue limits. Use the STORAGE parameter of the GETCONID command to reserve storage for the message data space. Use the QLIMIT parameter of the GETCONID command to define the queue limit.

Note: The QLIMIT and STORAGE attributes can also be set using the RACF OPERPARM segment.

The STORAGE parameter sets the maximum allowable size for the data space. The first active console in NetView sets the maximum storage value. Ensure that the first EMCS console to be activated sets the maximum storage value that you want. The queue limit value defined by the QLIMIT parameter for each console applies only to that console.

These messages are related to reaching the storage limit and queue limit for extended multiple console support consoles. Correct responses to these messages are especially important:

- DWO201I
- DWO202I
- DWO204I

For explanations of these messages, refer to the NetView online help.

Migrating from the Subsystem Interface to Extended Multiple Console Support Consoles

Migration from the subsystem interface to EMCS consoles must be done in stages. These are the suggested stages. For information about using the subsystem interface and EMCS consoles, see "NetView Program Interfaces" on page 81.

Establish Unique Names

Establish naming conventions for EMCS consoles before you start to use the consoles. For information about console naming conventions, see "Developing Console Naming Conventions" on page 66.

Migrate to a Later Release NetView Program at Each Host

In networks that use a communication management configuration (CMC) or a focal-point organization, start migrating to a later release of the NetView program (V2R4 or later) at the CMC or the focal-point host. Then distribute the migration to other hosts throughout the network.

Use the RMTCMD Command and LU 6.2 Sessions for Cross-Domain Communication

You can gradually migrate older NetView nodes to use the RMTCMD command and LU 6.2 sessions.

In a multiple CMC or multiple focal-point enterprise, update all CMCs or focal points to use the RMTCMD command and LU 6.2 sessions before you migrate these nodes to use extended multiple console support consoles. Also, in networks that use distributed automation, update all NetView programs that exchange messages to use the RMTCMD command and LU 6.2 sessions before you migrate the programs to use EMCS consoles. In both cases, if possible, complete the migration to the RMTCMD command and LU 6.2 sessions before you use extended multiple console support consoles, to avoid losing MDB data such as highlighting and some DOM information.

Restrict Operator Access to the MVS VARY Command

Unless restricted from doing so, an operator can use the VARY command to change attributes of an EMCS console, such as the route codes that the console receives. This type of change can cause duplicate message delivery and duplicate automation. Therefore, restrict the use of the NetView MVS command by using NetView command authorization checks, or by protecting the VARY command in a system authorization facility (SAF) product. For instance, protect the NetView MVS command using NetView command authorization table, or a system authorization facility (SAF) product such as RACF (Resource Access Control Facility). Refer to the *IBM Tivoli NetView for z/OS Security Reference* for information about command authorization.

Restrict AUTO Attribute of EMCS Consoles

The AUTO attribute causes a console to receive messages marked for automation using the MPF or the MRT statement ('Y' AUTOMATE). Avoid duplicate automation by ensuring that no NetView console has the AUTO attribute.

Chapter 7. Automation in an MVS Sysplex

This chapter describes an MVS sysplex, some of the advantages, suggestions for automation in a sysplex, and how to plan for automation in the sysplex.

MVS Sysplex

An MVS ***sysplex*** is a configuration of multiple MVS operating systems working as a single system by sharing functions and programs. An MVS component that enables these multiple MVS systems to operate as a sysplex is the cross-system coupling facility (XCF). The XCF provides coupling services so that authorized programs on one of the MVS systems can communicate, or exchange data, with programs on the same MVS system or other MVS systems. A major purpose of XCF is to enable multiple MVS systems to appear to be one system.

With XCF, a multiple-system environment is defined as two or more MVS systems residing on one or more processors. If there are two or more processors, the processors must:

- Be interconnected by one or more channel-to-channel (CTC) connections, or one or more coupling facilities
- Use the External Time Reference (ETR)
- Share an XCF couple data set

The set of one or more coupled MVS systems in a sysplex is given an XCF sysplex name so that authorized programs in the systems can use the XCF coupling services. XCF monitors the systems in the sysplex and can remove a failing system from the sysplex with minimal operator intervention.

In the sysplex, messages can be routed to a console on one MVS system from the other systems in the sysplex. Also, commands can be routed from a console on one MVS system to the other systems in the sysplex.

Each console name used in the sysplex must be unique.

Refer to the MVS library for information about planning the management of a sysplex.

Using NetView Program Automation in a Sysplex

One important advantage for using NetView automation in a sysplex is that the NetView program can receive messages from any or all members of a sysplex and can issue automatic responses to the appropriate member of the sysplex.

These are some methods for using NetView program automation in a sysplex:

- Develop a strategy for using different segments of the NetView program automation table to handle messages for different systems in the sysplex. You can add SYSID condition items to your existing automation table statements to block messages from certain systems, or to invoke certain automation table actions based on the system ID.
- Develop a strategy for naming consoles or use the NetView program's ConsMask capability to ensure unique names. The default is to assign console

names to be the same as the operator names. For details about the GETCONID and SETCONID commands, see the NetView program online help. For details about ConsMask feature, see the *IBM Tivoli NetView for z/OS Administration Reference* .

Planning for Automation in a Sysplex

Before you can start to plan for automation in a sysplex, you must be familiar with the planning required for managing a sysplex. Refer to the MVS library for information about managing a sysplex.

You can also use the Message Revision table (MRT) to perform the functions provided by the message processing facility (MPF). Additional information about the Message Revision table can be found in "Message Revision Table" on page 25.

Because a sysplex involves coordinated interaction among several MVS systems, planning for automation in a sysplex can be an intricate process. To help you in the planning process, the remainder of this section is divided into major stages. These stages are units of information presented in the order that you must consider the information when planning. Consider all of this information carefully.

Stage 1. Become Familiar with EMCS Consoles and How Their Attributes Affect Message Routing in a Sysplex

Review the information about extended multiple console support consoles given in Chapter 6, "Automation Using MVS Extended Multiple Console Support Consoles," on page 65. Next, consider these items:

- Console names must be unique across the sysplex.
- The value of the MSCOPE console attribute determines the MVS system or systems from which a console receives messages. Carefully consider these MSCOPE values in a sysplex, especially if you do not plan to use the NetView program defaults. You can set the MSCOPE value for a console to receive messages from:
  - The system on which the console is running
  - All systems in the sysplex
  - A list of systems within the sysplex
- The CMDSYS console attribute defines which system in a sysplex acts on MVS commands. With the CMDSYS attribute, a NetView operator can automatically direct all of that operator's MVS commands to a particular system of the sysplex. Consider what function each console has in the sysplex. Consider also that:
  - The default CMDSYS setting is the local MVS system; the system on which the NetView program is running.
  - One operator on a particular NetView program might want to issue commands to another member of the sysplex, exclusively. Therefore, the CMDSYS attribute must be set to that system.

Stage 2. Coordinate MPF Actions with the Definitions of EMCS Consoles

Because the automation of responses to MVS messages is affected by the message processing facility (MPF) table, coordinate the actions performed in MPF with the definitions you plan to use for extended multiple console support consoles. For example, decide which console receives messages marked with the AUTO(YES) keyword, and decide what MSCOPE values to use. Consider this information:

- Each message is processed by only one MPF table, which is the MPF table in the system that originated the message. However, the message can be processed by other facilities, such as the NetView program automation table, in the other systems in the sysplex.
- Defining the MPF tables the same way in all systems in the sysplex is not necessary, but might be wanted to ease maintenance or if workloads can be moved from one system to another during recovery actions.
- Define the MPF table for each system to provide processing for all MVS messages generated on that system. Understand and anticipate the effect of the additional processing or automation that the messages might undergo on other systems in the sysplex.

Stage 3. Decide Whether to Centralize Your NetView Program Automation on One System of the Sysplex

Although it is usually most efficient to provide automation as close to the source as possible, you can centralize system automation. If you run JES3, see Chapter 36, "Job Entry Subsystem 3 (JES3) Automation," on page 491 for considerations in centralizing automation in a JES3 environment.

You can set MSCOPE values for extended consoles so that one system in the sysplex can receive all system message traffic and provide automated responses. Use the MVS VARY command or the OPERPARM segment in RACF (or equivalent system authorization facility) to set the MSCOPE attribute for EMCS consoles.

Your automation actions, command lists, and command processors must use the MVS ROUTE command to route the automation action back to the appropriate system. You can determine which system in the sysplex issued the message by checking the SYSID condition item in the automation table. The SYSID information is also available to command lists written in REXX and the NetView command list language or command processors. Refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language* for information about using SYSID.

If the message traffic is extremely heavy, centralizing your system automation might not be a good option for your enterprise. Criteria have not been established for determining how much traffic can be handled with acceptable performance when using centralized system automation.

How Foreign Messages are Processed

A *foreign message* is defined as a message that originated in a different system from the local system in a sysplex. There are various ways to control how foreign messages are processed. When foreign messages are received by the local system, they are first passed through any SSI exits that are active, unless the .FORNSSI MPFLSTxx statement prevents the SSI exit from receiving them. (See the description of the .FORNSSI statement in the z/OS MVS Initialization and Tuning Reference, SA22-7592, for more information.) Because Message Revision Table (MRT) processing occurs during SSI exit processing, actions can be taken for foreign messages at that time (see Chapter 13, "The Message Revision Table," on page 127 for additional information on this topic). MVS then determines which consoles are to receive the foreign message. If the message is destined for a console owned by a NetView system operator, the message is delivered there and the message is considered solicited. If it is not destined for a NetView system operator, it is considered unsolicited and the following are true:

- The NetView program SSI determines whether a copy of the message should be forwarded to the NetView address space. By default, foreign messages are not forwarded to the NetView program address space. This can be overridden by setting AUTOMATE=Y in the MRT.
- The CNMCSSIR task receives the message if the message is marked as automatable.
- In order to reduce the chances of re-revising or re-automating a message, MRT processing and message automation should be performed as close to the source of the message as possible.
- Once an unsolicited foreign message is received by the NetView program address space, automation is performed against it as normal.

Chapter 8. Automation with the Resource Object Data Manager

This chapter introduces the Resource Object Data Manager (RODM) and describes some of the advantages, implications, and planning considerations for using RODM in automation.

For more information about the object-oriented terms used by the NetView product to describe RODM and its data model, refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

Introducing the Resource Object Data Manager

RODM is a data cache that is maintained in high-speed storage. RODM can hold many types of information about network and system resources. Because RODM keeps this information in high-speed storage, the NetView program can retrieve and update the information faster than if it were held in most other types of storage.

The NetView program can use RODM and the resource information held in RODM to assist in network and system automation.

Interactions with RODM

RODM can use programs called methods to perform many functions that retrieve, update, and manipulate information within RODM. To perform special user-defined functions in RODM, users can write their own methods and have RODM call the methods. Users can write methods in either PL/I or C language. Methods can be called directly from application programs (such as NetView program command processors) or can be triggered automatically when RODM fields (such as the status of an object) change.

Users of RODM who are not using the NetView program can interact with RODM through an application programming interface that RODM provides. Through the API, an application program can retrieve and update the resource information held in RODM or can call RODM methods. You can write application programs for RODM in PL/I, C, or assembler language.

The MultiSystem Manager RODM Access Facility provides a fast and efficient REXX program interface to RODM. It gives you the ability to create, update, query, and delete objects from RODM.

NetView program command processors can get values from RODM, change information in RODM, and call RODM methods. You can write NetView program command processors in PL/I, C, or assembly language. REXX programs, NetView program automation table statements and, to some degree, command lists can also call methods and can change information in RODM by using the ORCONV or the FLCARODM command.

See Chapter 28, "Automation Using the Resource Object Data Manager," on page 407 for examples of using method EKGSPPI and the ORCONV command.

The RODM methods can call any NetView program command list or command procedure by using the EKGSPPI object-independent method. For an example of using EKGSPPI, see Chapter 28, "Automation Using the Resource Object Data Manager," on page 407.

Using RODM in Automation

As an example of how you can use RODM, an application program for RODM can enable some external event, such as a change in status of a resource, to update the associated resource information in RODM. This update starts a specific RODM method. The method, in turn, can compare the updated information with other information in RODM, according to a predefined algorithm, and issue an appropriate response. Thus, by maintaining resource information in storage and by providing rapid access to the information through an API and through some of the methods, RODM can assist in determining the correct automatic responses to various network and system events.

Also, an "automation in progress" indicator is maintained in RODM for each resource affected by automation. This enables operators who are viewing the resource with the NetView program management console to wait until the automation is complete before attempting to fix a problem with the resource.

For more information about RODM and about writing RODM application programs and methods, refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

Advantages of Using RODM

RODM can accept and retain many types of information about resources, such as status, history, and configuration information. With the types and amount of information retained, more data is available to help in analyzing the causes and remedies for resource problems. Because RODM retains information in memory, you can quickly update and retrieve this resource information.

RODM retains information as objects and collections of objects and can associate objects according to program-defined relationships. Because the relationships among pieces of information can be specified in RODM, you can determine interactions between events and use this information in analyzing problems.

You can use RODM methods to provide automatic responses to network and system events. RODM methods can start NetView program routines, and the NetView program routines (including automation table statements) can start RODM methods.

Note: You might need to write NetView program command lists, NetView program command processors, or NetView program automation table statements to retrieve and update RODM information from the NetView program.

Planning for Using RODM in Automation

This section describes items to consider as you plan how to use RODM in the NetView program automation of your network and system. Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for more information.

Determining the Types of Events to Produce Automated Responses from RODM

RODM can produce automated responses to many types of network and system events. For some events, however, automated responses are best generated by the automation table alone or by a combination of the automation table and RODM. You need to determine the best method and best component (or components) to use for responding to each type of event.

For example, the automation table is best suited for automating responses to simple, quick events because the automation table is faster than RODM for such automation tasks and is simpler to code. RODM is best suited for automating responses to complex events that result from multiple messages or alerts. RODM is also best suited for automating responses that:

- Require more information to determine an appropriate response than is usually available with the automation table alone
- Require analysis of conditions before issuing a response
- Can take advantage of the algorithms in existing RODM methods or RODM application programs

Understanding RODM Automation Capabilities

Before using RODM in automation, review an outline of events (a scenario) that uses RODM capabilities for automation. For an example of a RODM scenario, see Chapter 28, "Automation Using the Resource Object Data Manager," on page 407.

Chapter 9. NetView Program Information Routing for Automation

The chapter explains how automation information is routed. These routing details include topics such as:

- NetView Program Interfaces
- NetView Message Routing
- NetView Hardware-Monitor Data and MSU Routing
- NetView Program Command Routing

In many cases, NetView program message and data flows are complex. However, being familiar with the general path of information can help you ensure that the messages and MSUs you want to automate go to the automation facility.

For example, you might want the automation table to generate automatic responses to network management vector transports (NMVTs) from the VTAM program's communication network management (CNM) interface. To do so, ensure that nothing impedes the flow of NMVTs to the automation table.

Another use of routing information is to determine what happens when two NetView facilities specify conflicting attributes for a single message or MSU. For example, an MSU might go to the hardware monitor, then to the automation table, then to installation exit DSIEX16B. Knowing that exit DSIEX16B processes the data last can help you determine that the attributes set by the exit take priority.

NetView Program Interfaces

NetView program automation can monitor data-processing events by receiving messages, MSUs, and other data from a variety of sources. Similarly, the NetView program can issue commands to many different targets and destinations. Figure 8 on page 82 shows commonly used interfaces for receiving event notifications and issuing commands. See Chapter 2, “Overview of Automation Products,” on page 21 for diagrams that elaborate on the interface to the operating system.

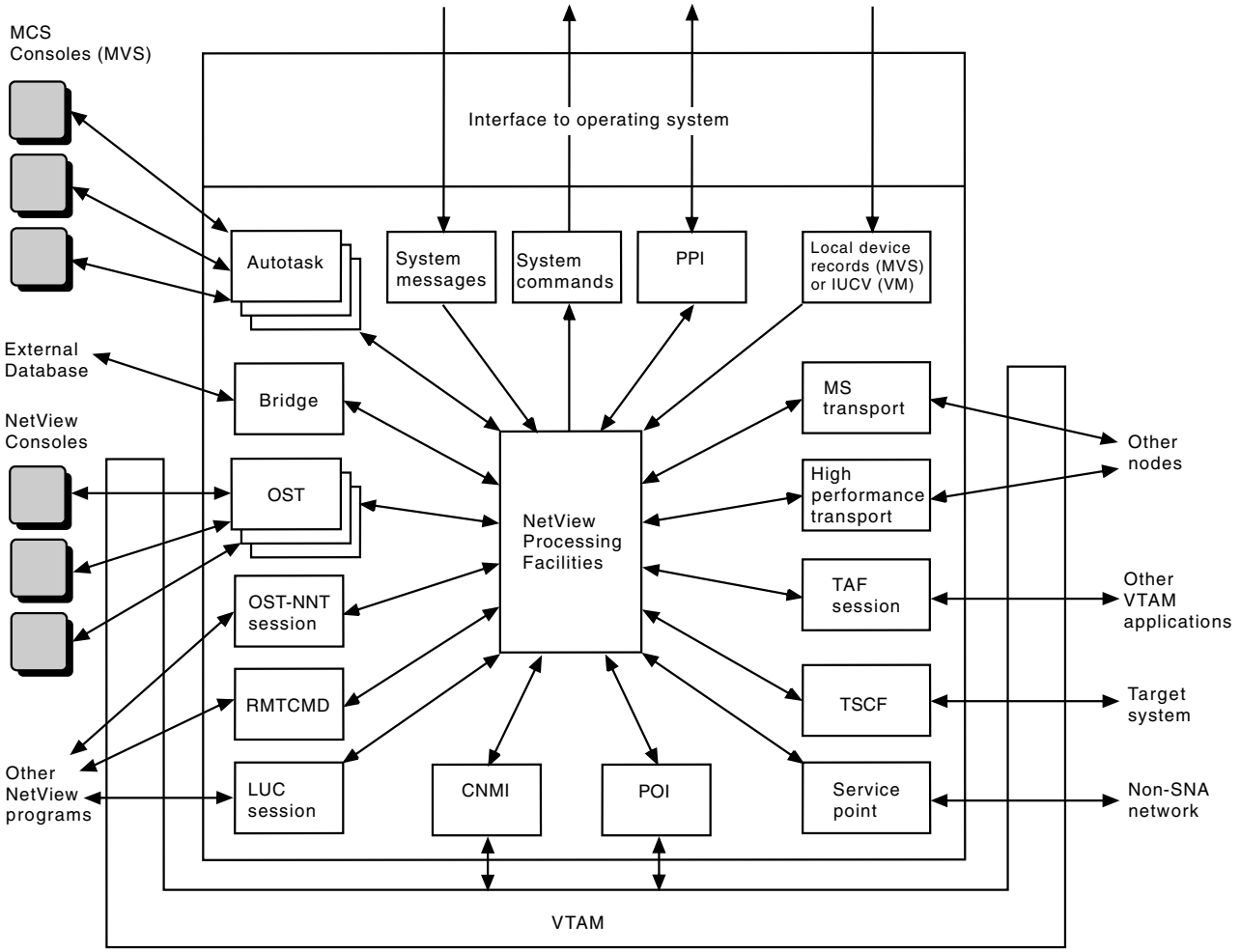


Figure 8. NetView Interfaces Used in Automation

Interfaces to the Operating System

To implement system automation, begin by giving the NetView program access to information that describes the state of the operating system, subsystems, and applications. Also set up the NetView program to send commands to the system and receive command responses. You can enter commands and receive responses using the subsystem interface or extended multiple console support (EMCS) consoles. Other interfaces that you can use for system automation include:

- System Automation for OS/390 Processor Operations can intercept traffic on system consoles.
- The NetView program terminal access facility (TAF) can intercept messages from other applications, including system applications, to their own consoles.
- Local devices of MVS can pass certain types of system problem notifications to the NetView program for processing.
- The MS transport and the high-performance option of the MS transport allow LU 6.2 communication between two applications. One use of the transports is to pass information between a system application and the NetView program.
- The program-to-program interface accepts MSUs from system applications running with the NetView program on the same system and can pass them to the NetView program hardware monitor or to the automation table.

If you intend to automate your system, ensure that the messages and other information you want to automate come to the NetView program.

See Chapter 2, “Overview of Automation Products,” on page 21 for an overview of the relationship between the operating system and the NetView program in system automation.

For step-by-step information about how to set up system communication, see “Establishing Communication between the NetView System and the Operating System” on page 291.

If you need system flow information in more detail, see Appendix D, “MVS Message and Command Processing,” on page 523.

Interfaces to Other NetView Programs

With these interfaces, you can send information between two systems running the NetView program:

- The RMTCMD command, for sending commands to other NetView programs and receiving any messages generated in response
- OST-NNT sessions, an alternative way of sending messages and commands between NetView programs
- LUC sessions, for forwarding alerts or status information to a focal point

You can also use TAF sessions, the MS transport, and the high-performance option of the MS transport for NetView program to NetView program communication. See Chapter 26, “Centralized Operations,” on page 373 for a discussion of NetView program to NetView program communication.

Other Message and Command Facilities

Other NetView program facilities for receiving messages and sending commands include:

- The program operator interface (POI) for VTAM messages
- NetView Bridge for communication with Information/Management and other external databases

Interfaces for Hardware-Monitor Data and MSUs

The NetView program enables direct automation of MSUs. You can also automate hardware-monitor data other than MSUs, by first converting them to messages. Hardware-monitor data and MSUs come to the NetView program from these sources:

- The communication network management interface (CNMI), for problem records from an SNA network
- Service points for NMVTs from non-SNA sources
- Local devices, for problem records from the operating system
- The program-to-program interface, for MSUs from other applications running with the NetView program on the same system
- The LU 6.2 transports, for MSUs from LU 6.2 applications

NetView Program Message Routing

After the NetView program receives a message, NetView program routing facilities control the destination of the message within the NetView program. You can use routing facilities to choose the operators who see the message or the autotasks that process it. To control message routing effectively, you must understand the distinction between solicited and unsolicited messages. You must also be familiar with the major routing facilities and the path of a message through the NetView program. If you need more information about message paths, see Appendix F, "Detailed NetView Message and Command Flows," on page 537.

NetView treats a message as ***solicited*** if a specific destination for the message is known; otherwise, the NetView program treats the message as ***unsolicited***.

Solicited Messages

The NetView program queues solicited messages to the known destination task: a NetView operator, an autotask, or a NetView program to NetView program task (NNT). These are examples of solicited messages:

- Responses to NetView commands.
- MVS system messages delivered directly to a NetView operator station task (OST) that obtained an EMCS console.
- Responses to system commands issued from a NetView operator console. See "Command Flow" on page 525 for details about how solicited messages are returned in response to system commands.
- Responses to VTAM commands.
- Messages issued from a terminal access facility (TAF) operator-control session with an application such as CICS or Information Management System (IMS), including all messages received from those applications on a TAF session.
- Messages issued as a result of a TRACE or SAY instruction in a NetView REXX command list.
- Messages issued by the NetView MSG command or the MVS and TSO SEND commands.

System messages from MVS can be either solicited or unsolicited. See "Unsolicited Messages from an MVS System" on page 85 for a description of unsolicited MVS system messages.

Unsolicited Messages

A message is unsolicited if a specific destination task is not known. For example, VTAM might send a message to the NetView program that is unrelated to any request by the NetView program, through the primary POI (program operator interface). The NetView program also regards a message as unsolicited if it is directed to the primary POI task (PPT), because the PPT cannot display messages. An MVS message that is a response to a command issued by the PPT routed through the subsystem interface is also regarded as unsolicited.

Note: The hardware monitor submits only unsolicited MSUs to automation.

The Authorized Receiver

Because there is no specific destination task for an unsolicited message, the NetView program routes all unsolicited messages to the authorized receiver, unless you use the ASSIGN command or the automation table to provide a destination.

The ***authorized receiver*** is simply a NetView operator you have authorized to receive unsolicited and authorized messages that do not have another destination.

Use the AUTH statement in an operator profile to determine the authority of a particular operator. All operators with AUTH MSGRECVR=YES in their profiles are permitted to be the authorized receiver. However, a NetView program has only one authorized receiver at a time.

Unsolicited Messages from a DST

Unsolicited messages from a data services task (DST) go to the task that started the DST (if it is still active), rather than to the authorized receiver. Although the ASSIGN command cannot affect routing of unsolicited DST messages, the automation table can affect the routing.

Unsolicited Messages from an MVS System

Unsolicited messages received from an MVS system are not sent to the authorized receiver. You can use the ASSIGN command to re-route these messages to another task. See "Using ASSIGN to Route Solicited Messages" on page 88 for more information about the ASSIGN command.

If you do not use the ASSIGN command, the CNMCSSIR task scans the automation table for each unsolicited message. The scan might result in a match in the automation table.

If one of the actions specified for the matching statement is to run a command and if the command is not routed to a logged-on task, the command specified in the automation table statement is ignored and a DWO050E message related to `Invalid cmd:` is added to the Netlog log.

If an operator is specified with the ROUTE action in the automation table, the command is sent to that operator instead of the primary autotask, that is, the task defined by the `function.autotask.primary` style statement.

All MVS system messages received by the CNMCSSIR task are unsolicited messages. System messages received by any other NetView operator task are solicited messages.

Message Routing Facilities

You can control message routing in the NetView program with installation exits, the automation table, and the ASSIGN command, the MSGROUTE command, and the Pipes ROUTE command. For MVS systems with EMCS consoles, you can also set your EMCS console attributes to control message routing. For example, you can route messages to an EMCS console based on the message route code. The MVS system messages with that route code are directly delivered to the NetView program task that obtained the EMCS console.

Attention: If you use route codes to route messages directly to EMCS consoles, duplicate automation of some messages can result. For more information, see Chapter 6, "Automation Using MVS Extended Multiple Console Support Consoles," on page 65.

Chapter 15, "The Automation Table," on page 147 and Chapter 17, "Installation Exits," on page 285 describe the automation table and installation exits, respectively. These sections discuss the ASSIGN command, the MSGROUTE command, and routing messages based on route codes.

Routing Messages with the ASSIGN Command

These sections describe how to use the ASSIGN command. For most message routing, use the automation table rather than using the ASSIGN command, as discussed in "ASSIGN Command Versus Automation Table Routing" on page 89. However, the ASSIGN command is useful for such things as assigning operators to groups and routing messages to autotasks to speed up automation.

The MVS system messages that are delivered directly to EMCS consoles in use by NetView program OSTs are considered solicited, and therefore are not subject to ASSIGN PRI and ASSIGN SEC processing. There is one exception: If the PPT has an EMCS console, the solicited messages sent to the PPT can be processed with ASSIGN PRI and ASSIGN SEC. The MVS system messages that are delivered to the EMCS console obtained by the CNMCSSIR task are considered unsolicited messages.

Assigning Messages to Operators

The MSG option enables you to direct copies of solicited, unsolicited, or authorized messages to:

- A particular operator
- A group of operators
- The system operator (SYSOP)
- The network log (LOG)

The ASSIGN command enables the operator to change message routing without editing and reloading the automation table.

Assigning Operators to Groups

The GROUP option enables you to assign a list of operators to a particular group. You can then use the operator group with other ASSIGN commands, with the MSGROUTE command in a command list, and with the EXEC(ROUTE) action in the automation table.

If you specify ROUTE(+*groupname*) in the automation table to route a message to a group, you can change the list of operators who receive the message by changing the contents of the group. You can issue the ASSIGN command with the GROUP option whenever you need to modify the list of operators belonging to a particular group.

Note: Because assignment changes are difficult to monitor, when you are setting the ASSIGN Options, consider authorizing operators to issue only the GROUP option. You can use the NetView LIST command to monitor what is assigned at any given time.

For more information, refer to the *IBM Tivoli NetView for z/OS Security Reference*.

Using ASSIGN to Route Unsolicited Messages

With the PRI option of the ASSIGN command, you can specify a list of operators to receive unsolicited or authorized messages. You can specify:

- An operator
- An autotask
- A list of operators and autotasks
- A group ID
- The system operator (SYSOP)
- The network log (LOG)

Only one operator receives each message. If you specify a list or a group ID, only the first operator in the list or group that is logged on receives the message.

The message sent to the primary receiver is flagged with a percent sign (%) in the last position of the DOMAINID field. The NetView program displays the percent sign (%) on the screen with the message and also records the percent sign in the network log. The percent sign does not appear in the HDRDOMID field of BUFHDR.

Installation exits that need to determine whether a message is a primary copy must check the IFRAUPRI and IFRAUSEC fields of the internal function request.

If you issue the ASSIGN command for a message with PRI=SYSOP or PRI=LOG, the NetView program automation table does not process the message.

With the SEC option of the ASSIGN command, you can specify a list of operators to receive secondary copies of the unsolicited or authorized messages. Before you can generate SEC copies, you must have a PRI assignment for a message.

All operators, or groups of operators, in the SEC list receive the message if:

- They are logged on
- At least one operator in the PRI list is logged on

The message sent to the SEC receiver is flagged with an asterisk (*) in the last position of the DOMAINID field. The NetView program displays the asterisk when displaying the message and also places the asterisk in the network log. The asterisk does not appear in the HDRDOMID field of BUFHDR.

Installation exits can check the IFRAUSEC field in the automation internal function request (AIFR) to determine whether a message is a secondary copy.

If no primary receiver is logged on, the NetView program continues as if you had not made an assignment. The routing of the message does not change, and a secondary copy of the message does not go to secondary receivers. To ensure that a message assignment does take effect and that secondary copies go to secondary receivers, you might want to include several operators on the PRI list or use a stable autotask as one of your primary receivers.

These points apply to secondary copies:

- They are not subject to automation table processing unless they are routed cross-domain to another NetView program operator. Secondary copies routed cross-domain are subject to automation table processing in the cross-domain NetView program.
- They are subject to WAIT processing in command procedures.
- They are useful for displaying messages to several operators.

You can use the ASSIGN command to route unsolicited messages. The command in Figure 9 routes all unsolicited messages to the first operator who is specified on the PRI option and who is logged on.

```
ASSIGN MSG=*,PRI=(OPER1,AUTO1),SEC=(NETOP1,LOG)
```

Figure 9. Using the ASSIGN Command to Route Unsolicited Messages

The NetView program logs each copy in the network log unless you indicate otherwise in installation exit DSIEX04. In the previous example, because LOG is specified in the SEC list of operators, duplicate logging occurs unless OPER1, AUTO1, and NETOP1 have suppressed logging.

Using ASSIGN to Drop Unsolicited Messages

You can also use the DROP option of the ASSIGN command with the MSG option or the GROUP option. When used with the MSG option, DROP=AUTH drops the specified messages from the PRI and SEC assignments. For example if you type the command shown in Figure 10, the system does not drop all assignments; it drops the assignments you made using MSG=*. (AUTH is the default value and does not have to be specified.)

```
ASSIGN MSG=*,DROP=AUTH
```

Figure 10. Using the ASSIGN Command to Drop Unsolicited Messages

Using ASSIGN to Route Solicited Messages

With the COPY option of the ASSIGN command, you can specify a list of operators who receive a copy of a solicited message. You can specify:

- An operator
- A list of operators
- A group ID
- The system operator (SYSOP)
- The network log (LOG)

Copies of the solicited message go to all recipients who are in the copy list and are logged on.

The message sent as a copy is flagged with a plus sign (+) in the last position of the DOMAINID field. The NetView program displays a plus sign (+) on the screen when the message is issued and also places a plus sign in the network log. The plus sign does not appear as part of the HDRDOMID field of BUFHDR. Installation exits can check the IFRAUCPY field of the internal function request to determine whether a solicited message is a copy.

These points apply to copies generated by the ASSIGN COPY option:

- They are not subject to automation-table processing unless they are routed cross-domain to another NetView program operator. Such copies are subject to automation table processing in the cross-domain NetView program.
- They are subject to WAIT processing in command procedures.

The first command in Figure 11 sends copies of all solicited messages to both NETOP1 and OPER1 (if they are logged on).

If you issue the ASSIGN command with DROP=COPY, the COPY assignments are dropped for the specified messages. The second command in Figure 11 drops those messages assigned with MSG=* from the COPY assignment, type `ASSIGN MSG=*,DROP=COPY`.

```
ASSIGN MSG=*,COPY=(NETOP1,OPER1)
```

Figure 11. Using the ASSIGN Command to Route Solicited Messages

Using ASSIGN to Route Messages to Autotasks

If your automation slows because many messages are queued on a single task, waiting for automation table processing, you can use the ASSIGN command to

split the messages among several tasks. In this case, you can still use the automation table for final routing of the message.

Note: The ASSIGN command cannot route messages to an optional task. See "Actions" on page 209 for details.

Using ASSIGN with Automation Logic

Independently from the specification of the destination of the ASSIGN command, you can apply automation logic to determine whether messages are routed to their assigned destination. When used with the MEMBER option, the ASSIGN command can be used to denote a DSIPARM member or PIPE message data that has automation table statements. These statements are compiled into an automation table. When messages pass through this table, it is determined whether they satisfy ASSIGN routing criteria. For more information on the ASSIGN command, refer to the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* or the online help.

Using the REFRESH and ASSIGN Commands for Dynamic Operator Control

Using the REFRESH command, you can dynamically delete operators and dynamically add operators without predefining the operators to the NetView program. The ASSIGN command enables you to assign messages to operators that are not presently defined to the NetView program. If you assign messages to an operator before you define the operator to the NetView program, you receive a message informing you that the operator specified in the ASSIGN command is not presently defined to the NetView program. The assignment is then completed successfully.

When the defined operator logs on, the operator begins receiving messages. Regardless of whether an operator is defined to the NetView program, messages assigned to operators that are not logged on are delivered to the next assigned operator or to the original destination.

If an operator definition is deleted using the REFRESH command, the operator session continues until that operator logs off. Messages assigned to operators that are logged on but no longer defined to the NetView program are still delivered to that operator.

ASSIGN Command Versus Automation Table Routing

You can use the ASSIGN command to route solicited and unsolicited messages. ASSIGN is most useful for assigning operators to groups, for preliminary routing of messages to autotasks to get messages to the automation table faster, and for assigning messages to the system operator. Otherwise, it is usually preferable to use the automation table for message routing, for these reasons:

- Message routing with the ASSIGN command occurs in a specific-to-general order, regardless of the order in which you issue ASSIGN commands. Figure 12 shows examples.

```
ASSIGN MSG=IST*,PRI=(VTAMOPER,AUTO1)
ASSIGN MSG=IST5*,PRI=(VTAMOPER,AUTO2)
```

Figure 12. General and Specific Message Routing

  Notice that the routing specified in the second command occurs first because IST5* is more specific than IST*. If a third ASSIGN command, such as this

example, is issued to undo the message routing specified in the first ASSIGN command, the second ASSIGN command is still processed.

```
ASSIGN MSG=IST*,DROP
```

An operator who wants to drop all ASSIGN commands for IST messages needs to know about the second command as well as any other commands issued for IST messages. The operator can then issue the appropriate commands to drop the ASSIGN commands.

When several different operators, command lists, and command processors are issuing ASSIGN commands, they are not necessarily aware of other assignments. Therefore, message routing with the ASSIGN command can be difficult to monitor. With the automation table, message routing is centralized, and thus is easier to monitor.

- If you route all messages with the automation table, the table is easier to maintain because all of the routing instructions are in one file or set of files. You are less likely to create conflicting route instructions and can correct them more easily if you do.
- When you route messages with the NetView program automation table, you usually do not need to be concerned about whether messages are solicited or unsolicited. However, you can use the automation table to identify whether messages are solicited if you desire. Bit 16 of IFRAUIND indicates whether the NetView program treats a message as unsolicited. You can use the IFRAUIND automation table action to check this bit.

Routing Messages with the MSGROUTE Command

You can use the MSGROUTE command to direct copies of messages to:

- A particular operator or autotask
- A group of operators
- The system operator (SYSOP)
- The NetView program hardcopy log
- The network log (LOG)

You can issue the MSGROUTE command from a command list initiated from the NetView program automation table. Like the NetView program automation table, MSGROUTE can set such actions as BEEP or DISPLAY for the message. However, actions specified on the MSGROUTE command cannot override the actions specified in the NetView program automation table for a given message. The NetView program does not send the message to the automation table again when the message is routed with the MSGROUTE command. However, if a copy is routed cross-domain, the cross-domain automation table processes the message.

Using the MSGROUTE command can help you decide where to route a message or what action to take without more information. For example, you can review a command list to check the second line of a multiline message before deciding where to route the message.

Routing Messages to EMCS Consoles Based on Route Codes

To route MVS system messages based on their route codes, set up your EMCS consoles to receive the route code or codes that interest you. To route messages based on route codes, also eliminate any duplicate message automation.

Specifying the Route Codes

You can use the Resource Access Control Facility (RACF) OPERPARM segment or the ROUT keyword on the MVS VARY command to specify the route codes you

want to receive at an EMCS console. The NetView program treats these messages as solicited messages because by requesting a specific route code, you have given the messages a known destination.

Eliminating Duplicate Automation of Messages

By default, all messages marked AUTO(YES) or AUTO(*token*) in the MVS message processing facility (MPF), or which are subject to NETVONLY or REVISE("1" AUTOMATE) revision table actions or similar, are delivered to the EMCS console obtained by the CNMCSSIR task. Also, by default, no MVS system messages are routed to this console based on route code. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for information about attributes for EMCS consoles.

If you use route codes to send messages directly to EMCS consoles, some messages might be automated twice because they are also delivered to other EMCS consoles based on other routing criteria. Examine the attributes of every EMCS console in your system to avoid duplicate automation.

The best way to avoid duplicating automation is to avoid using route codes to send messages directly to EMCS consoles. However, if you do use route codes to send messages directly to EMCS consoles, you can use the automation table to help avoid duplicate automation.

Some causes of duplicate automation include:
- Action messages routed to multiple operators
- Command lists called more than once for a single message

You can use NetView program automation table condition items such as OPID and ROUTCDE to ensure that specific automation actions are performed only once for a given message.

Message Routing Flow

The message routing flow in the NetView program is:

1. DSIEX17 processing
2. PIPE CORRWAIT
3. ASSIGN PRI/SEC processing
4. Authorized receiver processing
5. DSIEX02A processing
6. Wait processing
7. Automation table processing
8. DSIEX16 processing
9. ASSIGN COPY processing
10. Discard or display processing

Table 2 on page 92 shows the routing steps for these message types:
- Unsolicited messages from the MVS subsystem interface
- Other unsolicited messages
- Solicited messages

Read the table as if a message enters the top and flows down through the table. If the classification of a message changes, the flow of the message continues in the

new column of the table without repeating any steps already taken. The NetView program invokes the automation table and each installation exit only once for each original message.

For example, an unsolicited message from VTAM flows through the steps in the All Other Unsolicited Messages column. The unsolicited message undergoes ASSIGN (PRI/SEC), authorized receiver, DSIEX02A, and automation-table processing. Suppose that the automation table routes the message to an autotask. Thereafter, the NetView program treats the message as solicited. The message flow proceeds as described in the All Other Solicited Messages column without repeating any of the processing that has already taken place. The solicited message undergoes wait, DSIEX16, ASSIGN(COPY), logging, and display processing.

Table 2. NetView Program Message Routing

Step	Unsolicited MVS Messages	All Other Unsolicited Messages	Solicited MVS Messages	All Other Solicited Messages
DSIEX17	●		●	
PIPE CORRWAIT			Note 2	Note 2
ASSIGN (PRI/SEC)	●	●		
Authorized Receiver		●		
DSIEX02A	●	●	●	●
Wait Processing	Note 1	Note 1	●	●
NetView Program Automation Table	●	●	●	●
DSIEX16	●	●	●	●
ASSIGN (COPY)			●	●
Logging	●	●	●	●
Display to the NetView program			●	●
Display to System		●	●	●
Discard	●			

Notes:

1. Wait processing for unsolicited messages occurs only when the message is routed to a task that is waiting.
2. When a message is solicited by a command in a pipeline, all subsequent routing is superseded and does not occur. If the pipeline re-issues the message, it is treated like a non-MVS solicited message.

DSIEX17 Processing

Installation exit DSIEX17 is called to process all inbound MVS messages, solicited or unsolicited, or delete operator messages (DOMs). This exit can change, replace, or delete messages before the automation table is invoked. This exit enables you to delete a message or a DOM.

PIPE CORRWAIT

You can use the CORRWAIT stage of the NetView PIPE command to identify messages that:

- Are in response to a command issued from the pipeline
- Are to be processed by the pipeline

Messages are marked by exposure to installation exit DSIEX02A, ASSIGN routing, and automation. If a message has been through any of these steps and is later captured by a pipeline and reissued, it is not re-exposed to the same steps. Refer to exceptions under the ONLY option of the CONSOLE stage in the *IBM Tivoli NetView for z/OS Programming: Pipes*.

ASSIGN PRI/SEC Processing

ASSIGN PRI/SEC processing can be used only on unsolicited messages. If you are using EMCS consoles, MVS system messages that, based on route codes, are delivered directly to the NetView program OSTs are considered solicited messages; therefore, these messages are not subject to ASSIGN PRI/SEC processing. Solicited MVS system messages sent to the PPT can be processed with ASSIGN PRI and ASSIGN SEC.

The only unsolicited MVS system messages are those delivered to the CNMCSSIR task. Unsolicited messages are checked to determine if they are assigned to a primary receiver.

A primary receiver is an operator or autotask to which you have assigned the message with the PRI operand of an ASSIGN command. If a primary receiver is logged on, the message is assigned to that operator ID. Secondary copies of the message are then created for any operators specified in the SEC operand of the ASSIGN command. Secondary copies are not subject to automation table processing, except that secondary copies routed to a cross-domain NetView program are processed by the automation table of the cross-domain NetView program.

Authorized Receiver Processing

Unsolicited messages for which no primary receiver was found are directed to the authorized receiver, if one is available.

However, unsolicited messages going to a DST go to the task that started the DST in preference to the authorized receiver, if the DST was started by a task that is still active. Also,the NetView program does not send unsolicited messages from an MVS system to the authorized receiver.

DSIEX02A Processing

Installation exit DSIEX02A is called to process standard output to an operator's terminal. It can change, replace, or delete messages before the automation table is invoked.

If this exit deletes a message (with the USERDROP return code from the exit or by setting the IFRAUTBA field to B'0'), the NetView program does not search the automation table for that message or call exit DSIEX16.

If DSIEX02A sets the IFRAUMTB bit on for a message, NetView does not search the automation table for the message. However, DSIEX16 processes the message. For more information about DSIEX02A, see Chapter 17, “Installation Exits,” on page 285.

Wait Processing

After DSIEX02A processing, all routed messages are checked to determine if they satisfy an outstanding wait condition for a command procedure operating under the task to which the message was routed.

Command procedures written in PL/I, C, REXX, and the NetView command list language allow you to suspend processing while waiting for a particular message or group of messages. PL/I, C, and REXX command procedures use the TRAP and WAIT commands for this function. The NetView command list language uses &WAIT.

Messages that are subject to wait processing include:

- All messages solicited by an operator or autotask
- Copies of solicited messages created with ASSIGN COPY
- Unsolicited messages assigned to an operator or autotask with ASSIGN PRI or authorized receiver processing
- Secondary copies of unsolicited messages created with ASSIGN SEC

If the 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, the NetView program marks it for deletion. In this case, automation-table processing does not occur and the NetView program does not display or log the message. The message does go to installation exit DSIEX16. You can suppress messages in a PL/I or C command processor or REXX command list with TRAP and SUPPRESS. In the NetView command list language, you can use the &WAIT SUPPRESS statement.

Messages rerouted by the automation table can undergo wait processing a second time on the new task, as explained in "Automation-Table Processing."

Automation-Table Processing

Except for messages written directly to the log, solicited and unsolicited messages from all sources are subject to automation table processing for the original instance of the message. Copies of the message produced by the ASSIGN command with the SEC or COPY operands, by the MSGROUTE command, or by the ROUTE keyword in the automation table itself are not subject to automation-table processing. However, if you route a copy cross-domain, the automation table in the other domain processes the message.

Routing Messages

In automation-table processing, the ROUTE keyword can reroute an unsolicited message that you previously routed with ASSIGN PRI or authorized receiver processing. Similarly, you can change the automatic assignment of a solicited message to add other receivers or even to eliminate the original receiver. Copy assignment for solicited messages is not affected. Copies always go to the operators you specified with the ASSIGN COPY command.

You can code automation-table statements that direct messages or commands to any combination of operators, autotasks, operator groups, and the PPT. Routed commands can include command processors and command lists. The list of operator IDs that are to receive the message does not have to be the same as the list of operator IDs that are to process the commands you are issuing in response.

Assume that a message with an ID of DSI374A is ready to undergo automation-table processing and that the statement in Figure 13 is in your automation table.

```
IF MSGID='DSI374A' THEN
    EXEC(ROUTE(ALL OPER1 OPER2 *));
```

Figure 13. MSGID Statement in Automation Table

In this example, copies of message DSI374A are to be sent to OPER1, OPER2, and the operator associated with the message when it entered automation-table processing. Copies of messages created by the ROUTE keyword in the automation table and sent to a new task are subject to wait processing on the new task, as described in "Wait Processing" on page 94.

If a message has no match in the automation table, it goes to the receiver that was associated with that message when it entered automation-table processing. For a solicited message, that receiver is the task whose input generated the message. For an unsolicited message, that receiver is a primary receiver you assigned for the message if you assigned primary receivers and one of them is logged on.

For an unsolicited MVS system message with no primary receiver, the CNMCSSIR task scans the automation table. If a match exists, any command issued using EXEC(CMD) must be routed to a specific task using the ROUTE keyword. If no ROUTE keyword exists, the message is routed to the Primary Autotask, defined by the FUNCTION.AUTOTASK.PRIMARY statement in the CNMSTYLE member.

For an unassigned message from a DST, the default receiver can be one of these items:

- The task that started the DST (if that task is logged on)
- The authorized receiver (if there is one)
- The system console operator

Other unsolicited messages (without a primary receiver assigned) go either to the authorized receiver or to the system console operator.

Setting Message Attributes

The automation table can check or set the color and highlighting attributes of the messages. The automation table can set attributes, such as logging and display characteristics, for messages.

These automation table settings take precedence over attributes specified with the NetView DEFAULTS command. Except for message color and intensity as set with the SCRNFMT keyword, attributes specified with the NetView OVERRIDE command take precedence over the automation table settings.

DSIEX16 Processing

The NetView program calls installation exit DSIEX16 after a message is considered for automation. The exit allows the user to change message text and processing options.

For more information about DSIEX16, see Chapter 17, "Installation Exits," on page 285.

ASSIGN COPY Processing

After automation-table processing, the NetView program makes a copy of a solicited message for each designated operator if an ASSIGN COPY command is in effect. The copies take their display and logging attributes, such as DISPLAY, NETLOG, and BEEP, from the original instance of the message. Therefore, an automation table entry for the original message can also affect the copies made using the ASSIGN COPY command.

Secondary copies, created by the SEC operand for unsolicited messages, have NetView system defaults (unless you change the defaults with a DEFAULTS or OVERRIDE command). Copies created by the ASSIGN COPY process undergo the wait processing described in "Wait Processing" on page 94.

Discard or Display Processing

The NetView program either discards or displays a message after completion of routing. The NetView program discards all unsolicited MVS system messages if they have not been rerouted. Regardless of the operating system, the NetView program displays all other unsolicited messages and all solicited messages unless an installation exit or the automation table has turned off the display option for a message or messages.

NetView Program Hardware-Monitor Data and MSU Routing

This section describes the flow of data to the hardware monitor and the flow of MSUs to automation. You have several ways of sending data to the hardware monitor:

- Forwarding an alert from one NetView program to another over an LUC session
- Sending a multiple domain support message unit (MDS-MU) over the MS transport to the ALERT-NETOP application
- Sending a control point management services unit (CP-MSU) or network management vector transport (NMVT) to the hardware monitor over the program-to-program interface
- Receiving a hardware-monitor problem record (NMVT, record maintenance statistics [RECMS], or record formatted maintenance statistics [RECFMS]) over the CNM interface
- Using the GENALERT command to generate a hardware-monitor record from within NetView
- Receiving a system-format record for the hardware monitor (OBR, MDR, MCH, CWR, or SLH) from local MVS devices

Many of the records that the hardware monitor receives go to the automation table during normal processing. The automation table can change filtering and highlighting attributes or issue automatic responses. Specifically, the records that go to the automation table are NMVTs, CP-MSUs, MDS-MUs, RECMSs, and RECFMSs, collectively known as MSUs. The hardware monitor sends only MSUs containing:

- Alerts, key X'0000'
- Link events, key X'0001'
- Resolution, key X'0002'
- PD statistics, key X'0025'
- RECMSs, encapsulated in a X'1044'
- RECFMSs, encapsulated in a X'1045'
- Link configuration data, key X'1332'

A routing and targeting instruction GDS variable (key X'154D') can go to the automation table attached to an alert or resolution major vector. The hardware monitor converts certain other major vectors, such as many link events (key X'0001'), into alert major vectors. In these cases, the original major vector and the converted alert major vector go to the automation table.

The NetView program also enables you to send MSUs to the automation table directly without sending them through the hardware monitor. This capability can help you if, for example, you want to automate an MSU that does not contain a major vector that is automatically sent through the automation table.

To send an MSU directly to automation, use the CNMAUTO service routine for PL/I or C, or the DSIAUTO macro for assembler. Alternatively, use the MS transport interface and direct an MSU to the generic automation receiver (NVAUTO). The generic automation receiver is an application that simply presents an MSU to the automation table and then discards the MSU.

Figure 14 on page 98 shows the interfaces for sending problem records to the hardware monitor, the interfaces for sending MSUs to automation, and the path the data takes in each case. In the figure, each multiple domain support message unit (MDS-MU) going into the hardware monitor must contain a control point management services unit (CP-MSU). CP-MSUs going from the hardware monitor to the automation table must contain a major vector that is supported for automation. A description of the major steps is illustrated in Figure 14 on page 98.

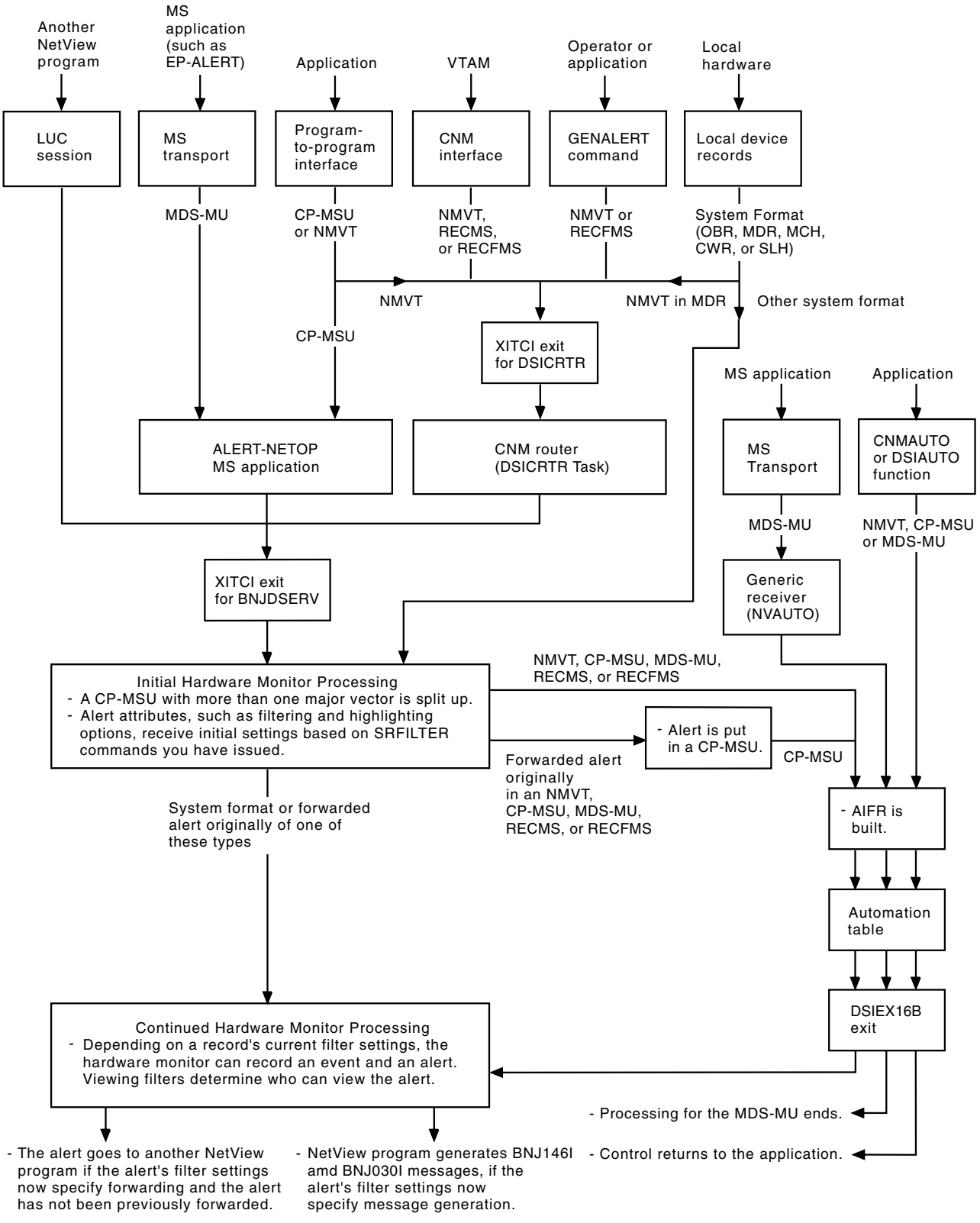


Figure 14. Flow of Data to the Hardware Monitor and MSUs to Automation

ALERT-NETOP Application

ALERT-NETOP, which is an MS application that is supplied with the NetView program, receives MSUs and passes them to the hardware monitor.

XITCI Processing

The NetView program calls installation exit XITCI for the BNJDSERV task whenever the hardware monitor receives an MSU or other problem record. If the problem record comes through the CNM router, the NetView program also calls exit XITCI for the DSICRTR task.

Either XITCI installation exit can change, replace, or delete the problem record. Any alert forwarded by an LUC session from another NetView domain is in a special forwarding format at this point. For more information about the XITCI exits or the forwarding format, refer to *IBM Tivoli NetView for z/OS Programming: Assembler* and to *IBM Tivoli NetView for z/OS Programming: PL/I and C*.

Initial Hardware-Monitor Processing

When you send a CP-MSU through the ALERT-NETOP application to the hardware monitor, either alone or in an MDS-MU, the CP-MSU can contain more than one major vector. If so, the hardware monitor first splits the data into separate CP-MSUs containing one major vector each. Thereafter, the NetView program processes each major vector separately. If the CP-MSU being split is in an MDS-MU, each of the new CP-MSUs goes in an MDS-MU with the same header information as the original. There are two exceptions:

- Basic encoding rules (BER)-encoded data that does not go through automation

  Specifically, major vector X'000F' followed by a X'130F' major vector, and major vector X'1330' followed by a X'132F' major vector, do not go through automation.
- Routing and targeting instructions GDS variables (X'154D')

  Routing and targeting information stays in the CP-MSU with the major vector that immediately follows it, but the NetView program moves the routing and targeting information to the end of the new CP-MSU.

A user-written application can submit record maintenance statistics (RECMSs) and record formatted maintenance statistics (RECFMSs) to automation just as you might submit a X'0000' major vector to automation. An application can encapsulate a RECMS in a X'1044' major vector or a RECFMS in a X'1045' major vector, and then encapsulate them again in a X'1212' CP-MSU.

You can send a RECFMS record through the ALERT-NETOP application by encapsulating the record in a X'132E' major vector within a CP-MSU in an MDS-MU. The RECFMS is then extracted and processed as normal by the hardware monitor.

Next, for all alert-type data coming to the hardware monitor, the NetView program initially sets filter and highlighting attributes based on your SRFILTER settings.

Automation-Table Processing

All MSUs processed by the hardware monitor are subject to automation-table processing if they contain X'0000', X'0001', X'0002', X'0025', X'1332', RECMSs, or RECFMSs. Forwarded alerts that were originally in MSUs on a distributed NetView system return to MSU format for automation. The hardware monitor places these alerts in CP-MSUs. System-format records, such as outboard record

(OBR), machine check handler (MCH), channel recovery word (CWR), and second-level interrupt handler (SLIH), do not go to the automation table.

The automation table can check or set any of these conditions:

- Color
- Highlighting
- Filtering attributes hardware monitor for MSUs

MSUs that do not come through the hardware monitor can come directly to automation through the CNMAUTO service routine of PL/I and C, the DSIAUTO macro of assembler, or the generic automation receiver MS application (NVAUTO), which invokes the automation table. Setting highlighting or filtering attributes does not work in these cases, because the hardware monitor does not process the MSU. However, you can use the automation table to initiate automatic commands in response to the MSU.

When automating the response to an MSU, route the command to an autotask. If the hardware monitor data services task (DST) BNJDSERV sends an MSU to the automation table and the matching statement in the table has an EXEC action specifying a command to be run has no ROUTE specification, the command goes to the OST that started BNJDSERV. If the OST is not active, the NetView program cannot route the command and issues a message to the network log to indicate the problem. Therefore, either start BNJDSERV from a stable autotask or always use ROUTE when applying an EXEC action to an MSU from the hardware monitor.

DSIEX16B Processing

The NetView program invokes installation exit DSIEX16B after an MSU is considered for automation. This exit enables you to change, replace, or delete an MSU. For more information, see Chapter 17, “Installation Exits,” on page 285.

Continued Hardware Monitor Processing

Problem records of the types processed by the hardware monitor can go into the event and alert databases, depending on the final settings of the ESREC and AREC filter attributes for the record.

If a record passes the ESREC or AREC recording filters and gets recorded as an event or an alert, operators can view the event or alert on the hardware monitor panels. Viewing filters determine which operators can view the event or alert. A percent sign (%) on the right side of the hardware monitor console marks any event or alert that matched at least one statement in the automation table.

If a record passes both the ESREC and the AREC recording filters, other filters apply including ROUTE, OPER, TECROUTE, and TRAPROUTE. For more information, see “Filtering Alerts” on page 299.

NetView Program Command Routing

You can control the routing of commands to NetView program tasks. These sections describe which commands you can route to which tasks and the facilities for routing commands.

Compatibility of Commands with Tasks

You must ensure that the command, command processor, or command list that you are routing can run under the destination task. The different classes of tasks that run under the NetView program main task are:

- Tasks that can receive messages and control the processing of commands, command processors, and command lists. These tasks include autotasks, other operator station tasks (OSTs), NetView-NetView tasks (NNTs), and the primary POI task (PPT). You can route commands, command lists, and command processors that run as regular commands (TYPE=R) or immediate commands (TYPE=I) to this type of task.

  However, some restrictions apply. Autotasks cannot run commands that produce full-screen panels. Also, use caution when having an autotask run a command procedure that includes wait processing. To avoid the possibility of indefinite waiting that ties up an autotask, use a timeout value on the WAIT instruction. Some commands cannot run under the PPT. These include commands that produce full-screen panels, commands that do wait processing, and several others.

  Refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language* for information about wait processing.
- If the BNJDSERV DST or the CNMCSSIR task sends an MSU to the automation table and the matching statement in the table runs a command but has no ROUTE specification, the CMD action goes to the OST that started BNJDSERV or CNMCSSIR.
- DSTs that provide services such as I/O operations for the user. You can route commands that run as data services commands (TYPE=D) to DSTs.
- Hardcopy task. You cannot route commands to the hardcopy task. Route only messages to this task.

Command Routing Facilities

The primary facilities for routing commands are:

- The automation table ROUTE keyword, for choosing a task when issuing a command from the automation table
- The NetView EXCMD command, for sending a command from one task to another
- The CNMSMSG service routine and the DSIMQS macro, for initiating commands from command processors
- The NetView RMTCMD command, for sending commands to other NetView domains
- Command label prefixes, which route commands in the same manner as RMTCMD and EXCMD

Automation-Table ROUTE Keyword

You can route a command by putting a ROUTE keyword in the automation table with an EXEC(CMD) action. When an incoming message or MSU matches the entry and the NetView program issues a command in response, the command goes to the task or tasks you specify with the ROUTE keyword. If you do not use ROUTE on an EXEC(CMD) action, the NetView program uses the rules explained in Note 6 on page 217 to select a task for the command.

CNMSMSG Service Routine and DSIMQS Macro

You can use the CNMSMSG service routine in PL/I or C and the DSIMQS macro in assembler to send commands to specific tasks, logs, and other destinations.

EXCMD Command

Using the EXCMD command, you can route a command, command list, or command processor to a designated task to be run. Ensure that the command can run under the type of task to which you are routing. For example, data-services command processors can run only under a DST.

In Figure 15, the LOGOFF command is routed to the AUTO1 task, which processes the command. As a result, AUTO1 is logged off.

```
EXCMD AUTO1 LOGOFF
```

Figure 15. EXCMD Command Example

Note: Do not queue commands to run under these server tasks: DSIIPLOG, DSIRXEXC and DSIRSH. These tasks must be free to process TCP/IP requests.

RMTCMD Command

The RMTCMD command sends system, subsystem, and network commands to another NetView program elsewhere in the network. The commands are processed by the other NetView program. Use the RMTCMD command instead of the ROUTE command because the RMTCMD does not require you to start OST-NNT sessions.

Command Label Prefixes

Using ***command label prefixes*** enables you to route commands as you would with the NetView RMTCMD or EXCMD commands, and correlate the responses. Correlation of responses is useful with the CORRCMD pipe stage. For a description of labeled commands, refer to the *IBM Tivoli NetView for z/OS User's Guide: NetView*.

Command Priority

Each of the NetView tasks that process regular commands (autotasks, other OSTs, NNTs, and the PPT) recognize NetView ***command priority*** for queued commands. Queued commands have a priority of either low or high. Priority helps to determine how soon the NetView program runs a command.

You can set the command priority globally with the DEFAULTS command. You can set the priority for a task with the OVERRIDE command and for a single command with the CMD command. Other means of queuing commands have rules for setting the priority.

Command priority affects regular commands issued by an operator, including:

- Operators entering NetView commands from an MVS system console
- Commands relayed by means of the EXCMD command

Command priority does not affect:

- Commands in a command list

  These commands are run sequence rather than being queued.
- Commands that you issue from the automation table

  These commands are always queued at low priority.

Do not use the CMD prefix from the automation table to change the priority to high. When you schedule a command with an AT, EVERY, or AFTER timer command, the DEFAULTS and OVERRIDE settings that apply to the scheduled command are those in effect when the timer expires.

If your automation application queues commands at both low and high priority, be aware that the high-priority commands can run out of sequence before the low-priority commands. Low-priority commands run in order with respect to each other; the first command queued for a task runs first. High priority commands also run in order with respect to each other, except in the case of command procedures.

Command procedures give up control at several points to enable service for the task's high-priority queue; so a high-priority command can interrupt a command procedure, even if the command procedure itself had a high priority. Command procedures enable interruption when running long-running commands and when performing wait or pause processing (for example, a WAIT or PARSE PULL in REXX). In addition, procedures in REXX or the NetView command list language enable interruption immediately upon invocation (before the first instruction) and after each command in the command list.

To process command procedures in the order issued, queue them all at low priority. Command procedures allow interruption by low-priority commands only when processing long-running commands.

For more information about command priority, along with the syntax of the CMD and DEFAULTS commands, refer to the NetView program online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)*; for related information about the OVERRIDE command, refer to the NetView program online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)*.

Part 4. NetView Program Automation Facilities

Chapter 10. Command Lists and Command Processors

To perform complex actions when you issue a single command, use command lists and command processors to create automation procedures.

Command lists are sets of commands and special instructions written in either REXX or the NetView command list language. Command lists written in the NetView command list language are interpreted, and command lists written in REXX can be either interpreted or compiled. *Command processors* are modules written in assembler, PL/I, or C. Command processors (written in PL/I or C) and command lists are also known collectively as *command procedures*. You can issue a command list or command processor as if it were a NetView program command.

Those who can use command lists and command processors to simplify the job of the operator and to assist in automation are:
- Operators
- The automation table
- Timer commands
- The EXCMD command
- Other command lists
- Other command processors

You can also designate initial command lists to be processed during NetView program initialization and OST initialization. These are functions that can be done by command lists and command processors:
- Use a single command to replace a series of queries, replies, and commands normally issued by an operator.
- Issue different replies based on input criteria.
- Ensure consistency among operator responses for lengthy or complex functions.
- Run under an autotask.

Available Languages

The languages available for writing NetView program command lists and command processors are:
- NetView program command list language
- REXX
- PL/I
- C
- Assembler

For a discussion of the capabilities of each language, see the *IBM Tivoli NetView for z/OS Customization Guide*.

Obtaining Messages and MSUs

To automatically issue command procedures when the automation table receives a message or management services unit (MSU), use the NetView program, automation table. When issued in this way, the command procedure has access to information pertaining to the message or MSU that issued the command procedure. When the NetView program receives an MSU over an LU 6.2 transport,

the NetView program can issue a specified command procedure. This command procedure also has access to information for the MSU that was received.

Because the message or MSU information is available to the command procedure, much of the data associated with the message or MSU does not need to be parsed in the automation table statement and then sent explicitly to the command procedure. The attributes for the message or MSU are accessed using various functions in the command procedure. For more information, refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

You can also use the EDIT action in the automation table to make changes to automated messages and MSUs. The changes are made using the syntax and functions provided by the PIPE EDIT stage. For more information about the EDIT specifications, refer to the online help for PIPE EDIT.

Message Functions

The command-procedure languages provide keywords for obtaining access to various message attributes. For example, you can examine the message ID, message type (HDRMTYPE), and message text. For MVS system messages, you can also examine the job name or reply ID.

MSU Functions

Command procedures can also examine and work with MSUs. In REXX, an HIER function gives the hardware monitor resource hierarchy of an alert. An MSUSEG function gives the contents of an MSU, which can include an MDS-MU's header information. The HIER and MSUSEG REXX functions are similar to the HIER and MSUSEG compare items in the automation table, although the syntax details differ. The NetView command list language offers similar &HIER and &MSUSEG control variables. In addition, REXX provides a CODE2TXT function that can translate hardware monitor generic alert code points into the text strings they designate. This function is also available in PL/I and C with the CNMC2T (CNMCODE2TXT) service routine.

Saving Information

You can save information with either global variables or MVS data sets.

Global Variables

Command lists and command processors offer functions that enable you to automate operating procedures. One function is the ability to create and update global variables, which you can use to pass information between command lists, command processors, and the automation table. Global variables are useful in creating automation procedures for purposes such as:

- Maintaining the current status of system and network elements when automation monitors your environment
- Eliminating the need to code system names into automation procedures, which enables you to adapt the procedures to other systems by redefining the global variables rather than by making coding changes in numerous places
- Eliminating the need to code specific parameter values when automating parameter-driven processes, which enables you to change the parameter values without re-coding your command lists and command processors
- Maintaining job names and subsystem commands to be issued as required

The two types of global variables are Task and Common.

Use the QRYGLOBL command to view the number of your common global and task global variables and their values. Refer to the NetView program online help for information about the QRYGLOBL command.

Task Global Variables

Each command list or command processor running under the task can set, inspect, or update a task global variable. Other NetView program tasks do not have direct access to the variables. Therefore, several NetView program tasks can use the same names for task global variables without referring to the same variables. The NetView program gives no indication that two tasks are using the same names.

For a task to inspect or update a task global variable belonging to another task, it must issue a request to the owning task. Therefore, tasks can maintain control of their own variables. Each task has its own task global dictionary for storing task global variables. You can save, restore, and purge task global variables.

Common Global Variables

Any task that can run a command list or a command processor can also use common global variables. One common global dictionary exists for storing all common global variables.

You can save, restore, and purge common global variables. When you save a global variable, NetView places it in a VSAM database. Later, you can restore the variables to the global dictionary from which they were saved. If you no longer need a global variable you have saved, purge it from the database. Saving critical global variables can facilitate recovery from a failure or from a planned outage.

Choosing a Type of Variable

Task global variables are the best choice for data used in a single, local frame of reference. If only one task needs a variable, you can avoid potential naming conflicts with other tasks by using a task global variable. However, use common global variables for information that you want to check or update from more than one task. If you want to pass information to the automation table, common global variables are best, because you do not need to be concerned with which task uses the automation table.

For more information about global variables, refer to the *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*, the *IBM Tivoli NetView for z/OS Programming: Assembler*, and *IBM Tivoli NetView for z/OS Installation: Getting Started*.

For a description of how to read the value of a global variable from the automation table with ATF('DSICGLOB') and ATF('DSITGLOB'), see DSICGLOB “DSICGLOB” on page 163.

You can insert the value of a global variable at the time your table is loaded using Data REXX. For example, the following statement will insert a line to compare the source of a message with the name of the subsystem router task:

```
%> 'IF IFRAUSDR =' CGLOBAL('CNMSTYLE.SSINAME') 'THEN'
```

For more about Data REXX, see *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

MVS Data Sets

Another way of saving data from command lists and command processors is to use a data set. REXX EXECIO and PIPE QSAM can read from and write to sequential data sets. You can use this ability for a wide variety of purposes.

Command processors written in PL/I and C can use high-level language service routines that provide read access to NetView partitioned data sets (CNMMEMO, CNMMEMR, CNMMEMC) and request VSAM I/O (CNMKIO). You can also use PL/I and C I/O services to read from and write to data sets.

Command processors written in assembler can use NetView program macros that provide read access to NetView files (DSIDKS) and request VSAM I/O (DSIZVSMS).

Waiting for a Specific Event

The NetView program enables you to wait for the receipt of messages and other events and to modify processing based on the information received. For best performance, use the CORRWAIT stage of the PIPE command. Refer to *IBM Tivoli NetView for z/OS Programming: Pipes* for more information.

The NetView program also enables you to solicit input from an operator, such as &PAUSE in the NetView command list language, PARSE PULL and PARSE EXT in REXX, and WAIT FOR OPINPUT in high-level languages. However, because autotasks are unattended, avoid using input-soliciting facilities in automation command lists and command processors running under an autotask.

The commands used in waiting for events differ between the languages for command lists and command processors. The differences are described in the following sections. For automation command lists and command processors running under an autotask, try to avoid having the autotasks wait for events. If you use a wait facility, ensure that you specify a time-out value to prevent the autotask from waiting endlessly.

NetView Command List Language Waiting

The basic form of the &WAIT control statement causes a command list to suspend processing until a specified event occurs. The &WAIT control statement is made up of two parts. The first part, which is optional, specifies a command or another command list that is to be processed when the &WAIT statement is reached in the processing of the command list. The second part is a list of event-label pairs that specify where processing is to be transferred when specified events occur. The events you can specify include:

- Receipt of messages that are displayed to the NetView program console
- Receipt of a nonzero return code from the called command or command list
- The expiration of a specified amount of time
- The operator's entry of a GO command

If receipt of a message satisfies the &WAIT statement, use NetView program control variables to obtain the contents of the message.

For more information about waiting for events in the NetView program command list language, refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

REXX Waiting

REXX uses several instructions that interact to provide a method of waiting for messages and analyzing messages and other events. The TRAP instruction specifies messages to be trapped and specifies whether messages that are trapped are displayed to the operator. Messages that are trapped are placed in a message queue, so more than one message can be processed. The WAIT instruction causes a command list to suspend processing until a specified event occurs. Possible events include:

- Messages that you trap
- A time-out value in seconds or minutes
- The operator's entry of a GO command

The MSGREAD instruction causes the NetView program to read a trapped message from the messages currently trapped. The command list can then take action based on the message received. The FLUSHQ instruction is used to discard all trapped messages from the message queue.

For more information about waiting for events in REXX, refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

PL/I and C Waiting

The high-level language application program interface (API) provides several commands and service routines that interact to create a method of waiting for messages and analyzing messages and other events similar to the method used by REXX. The TRAP command specifies messages to be trapped and specifies whether trapped messages are displayed to the operator. Messages that you trap go into a message queue for the command processor, enabling you to work with more than one message. The WAIT command causes a command processor to suspend processing until a specified event occurs. The possible events follow:

- Messages are displayed to the NetView program console.
- The interval set for the time-out value, in seconds or minutes, elapses.
- The operator enters a GO command.
- Data is sent by the CNMSMSG service routine.

The CNMGETD service routine provides access to data queues, one of which is a message queue that contains all messages trapped using the TRAP and WAIT commands. The CNMGETD service routine provides equivalent functions to the REXX MSGREAD and FLUSHQ instructions and other functions.

For more information about waiting for events in high-level language command processors, refer to *IBM Tivoli NetView for z/OS Programming: PL/I and C*.

Additional Command-List Capabilities for MVS

On MVS systems, command lists can send messages to MVS consoles. Command lists can also allocate disk, tape, or print files. Command lists can also save commands and text for later manipulation by operators.

Sending Messages to an MVS Console

To send messages to and remove messages from an MVS console, use these NetView program commands in automation command lists:

WTO Sends a message to an MVS console. For example, you can use the WTO command if operator intervention (such as adding paper to a printer or choosing among processing alternatives) is required.

WTOR Sends a message to an MVS console and requests a reply. Command lists that use WTOR are not completed until the operator replies.

DOM Removes a WTO message from an MVS console. You can use DOM to remove action messages when you know that the action has already been taken.

For more information about these commands, refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

Allocating Disk, Tape, and Print Files

Use the ALLOCATE command with REXX EXECIO or the data set access capabilities of command processors to allocate disk, tape, print files, and the internal reader. These abilities enable you to build JCL from an automated procedure and submit it. For example, if the NetView program receives the message indicating that a system management facilities (SMF) data set is full, define the automation table to pass the SMF data set name to the appropriate command list or command processor. The data set name is embedded in the JCL and the job is submitted to dump the data set using the NetView SUBMIT command.

Note: You cannot allocate a Job Entry Subsystem (JES) data set (internal reader or SYSOUT) if running under a NetView program that started before JES started.

Loading Command Lists into Storage

To promote better performance of your system, you can load command lists into main storage before processing. When you invoke a command list that was not preloaded, it is loaded into main storage, processed, and then dropped from main storage. Therefore, every time the command list is processed, it is retrieved from the auxiliary storage device where it resides. If you preload the command list, it can be processed several times without having to be retrieved from auxiliary storage each time.

These NetView commands move command lists into and out of main storage and identify 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 Identifies command lists that currently reside in main storage.

The NetView program provides a sample command list (CNMS8003) that can help you manage the command lists that have been loaded into storage using the LOADCL command. The sample uses the MAPCL and DROPCL commands to conditionally drop command lists from main storage. You can also use the MEMSTORE command to manage command lists and other NetView program data set members that are loaded into storage.

For more information about these commands, refer to the NetView program online help and to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.

Chapter 11. Timer Commands

In NetView program automation, you can use timer commands to schedule the processing of other commands. Any command or command list that can be issued from a task can be scheduled using a timer command. This chapter describes the timer commands and some related commands.

Overview of Timer Commands

Timer commands inform the NetView program that you want to issue other commands, including command lists and command processors. You can issue timer commands to schedule activities many days in advance or to schedule an activity that takes place once a day or once a month. Use a timer command to schedule another command:

- After the lapse of a specified time
- At a specified time
- Repeatedly at specified intervals

The timer commands are AFTER, AT, EVERY, and CHRON. Two related commands, LIST TIMER and PURGE TIMER, can help you manage command scheduling. AT, EVERY, AFTER, CHRON, and PURGE commands are always echoed to the Canzlog log. In cases where commands are not echoed to log (for example, when issued by a REXX procedure) these command echoes are tagged as TRACE, instead of being tagged as NetView program messages.

This section describes the AFTER, AT, EVERY, and CHRON commands. An operator can issue them directly, or you can use them in other automation facilities, such as command lists and command processors. Refer to the NetView program online help for the syntax and parameter descriptions of these commands.

Note: The AFTER, AT, EVERY, and CHRON commands support customized date and time formats. All examples shown in this chapter assume default formats.

Note: Avoid scheduling interactive commands unless they are to be run on an operator's task with an operator present.

AFTER

The AFTER command enables you to schedule a command or command procedure to run after a specified period of time.

The AFTER command can be useful for waiting a certain amount of time for something that is expected to happen and then checking to ensure that it did happen. For example, if you use the NetView program to initialize a product and the product is to be initialized within 5 minutes, you can schedule a command list to run after 5 minutes to check whether the product started successfully.

The AFTER command, shown in Figure 16 on page 116, schedules the MVS D A,L command to be issued after 5 minutes to solicit status information about system elements.

```
AFTER 00:05:00,ID=DISPSTAT,MVS D A,L
```

Figure 16. Sample AFTER Command

Consider using the AFTER command instead of the DELAY command. When the DELAY command is issued from a command list or command processor, the command list or command processor and the task on which it is executing wait the specified amount of time, thus preventing other work from executing on that task. In contrast, the AFTER command schedules a command and then frees the command list or command processor and the task to do other work during the specified time interval.

AT

The AT command schedules a command or command procedure to be run at a specific time.

For example, the AT command in Figure 17 schedules the STOPSYS command list to shut down the system at 6:00 p.m. on December 24 and saves the command in the Save/Restore database.

```
AT 12/24 18:00:00,ID=EVESAVE,SAVE,STOPSYS
```

Figure 17. Sample AT Command

AT is useful for scheduling commands that you want to happen once, at a specific time or on a specific day.

EVERY

The EVERY command schedules a command or command procedure to be processed repeatedly at a timed interval. The intervals can be specified in seconds, minutes, hours, or days. The command or command procedure is processed at the indicated interval until the EVERY command is purged.

The EVERY command in Figure 18 schedules the command list CHEKSTAT every hour, starting one hour after the timer command is run.

```
EVERY 01:00:00,ID=CHEKST,CHEKSTAT AUTOVTAM
```

Figure 18. Sample EVERY Command

Use an EVERY command similar to the one in Figure 18 to check the status of your autotasks to ensure that they are logged on and are not in a wait condition that prevents other work from executing. The automation sample set provided with the NetView product includes an example of a method for checking on autotasks. This example method uses timer commands as well as the automation table and command lists. The samples are described in Appendix I, "The Sample Set for Automation," on page 577.

TIMER

The TIMER command displays a panel that enables you to display, add, change, test, or delete scheduled timers. The command operates in fullscreen mode only.

CHRON

The CHRON command provides efficient timed command scheduling by decreasing the amount of code in REXX procedures that are used in determining exception cases and time shifts. CHRON also reduces the number of timer

elements by combining criteria that previously required multiple timers or combinations of AT and EVERY commands.

The CHRON EVERY command provides the ability to specify starting times that are earlier than the current time. This is useful for scheduling timed events for multiple days during a shift, and starting the first timer during the shift. This also helps when using CHRON EVERY in a procedure, because the intervals start with the next one in the sequence.

For example, you can schedule a command to be issued on certain days. The CHRON command in Figure 19 issues the LOGTSTAT command once every hour from 8:00 a.m. until 5:00 p.m. on all weekdays except holidays, from now until the last day of the year 2011. The LOGTSTAT command runs on the PPT task. If this CHRON is entered between 8:00 A.M. and 5:00 P.M., LOGTSTAT runs at the next hour. This enables you to specify a shift for following days and have a partial shift run today. This is an example of such a command:

```
CHRON AT=(08:00:00) EVERY=(INTERVAL=(01:00:00 OFF=17:00:00)
      REMOVE=(12/31/11 00:00:00) DAYSWEEK=(WEEKDAY)
      CALENDAR=(NOT HOLIDAY)) COMMAND=LOGTSTAT ROUTE=PPT
```

Figure 19. Sample CHRON Command

Choosing a Task

A scheduled command runs on the same task that issued the timer command, unless you use the primary program operator interface task (PPT) option to specify the PPT. If the task that issued the timer command is no longer active, the scheduled command cannot run. Therefore, it is a good practice to issue timer commands from autotasks. You can do this by using a command-routing facility, such as EXCMD, to send the timer command (AT, EVERY, or AFTER) to an autotask.

By running your scheduled commands on a continuously available autotask, you ensure that the scheduled command is able to run. By using an autotask instead of the PPT, you avoid overburdening the PPT. You also avoid the restrictions about commands that can run on the PPT.

Saving and Restoring Timer Commands

If the NetView program ends, you lose all scheduled timer commands that you have not saved. You can save timer commands in a database to ensure that critical scheduled commands are not lost when you stop and restart NetView. You do not have to re-enter the saved timer commands. You can restore them with the RESTORE command. Issue the RESTORE command after the DSISVRT (Save/Restore) task is activated.

When you issue the RESTORE command, any scheduled command or command list that ran while the NetView program was down results in a multiline message CNM465I. You, or your automation, can use the message to get information about the scheduled command. You can then decide whether to run the scheduled command that was skipped because NetView was down.

Figure 20 on page 118 shows a multiline message you might get for a skipped timer command when you issue RESTORE.

```
CNM465I TIMER EVENT CANNOT BE RESTORED - CURRENT TIME PAST EXECUTION
TYPE: AFTER      TIME: 12/15/10 16:42:17
  COMMAND:  MAJNODES
  OP: OPER1                 ID: AFTMAJ
```

Figure 20. Message Resulting from a Skipped TIMER Command

The message in the sample code contains the following information:

- Line one contains the message ID and text, including the reason that the NetView program cannot restore the timer event.
- Line two gives the type of timer command (AT, EVERY, or AFTER), along with the date and time the command was to run.
- Line three gives the scheduled command.
- Line four gives the ID of the operator who issued the command.

If the operator had used the PPT parameter with the command, lines 2 and 4 indicate that fact as well.

After the DSISVRT task is activated, a command procedure can issue a RESTORE command and wait for CNM465I messages. If any arrive, the command procedure can examine the information in each message to determine whether to reissue the timer command.

LIST TIMER and PURGE TIMER

The LIST TIMER and PURGE TIMER commands can help you manage timer commands. With LIST TIMER, you can display a list of pending timer commands. With PURGE TIMER, you can cancel them. Refer to the NetView program online help for the syntax and parameter descriptions of these commands.

LIST TIMER

LIST TIMER lists all commands and command procedures currently timed for processing, along with associated information. For example, the first command in Figure 21 displays the command or command procedure scheduled by operator OPER1 using AT, EVERY, and AFTER with a timer ID of DISPSTAT (if it exists).

The second command in Figure 21 displays a list of all commands and command procedures scheduled by AT, EVERY, or AFTER on your system regardless of scheduling operator or timer ID.

```
LIST TIMER=DISPSTAT,OP=OPER1

LIST TIMER=ALL,OP=ALL
```

Figure 21. LIST TIMER Command Examples

PURGE TIMER

PURGE TIMER cancels currently scheduled timer commands. For example, the first command in Figure 22 on page 119 purges the command scheduled by OPER1 with a timer ID of DISPSTAT (if it exists).

The second command in Figure 22 on page 119 cancels all AT, EVERY, and AFTER commands scheduled by OPER1. Use all-inclusive purges with caution.

```
PURGE TIMER=DISPSTAT,OP=OPER1

PURGE TIMER=ALL,OP=OPER1
```

Figure 22. PURGE TIMER Command Examples

Chapter 12. Autotasks

Autotasks are a special kind of operator station task (OST) that require neither operators nor NetView terminals. Like other operator OSTs, autotasks can receive messages, process commands and command procedures, and establish NetView program to NetView program sessions. Autotasks can run full screen commands using the NetView program full screen automation function. Because autotasks are not associated with a terminal, they can run when VTAM is not active. For this reason, and because they can perform tasks similar to those that an operator can perform, autotasks are ideal for performing much of your system and network automation.

Defining Autotasks

The requirements for defining autotask IDs are the same as those for defining NetView program operator IDs. Autotasks are OSTs, and you can dynamically define autotask OSTs to the NetView program by editing DSIOPF or system authorization facility (SAF) definitions and then using the REFRESH command.

Sample DSIOPF shows sample definition statements for NetView program OSTs, including autotasks. The statements define each operator's profile. The definition statement for AUTO1, an autotask used in the NetView program initialization process, is shown in Figure 23:

```
AUTO1     OPERATOR      PASSWORD=AUTO1
          PROFILEN      DSIPROFC
```

Figure 23. Definition Statements for AUTO1

The password for an autotask prevents intruders from gaining access to the NetView program by logging on to an autotask operator ID. You can use an SAF product, such as Resource Access Control Facility (RACF), to require a password or password phrase before logging on to an MVS system. If you do not use an SAF product, you can use DSIOPF to define a password for each autotask.

Define a password or password phrase and keep it confidential to protect your autotask IDs. If you are not using an SAF product for password or password phrase checking, you can also prevent someone from logging on to an autotask operator ID by not defining a password in DSIOPF. If you do not define a password, only an AUTOTASK command can start that operator ID. You can then use command authorization on the NetView AUTOTASK command to limit its use.

Activating Autotasks

An autotask is differentiated from other NetView program OSTs by the way an operator starts it. An operator OST starts when a NetView operator logs on at a terminal, but autotasks start when an operator issues the AUTOTASK command. Because either an operator or an autotask can start a single operator ID, it is important to maintain the proper level of security for all IDs defined in the NetView program. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for an explanation of security issues.

You can use the AUTOTASK statement in the CNMSTYLE member to start an autotask when the NetView program initializes. For more information, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Using the AUTOTASK Command

A single primary program operator interface (POI) task (PPT) is started when you start the NetView program. During NetView program initialization, the PPT can start automation tables and AUTOTASKs if they are specified in the CNMSTYLE member. For more information, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

An operator with the proper level of authority can also issue the AUTOTASK command, either at the terminal or with a command list or command processor.

Associating Autotasks with Multiple Console Support Consoles

You can associate an autotask with a multiple console support console when using the AUTOTASK command or AUTOTASK statement in the CNMSTYLE member. This can also be done later after the task is active. Association enables the console to display all messages that the autotask receives and to accept NetView commands and forward them to the autotask.

For example, if you want the autotask AUTOMVS to act as the interface between an MVS system and a NetView system:

1. Determine which MVS console name to use to access the NetView program. In our example, the console name is netvsys2.
2. To associate the autotask with console netvsys2, issue this command:

   ```
   AUTOTASK OPID=AUTOMVS,CONSOLE=netvsys2
   ```

   You can associate the autotask with the console even when the multiple console support console is not online. If the console is not already active, the association is completed when the console is varied online.

If you define an autotask for this purpose and also use the autotask for other automation, remember that all messages sent to the autotask are displayed on the console.

If a write-to-operator (WTO) message comes from an MVS system to a NetView system over the subsystem interface and if you use an associated autotask to route the message back to a multiple console support console, the message appears in the system log and the Canzlog log twice, once in its original format and once as the NetView program sent it to the multiple console support console. To avoid duplication, define dedicated autotasks that you use for multiple console support consoles only.

For more information about the AUTOTASK command, see the NetView program online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)*.

Deactivating Autotasks

You can deactivate an autotask with one of these commands:

- `EXCMD autoid,LOGOFF`
- `%LOGOFF` (issued from a multiple console support console associated with the autotask)

Here % is the default NetView program subsystem descriptor. The subsystem address space must be active for this command to work.

Note: Any command entered on the multiple console support console and prefixed by the descriptor automatically restarts the autotask, unless you use the AUTOTASK command to drop the console association.

If an autotask is stuck in an infinite loop, issue `EXCMD autoid,RESET` to stop the command list that is running before attempting a logoff. If necessary, you can also use `STOP FORCE=autoid` to deactivate an autotask that is in an infinite loop. The EXCMD and %LOGOFF commands simply queue the LOGOFF command under the autotask along with other queued command lists and commands. STOP FORCE is an immediate command.

Automating with Autotasks

This section describes some of the many ways you can use autotasks for automation.

Managing Subsystems

Because they do not depend on the VTAM program, autotasks are useful when the system is running without VTAM. For example, when the NetView program initializes, you can start an autotask and have it manage the subsystems, including VTAM. The autotask can help VTAM activate or recover from failure, as appropriate. Keep in mind that although autotasks are not associated with an operator console, they still require APPL statements in the VTAM definition, and they can issue commands to VTAM.

Attention: If the NetView program is started before VTAM, any autotasks started while VTAM is inactive are assigned a specific VTAM application identifier (APPLID) using the hexadecimal numbering scheme. Because the NetView program does not know whether the assigned APPLID is available when VTAM is started, it must assume that the APPLID is available for use. Therefore, you must define consecutively numbered VTAM APPL statements for each of these autotasks. Numbering uses the hexadecimal scheme, starting after those reserved by any POS terminals. For example, if 12 POS terminals have been defined, and 6 autotasks are started before VTAM is started, and your domain name is CNM01, you must define APPL names CNM0100C, CNM0100D, CNM0100E, CNM0100F, CNM01010, and CNM01011 for these autotasks.

Processing Unsolicited Messages

Autotasks can process all of your unsolicited messages and the commands you issue in response. This approach has two advantages related to processing messages:

- Does not depend on a specific user being logged on
- Processing can be faster

For example, if an operator is executing a long-running command and receives an unsolicited message, the command that the operator issues in response to the message is queued until the long-running command ends. If autotasks receive an unsolicited message, the command in response runs immediately.

To ensure that your autotasks are continually available, you can have automation monitor the autotasks. The advanced automation sample set demonstrates one technique for monitoring autotasks.

See "Using the Advanced Automation Sample Set" on page 585 for more information.

Processing Commands

An autotask can process commands and command procedures sent by the automation table. If you use the ROUTE keyword to explicitly choose a destination for a command, you can use an autotask. A command might also go to an autotask through default routing if you do not use the ROUTE keyword. This is the case, for example, if the autotask solicited the message that is triggering the command.

An autotask can process commands and command procedures that are scheduled under it by an AFTER, AT, CHRON, or EVERY command. You can define and start several autotasks to monitor different resources or types of resources. Each autotask can then use different time intervals for monitoring and different collections of task global variables for storing information.

A NetView command procedure can wait for the receipt of a message or another event before continuing processing. This is referred to as WAIT processing. Use caution when running command procedures containing WAIT processing under an autotask. If you must run such a command procedure under an autotask, ensure that you specify a timeout value for the WAIT command within the procedure. In addition, you might want to limit the autotasks that run such command procedures.

For more information about WAIT processing, see "Waiting for a Specific Event" on page 112.

Starting Tasks

Consider starting the BNJDSERV task and the CNMCSSIR task from autotasks. Table 3 shows the destination of commands when messages or MSUs are automated in the NetView automation table and you do not use a ROUTE keyword or you specify a destination of *.

Table 3. Command Destinations When Using Autotasks to Start Tasks

Command in response to:	Goes to:
Unsolicited subsystem interface message	Task that started the CNMCSSIR task
MSU from BNJDSERV	Task that started the BNJDSERV task

By having an autotask start these tasks, you can ensure that a task is ready to process such commands and messages.

To define an autotask for a specific function during NetView program initialization, use the function.autotask statement in the CNMSTYLE member. For more information, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Sending Commands to an Autotask Using the EXCMD Command

Other tasks can use the EXCMD command to send commands to an autotask.

- Operators can use autotasks to perform work that might otherwise require time on their OSTs.
- Autotasks can send commands to each other to perform work that logically requires serial processing.
- You can send slow commands to an autotask to avoid interfering with the throughput or response time of tasks that are performing more critical activities.
- An autotask can process commands that are sent to it by other operators using the EXCMD command.

Used this way, an autotask creates a kind of background processor to support work that:

- Logically requires serial processing under a single task
- Might interfere with more critical operator tasks

However, the NetView program does not automatically return the resulting messages to the originating operator.

Chapter 13. The Message Revision Table

This chapter describes:
- The NetView message revision table (MRT)
- The statements you can use in a message revision table
- How to code a message revision table
- Message revision table statements
- Example of a message revision table listing
- Usage reports for message revision tables
- Managing multiple message revision tables

For information on testing the logic of the message revision table, see "Message Revision Table Testing" on page 133.

The WHEN and REVISE statements consist of PIPE EDIT orders. See the NetView program online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for information about how to use these edit orders.

What Is the Message Revision Table?

The message revision table (MRT) enables you to intercept MVS messages before they are displayed, logged, automated, or routed through your sysplex. You can make decisions about the message based on its message ID, job name, and many other properties.

You can make changes to many aspects of the message, including these:
- Message text
- Color
- Route codes
- Descriptor codes
- Display and system log attributes

The MRT can remain active even while the NetView program is not active, but the SSI address space is required. However, loading or querying the MRT, or gathering statistics, depends on the functional NetView program address space being active.

Elements of Message Revision Table Statements

A message revision automation table contains these elements:
- Use the *DoForeignFrom statement* ("DoForeignFrom Statement" on page 129) to indicate that foreign messages are to be processed by the MRT.
- Use the *END statement* ("END Statement" on page 129) to close a section started with a SELECT statement.
- Use the *EXIT statement* ("EXIT Statement" on page 130) to stop any further message revision when an action is matched.
- Use the *NETVONLY statement* ("NETVONLY Statement" on page 130) to provide for NetView program automation, but suppress display, logging, and sysplex routing.
- Use the *OTHERWISE statement* ("OTHERWISE Statement" on page 130) to provide for NetView program automation, but suppress display, logging, and sysplex routing.

- Use the *REVISE statement* (“REVISE Statement” on page 130) to include revision actions.
- Use the *SELECT statement* (“SELECT Statement” on page 131) to introduce a series of WHEN statements.
- Use the *UPON statement* (“UPON Statement” on page 131) to introduce each section.
- Use the *WHEN statement* (“WHEN Statement” on page 132) or the *OTHERWISE statement* to specify a condition.
- Use the *%INCLUDE statement* (“%INCLUDE Statement” on page 228) to include separately coded and maintained sections of the message-revision table to divide your message-revision table maintenance among several groups or individuals. You can view your INCLUDE structure using the automation-table management function (AUTOMAN). See sample CNMSMRT1 for additional information on this function.

Message Revision Table Processing

You can use the REVISE MSG command to activate, deactivate, test, list, or check the status of a message revision table. See the NetView program online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for more information about using the REVISE MSG command.

Message Revision Table Searches

When an MVS message is issued, the NetView program SSI code employs a fast search algorithm to locate the particular UPON statement that is relevant for that message. Conditions and actions under that UPON are then applied sequentially. If a message matches no particular UPON condition, this is quickly determined and the message is then subject to conditions and actions under the UPON(OTHERMSG) condition, if any.

You can include an UPON statement with no subordinate conditions or actions, simply to cause your MRT report to contain a count of matching messages. An UPON statement with no subordinate conditions or actions is called a *null UPON*.

Conditions subordinate to an UPON statement, including UPON(OTHERMSG), are examined sequentially. Therefore, you might improve performance by including a specific null UPON statement to match common messages, to prevent their being examined by the UPON(OTHERMSG) conditions.

Coding a Message Revision Table

These directions and restrictions apply to coding the message revision table:

- Comments can begin with an asterisk (*) in column 1 or following an exclamation point (!) anywhere in the file.
- You can use blanks to indent lines and to separate keywords, logical operators, and parentheses.

  However, blanks used within a comparison string are considered characters in that string.
- You must use single or double quotation marks as the delimiters for comparison text and for synonym values. If a literal has one kind of quotation mark, use the other as the delimiter.
- You can include actions such as REVISE and NETVONLY directly under an UPON statement without any SELECT, WHEN, or OTHERWISE statement. Such actions are labeled as type OTHERWISE in an MRT report.

- WHEN and REVISE statements consist of PIPE EDIT orders. See the NetView program online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for information about how to use these edit orders.
- If the MVS Message Processing Facility (MPF) has suppressed a message or has disabled system logging, then the DISPLAY order and the SYSLOG order can neither detect nor override the suppression. This is a z/OS system limitation. To overcome this restriction, use a REVISE statement to obscure the message ID, and then use NETVONLY and NetView program message automation to reissue the disguised message.
- If any message is unnecessarily transmitted across your sysplex, you can improve performance dramatically by using the NETVONLY and REVISE('N' DELETE) MRT actions to prevent such a transmission.

Changing Route Codes and Descriptor Codes

There are 16 routing code FLG orders named FLGRTCD1 through FLGRTCD16 that correspond to the 16 bytes of extended routing codes defined in the WQE. There are four descriptor code FLG orders named FLGDSCD1 through FLGDSCD4.

ROUTEZERO can be used to set all route codes to zero. Here are some examples:

```
* zero out all route codes and set route codes 8 and 16
REVISE (ROUTEZERO "xxxxxxx1" FLGRTCD1 "00000001"   FLGRTCD2)
* set descriptor code 2 meaning immediate action required and retain message
* in AMRF
REVISE ("x1xxxxxx" FLGDSCD1 'Y' AMRF)
```

DoForeignFrom Statement

Foreign messages are not processed by the MRT by default (see "How Foreign Messages are Processed" on page 75 for additional detail on this topic). The DoForeignFrom statement can be used to indicate that foreign messages are to be processed by the MRT. When specified, the DoForeignFrom statement must occur prior to any UPON statement. The format of the DoForeignFrom statement is as follows:

```
DoForeignFrom = *ALL | *NONE
```

- When DoForeignFrom is set to *ALL, the MRT processes foreign messages that originated at any other system in the sysplex (in addition to messages that originated at the local system). MRT processing can be limited to specific system names using the SYSNAME edit order on the WHEN statement. The value of AUTOMATE can be set to Y using a REVISE statement which causes the message to be sent to the NetView program address space.
- When DoForeignFrom is set to *NONE, the MRT processes only those messages that originated at the local system. Note that the NetView program SSI, and therefore the MRT, does not receive foreign messages if they are disallowed by the FORNSSI statement in the MPFLSTxx MVS PARMLIB member.

END Statement

An END statement closes a section started with the corresponding SELECT statement.

EXIT Statement

Typically, a message is matched against everything in a given UPON group (counting each SELECT-WHEN-END entry as one item). When an EXIT action is matched, however, the remaining actions under the same WHEN or OTHERWISE are performed, but subsequent SELECT statements in that UPON group are bypassed.

NETVONLY Statement

The NETVONLY statement causes the message to be marked for suppression from display, but allows the message to be sent to the NetView program. When the NETVONLY statement is received in the NetView program address space, the message is either submitted to an automation task or is routed as a command response. The NETVONLY statement always queues messages to the NetView program over the SSI.

OTHERWISE Statement

An OTHERWISE statement is like a WHEN statement, except that there is no condition and it must follow all the WHEN statements under a given SELECT statement.

REVISE Statement

A REVISE statement is followed by a set of parentheses enclosing a single edit script. Such a script is called a *revision script*. For this case only (not for WHEN, EDIT, or ACQUIRE conditions, for example), an exception is made for text handling: if no output order changes the message text, then the entire text is replicated into the output message (in other scripts, this results in null text). Multiple REVISE statements can be in any group, with each acting on the result of the previous revision. If a subsequent SELECT group reexamines the message, it sees the result of the action of the preceding REVISE action.

Examples of the REVISE statement:

```
REVISE("Cr" COLOR)                          * turn msg red
REVISE("CY" COLOR  "ABCDEFGH" autotoken)    * turn msg yellow and set autotoken
REVISE(ROUTEZERO)                           * set all routecodes to false/zero
REVISE('1xxxxxxx' FLGDSCD2)                 * set descriptor code 9 to true
REVISE('1xxxxxx0' FLGDSCD3)                 * set descriptor code 17 to true
                                            * and descriptor code 24 to false

REVISE('1xxxxxxx' FLGRTCD4)                 * set route code 25 to true
REVISE("cr hr" color )                      * turn msg color to red with reverse video
REVISE("ct hu" color )                      * turn msg color to turquoise and underline
REVISE ('N' automate)                       * do not automate this message
REVISE ('Y' automate)                       * automate this message
REVISE ('Y' AMRF)                           * retain Action message in AMRF
REVISE ('N' DISPLAY)                        * do not show message at the console
REVISE ('Y' DISPLAY)                        * show message at the console
REVISE ('Y' BROADCAST)                      * send message to all active consoles
REVISE ('Y' PROG)                           * for programmer information (route code 11)
REVISE ('N' SYSLOG)                         * do not write this message to the system log
REVISE ('80'x SYSLOG)                     * write this message to the system log
REVISE ('Y' DELETE)                         * totally delete message

* turn msg blue and append "SHOULD BE BLUE" to the end of message
REVISE("CB" COLOR 1.* 1 "SHOULD BE BLUE" NW)
* same as previous example except using ALL
```

```
REVISE("CB" COLOR ALL 1 "SHOULD BE BLUE" NW)
REVISE ( ALL UPCASE )                    * UPPER case the entire message
revise (MSGID 1 "changed message" NW)    * Keep msgid and append "changed message"
REVISE("0000000x" FLGRTCD1)              * turn off routecodes 1-7 leave 8 as before
REVISE ( "WHOKNOWS" CONSNAME)            * send message to console with the name WHOKNOWS
```

See the NetView program online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for information about using PIPE EDIT orders.

Note: MVS imposes a limit of 127 characters in text output. The MRT does not provide a warning or condition when longer messages are truncated.

SELECT Statement

A SELECT statement introduces a series of WHEN statements, followed by a required OTHERWISE statement and an END statement. The SELECT statement does not include any arguments.

UPON Statement

An UPON statement is a top-level conditional that introduces each section. There are four types of conditions:

- MSGID, which can be in the range of 1 - 12 characters
- JOBNAME, which can be in the range of 1 - 8 characters
- PREFIX, which is always three characters
- OTHERMSG

They are tested in the order provided here and the first three conditions are always compared with a literal. The MSGID literal can be in the range of 1 - 12 characters, the JOBNAME literal can be in the range of 1 - 8 characters, and the PREFIX literal is always three characters.

These are examples of the conditions:

```
UPON (msgid = 'CNM233I' | JOBNAME='VTAM' | prefix = 'IST')
UPON (MSGID="TST102A" |
      MSGID="TST102B" |
      MSGID="TST102C" |
      MSGID="TST102D" )
```

If any message matches one type of an UPON statement, the message is not compared with lower-ranking UPON statements. For example, if message CNM233I is presented to the preceding table, it is not compared for the JOBNAME or PREFIX conditions and it is not submitted to any statements listed under the UPON(OTHERMSG) section. Note that MSGID is tested first even if that UPON statement is not physically first in the table definition.

Limit your use of UPON(OTHERMSG) statements to avoid performance degradation.

Within a given UPON statement, multiple conditions can be joined by an OR symbol (|), but not AND.

Subordinate to each UPON statement, there can be zero or more statements of type SELECT, REVISE, NETVONLY, and EXIT. This group of statements is called an *UPON group* and it is evaluated in the same order that it is specified.

When a message has matched an UPON statement and acted on by the UPON group, no further action is taken by the MRT. In particular, such a message is not compared with other UPON conditions.

Note that only the first line of an MLWTO message is examined.

WHEN Statement

The WHEN statement is subordinate to a SELECT statement and is always followed by an expression enclosed in parentheses. Each WHEN statement is followed by a set of zero or more action statements preceding the next WHEN or OTHERWISE statement. This is called a *WHEN group*. The expression is a pair of edit scripts separated by either an *equal* sign (=) or a *not equal* set of symbols (¬=). The two scripts are run against a message and the results are compared, after the leading and trailing blanks or null values are removed. If the two are equal (or not equal, depending on the separator value), then the message is considered to have matched that WHEN statement. Such a message is acted upon by the action statements of the WHEN group and is not compared with other WHEN statements under the same SELECT statement, and it is not matched to the OTHERWISE statement.

Examples of the WHEN statement:

```
WHEN (msgid right 1 = 'A')            * when action message
WHEN (MSGID SUBSTR 5.* RIGHT 1 = 'E') * when error message
WHEN (SYSLOG yesno = 'Yes')           * when SYSLOG is on. 'Yes' is case sensitive.
WHEN (SYSLOG yesno = 'No')            * when SYSLOG is off. 'No' is case senstive.
WHEN (SYSLOG = '80'x)                 * when SYSLOG is on
WHEN (SYSLOG = '00'x)                 * when SYSLOG is off
WHEN (FLGRTCD1 SUBSTR 2.1 = '1')      * when routing code 2 is on
```

See the NetView program online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for information about using PIPE EDIT orders.

Example of a Message Revision Table

This is an example of an MRT:

```
UPON ( MSGID = 'IEA404A'      ! SEVERE WTO BUFFER SHORTAGE - 100% FULL
     | MSGID = 'IRA200E'      ! AUXILIARY STORAGE SHORTAGE
     | msgID = 'DSI125I')     !  CRITICAL STORAGE SHORTAGE FOR NCCF

    REVISE('CR HR' COLOR)     ! make msgs red/reverse
* note, when adding to text, be sure to put text in there first!
    SELECT
      WHEN (MSGID = 'DSI125I')     !
           REVISE("N" AUTOMATE)    ! do not try to automate dsi125
           EXIT                    ! skip further revision, too.
      OTHERWISE
    END
    REVISE('11xx0xxx' FLGRTCD1    ! send to Rt Cd 1,2 but not 4 ...
          1.* 1 "919-555-5677")   ! and add my phone number to text

* Some VTAM related messages ...
 UPON (MSGid = 'CNM233I' |MSGID = 'CNM234I' |MSGID = 'CNM235I'
     | MSGID = 'CNM385I' |MSgID = 'CNM386I' |MSGID = 'CNM435I'
     |MSGID = 'CNM439I'
     | JOBNAME = 'VTAM' | preFix = 'IST')
    SELECT
      WHEN (MSGID LEFT 3 = 'CNM')   ! like "prefix" for WHEN statment
         REVISE("CP" COLOR)         ! nv msgs above turn pink
      OTHERWISE
```

```
        REVISE("CP HR" COLOR)      ! others also underscored
        NETVONLY                   ! sys consoles not to see these
   END
 UPON (PREFIX = 'DSI' | prefix = 'CNM' | prefix = 'DWO')
     revise ('xxxx11xx' flgRtCd1)  ! in addition to route codes already
                                   ! set, add 5 and 6

  ! Despite being specified first, the prefix condition above is
  ! evaluated AFTER all MSGID conditions.  Due to the following,
  ! DSI802A & 803A are not affected by the Rt Cd 5 & 6  revision.
 UPON (MSGID = 'DSI802A'           ! changing text of these msgs
      | MsgID = 'DSI803A')
     revise(w1 1 msgid nw          ! put in reply ID and msgid
            "reply CLOSE or MSG" nw)! list valid cmds
! note: above revision sets text, so text placed by the edit is the
!       ONLY text in the resulting message.
    SELECT
      WHEN(MSGID RIGHT 2 = '3A')    ! for one of the above MSGIDs,...
       REVISE(1.* 1 'ONLY!')      ! make addtional text changes
      OTHERWISE
    END
    SELECT
      WHEN(W3 ¬= '&DOMAIN.')        ! msg from other NetView?
         NETVONLY                   ! steal msg from MVS, give only
                                    ! to this NetView, for automation
   ! Be sure your automation does something with these msgs!
      OTHERWISE
    END
 UPON(OTHERMSG)          ! more performance cost for these tests...
   SELECT
     WHEN (MSGiD RIGHT 1 = 'A')     ! action msg
         REVISE("HB"  COLOR)        ! keep same color, add blink
     WHEN (MSgid RIGHT 1 = 'E')     !
        REVISE('xx1xxxxx' FLGRTCD2  ! add Rt Cd 11 to any present
               'CY'  color)         ! and color
     OTHERWISE
   END
```

Usage Reports for Message Revision Tables

You can use the REVISE MSG REPORT command to gather statistics and usage information about the active revision table. If successful, a BNHRVaaaI message is issued. When the REPORT keyword is specified with the MEMBER operand, the information displayed is about the table being replaced and the time it was replaced.

See the NetView program online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for more information about using the REVISE MSG command.

Message Revision Table Testing

You can use these steps to test your message revision table and verify that the route codes, descriptor codes, and console name specified on a REVISE statement are working as expected:

1. Issue a message using the WTO command. For example:

   ```
   WTO TST125A First test message
   ```

2. Allow the message to go through your message revision table and modify the console name, a descriptor code, and a route code.
3. Use the NetView program automation table to call a REXX routine to print the descriptor code, route code, and console name information. This is an example

of an automation table entry, a section of the Message Revision Table, and a REXX example that the Automation Table entry calls:

- Automation table entry

```
IF MSGID = 'TST' .  THEN
   HOLD(N) EXEC(CMD('RexxExec') ROUTE(ONE ConsoleName));   * where ConsoleName
   *is the NetView console name obtained using GETCONID
```

- A section of the Message Revision Table

```
UPON (MSGID="TST125A")
SELECT
When (MSGID="TST125A")
REVISE('1xxxxxxx' FLGDSCD1)
REVISE('xxxxxxx1' FLGRTCD2)
REVISE ("ConsoleName"  CONSNAME)   * where ConsoleName is a valid Console Name
Otherwise
END
```

- REXX example called by the Automation table entry

```
/* RexxExec */
say ROUTCDE()
say "desc code =" DESC()
say "CONSNAME=" SYSCONID()
exit
```

See the NetView program online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for more information about the WTO command.

Chapter 14. The Command Revision Table

This chapter describes:
- The NetView command revision table (CRT)
- How to code a command revision table
- Command revision table statements
- Example of a command revision table listing
- Usage reports for command revision tables
- Testing the logic of the command revision table

Note: The MVS Command Revision function replaces the existing MVS Command Management function. For information on migrating to the MVS Command Revision function, see the *IBM Tivoli NetView for z/OS Installation: Migration Guide*.

What Is the Command Revision Table?

The command revision table (CRT) enables you to intercept MVS commands before they are processed. Command sources include the MVS console and the NetView MVS command.

The CRT intercepts any text entered on an MVS console command line as an SDSF system command, using the JCL COMMAND parameter, or by any program using the MGCRE macro or direct SVC 34. The text entered might or might not be a valid MVS command before being altered or redirected by CRT processing. However, you can use the REISSUE command as part of the CRT processing and the subsequent command is exempt from CRT action. For information on the NETVONLY statement, see "NETVONLY Statement" on page 141; for information on the REISSUE statement, see the online help. You can make decisions about the command based on its source, command verb, and command parameters.

You can make changes to the command text, write a message to the command issuer, and then run the command or suppress the command. You can also transfer the command to the NetView program for more involved actions. The CRT can remain active even while the NetView program is not, but the SSI address space is required. However, loading or querying the CRT, or gathering statistics, depends on the functional NetView address space being active.

Elements of Command Revision Table Statements

A command revision automation table contains these elements:
- Use the *UPON statement* to introduce each section.
- Use the *SELECT statement* to introduce a series of WHEN statements.
- Use the *WHEN statement* or the *OTHERWISE statement* to specify a condition.
- Use the *END statement* to close a section started with a SELECT statement.
- Use the *REVISE statement* to include revision actions.
- Use the *EXIT statement* to stop any further command revision when an action is matched.
- Use the *NETVONLY statement* to provide for NetView automation.

- Use the *WTO statement* to generate a message to the console from which the command was issued. If the command originated from an INTERNAL, INSTREAM, INTIDS, or HC console, the message is written to SYSLOG only.
- Use the *%INCLUDE statement* to include separately coded and maintained sections of the command-revision table to divide your command-revision table maintenance among several groups or individuals. You can view your INCLUDE structure using the automation-table management function (AUTOMAN). See sample CNMSCRT1 for additional information.

Command Revision Table Processing

You can use the REVISE CMD command to activate, deactivate, test, list, or check the status of a command revision table. See the NetView online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for more information about using the REVISE CMD command.

Command Revision Table Searches

When an MVS command is issued, the NetView SSI code employs a fast search algorithm to locate the particular UPON statement that is relevant for that command. Conditions and actions under that UPON are then applied sequentially. If a command matches no particular UPON condition, this is quickly determined and the command is then subject to conditions and actions under the UPON(OTHERCMDS) condition and then the UPON(ALLCMDS) condition.

You can include an UPON statement with no subordinate conditions or actions, simply to cause your CRT report to contain a count of matching commands. An UPON statement with no subordinate conditions or actions is called a *null UPON*.

Conditions subordinate to an UPON statement are examined sequentially. Therefore, you might improve performance by including a specific null UPON statement to match common commands, to prevent their being examined by the UPON(OTHERCMDS) or UPON(ALLCMDS) conditions.

Coding a Command Revision Table

These directions and restrictions apply to coding the command revision table:

- Comments can begin with an asterisk (*) in column 1 or following an exclamation point (!) anywhere in the file.
- You can use blanks to indent lines and to separate keywords, logical operators, and parentheses.

  Blanks used within a comparison string are considered characters in that string.
- You must use single or double quotation marks as the delimiters for comparison text and for synonym values. If a literal has one kind of quotation mark, use the other as the delimiter.
- You can include actions such as REVISE, NETVONLY, and WTO directly under an UPON statement without any SELECT, WHEN, or OTHERWISE statement. These actions are labeled as type OTHERWISE in a CRT report.
- PIPE EDIT orders can be used with the WHEN, REVISE, and WTO statements. See “Edit Orders” on page 142 for more information.

Command Revision Table Statements

You can use the following statements in a CRT:

- "TRACKING.ECHO Statement"
- "ISSUE.IEE295I Statement"
- "UPON Statement" on page 138
- "SELECT Statement" on page 139
- "WHEN Statement" on page 139
- "OTHERWISE Statement" on page 140
- "END Statement" on page 140
- "REVISE Statement" on page 140
- "NETVONLY Statement" on page 141
- "WTO Statement" on page 141

TRACKING.ECHO Statement

By default, the z/OS operating system issues an additional command echo when the CRT makes a change to a command. This appears in your system log. This new message reflects the command as it was changed by the CRT.

Notes:

1. System APAR OA28464 is required for this statement to be in effect.
2. If specified, the TRACKING.ECHO statement must precede any UPON statement.

The TRACKING.ECHO statement uses the following syntax:

TRACKING.ECHO

Parameter	Description
YES	Specify YES to allow the extra echo. This is the default.
NO	Specify NO to suppress the extra echo.

ISSUE.IEE295I Statement

By default, the z/OS operating system issues an additional IEE295I, a multi-line message, to document changes made directly in your CRT. This appears in your system log. The message is not issued when you specify the NETVONLY action. This new message reflects the command as it was prior to and after the changes made by the CRT.

Notes:

1. System APAR OA28464 is required for this statement to be in effect.
2. If specified, the ISSUE.IEE295I statement must precede any UPON statement.

The ISSUE.IEE295I statement uses the following syntax:

ISSUE.IEE295I

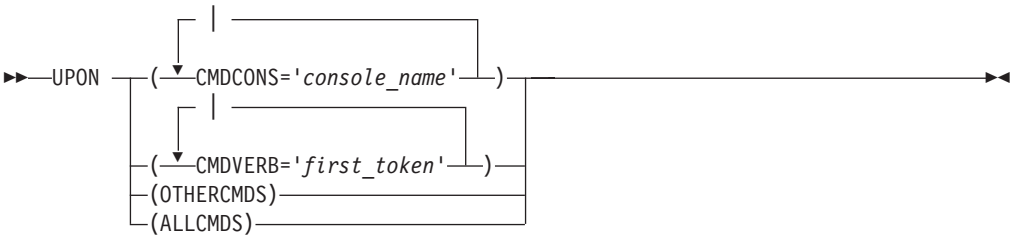
Parameter	Description
YES	Specify YES to allow the tracking message. This is the default.
NO	Specify NO to suppress the tracking message.

UPON Statement

An UPON statement is a top-level conditional statement that introduces each section.

The UPON statement uses the following syntax:

UPON



Parameter	Description
CMDCONS='*console_name***'**	Name of the console issuing the command.
CMDVERB='*first_token***'**	Value of the first token delimited by a blank or comma, which can be in the range of 1 - 12 characters. Command synonyms are not resolved.
OTHERCMDS	All commands not matched by the preceding conditions.
ALLCMDS	All commands.

Usage Notes:

1. The CMDCONS and CMDVERB conditions can accept multiple values or can be coded multiple times with different values.
2. Code the OTHERCMDS and ALLCMDS conditions only once in a table.
3. The parameters are examined in the following order:
   a. CMDCONS
   b. CMDVERB
   c. OTHERCMDS
   d. ALLCMDS
4. If any command matches one type of an UPON statement, the command is not compared with lower-ranking UPON statements.
5. Limit your use of UPON(OTHERCMDS) and UPON(ALLCMDS) statements to avoid performance degradation.
6. Within a given UPON statement, multiple conditions can be joined by an OR symbol (|), but not AND.

7. Subordinate to each UPON statement, there can be zero or more statements of type SELECT, REVISE, NETVONLY, and WTO. This group of statements is called an *UPON group* and it is evaluated in the same order that it is specified.
8. OTHERCMD is a synonym for OTHERCMDS. ALLCMD is a synonym for ALLCMDS.

Examples:

These are examples of the UPON conditions:

```
UPON (CMDVERB ='CONTROL' | CMDVERB='K' | CMDCONS='SAMSOWN')
UPON(CMDVERB='FORCE')
UPON(OTHERCMDS)
```

SELECT Statement

A SELECT statement introduces a series of WHEN statements, followed by a required OTHERWISE statement and an END statement. The SELECT statement does not include any arguments.

The SELECT statement uses the following syntax:

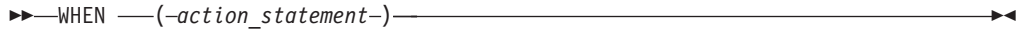
SELECT

WHEN Statement

The WHEN statement is subordinate to a SELECT statement and is always followed by an expression enclosed in parentheses. Each WHEN statement is followed by a set of zero or more action statements preceding the next WHEN or OTHERWISE statement. This is called a *WHEN group*. The expression is a pair of edit scripts separated by either an *equal* sign (=) or a *not equal* set of symbols (¬=). The two scripts are run against a command and the results are compared, after the leading and trailing blanks or null values are removed. If the two are equal (or not equal, depending on the separator value), then the command is considered to have matched that WHEN statement. Such a command is acted upon by the action statements of the WHEN group and is not compared with other WHEN statements under the same SELECT statement, and it is not matched to the OTHERWISE statement.

The WHEN statement uses the following syntax:

WHEN



Parameter	Description
action_statement	Edit orders that you can specify. See Table 4 on page 142 for a list of the edit orders that you can specify.

Examples:

These are examples of the WHEN statement:

```
WHEN(CONSAUTH = 'I')           ! console's authority is "I/O"
WHEN(CONSNAME left 3 = 'MST')  ! console's name begins with "MST"
WHEN(WORD 2 = '')              ! command entered with no arguments
WHEN(ASTYPE = 'D')             ! command from USS persistent procedure
WHEN(CMDVERB = 'SWITCH')       ! switch command issued  -- or ....
WHEN(CMDVERB = 'I')            ! switch command issued
```

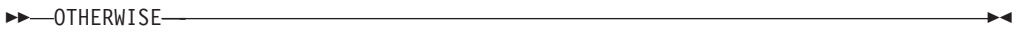
See the NetView online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for information about using PIPE EDIT orders.

OTHERWISE Statement

An OTHERWISE statement is like a WHEN statement, except that there is no condition and it must follow all the WHEN statements under a given SELECT statement.

The OTHERWISE statement uses the following syntax:

OTHERWISE

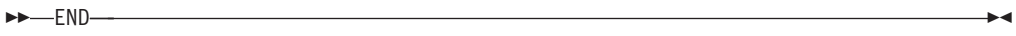


END Statement

An END statement closes a section started with the corresponding SELECT statement.

The END statement uses the following syntax:

END



REVISE Statement

The REVISE statement modifies the command string (text).

The REVISE statement uses the following syntax:

REVISE

Parameter	Description
action_statement	Edit orders that you can specify. See Table 4 on page 142 for a list of the edit orders that you can specify.

Usage Notes:

1. A REVISE statement is followed by a set of parentheses enclosing a single edit script. This script is called a *revision script*. If no output order changes the command text, the entire text is replicated into the output command.
2. Multiple REVISE statements can be in any group, with each acting on the result of the previous revision. If a subsequent SELECT group reexamines the command, it sees the result of the action of the preceding REVISE action.

Restriction:

MVS imposes a limit of 126 characters per command. The CRT does not provide a warning or condition when longer commands are truncated.

Examples:

These are examples of the REVISE statement:

```
REVISE(ALL 1 ",AREA=BLD410" N)  ! adding a parameter to a command
REVISE('Y' DELETE)              !  command is deleted (forbidden)
REVISE(ALL 1 ",L" NEXT )        ! adding a parameter
```

See the NetView online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for information about using PIPE EDIT orders.

NETVONLY Statement

The NETVONLY statement specifies the REXX procedure (command) that is to be run in the NetView address space. The parameters of the procedure become the command text submitted to the CRT and includes any revisions made in the CRT prior to the NETVONLY action.

The NETVONLY statement uses the following syntax:

NETVONLY

Parameter	Description
procedure	1 to 8 character REXX procedure name

Usage Notes:

1. When the *procedure* (command) is received by the NetView address space, it is submitted to the command revision environment automation task. The automation task is identified by the ?MVSCmdRevision statement in the CNMSTYLE member. The NETVONLY statement queues commands to the NetView program over the SSI.
2. Only one NETVONLY statement can be specified in each WHEN or OTHERWISE statement.
3. Review the CNMSRVMC sample for example coding techniques.

WTO Statement

The WTO statement creates text for a WTO message that is sent to the console from which the command was issued.

The WTO statement uses the following syntax:

WTO

Parameter	Description

action_statement
Edit orders that you can specify. See Table 4 for a list of the edit orders that you can specify.

Restrictions:

1. You cannot set route codes, descriptor codes, or other parameters of the WTO.
2. Text exceeding 126 characters is truncated.

Edit Orders

Table 4 lists the edit orders that you can use with the WHEN or REVISE specifications.

Table 4. Edit Orders

Edit Order	Description
ALL	Indicates to use the entire text of the command. This is the same as "1.*".
ASID	Indicates the address space ID of the MVS originator of the command (2-byte binary value).
ASTYPE	Indicates how the address space was started (job type): **Value** **Description** **D** USS persistent procedure. The address space has a name for initiated programs, appropriate for a JOB. However, the existence of an OpenMVS address space block indicates a special purpose USS persistent procedure. **J** The address space is a JOB. **N** The address space is a system address space started during operating system initialization (NIP) processing. **S** The address space is a Started Task (STC). **T** The address space is a Time-Sharing User (TSO). **U** The address space is a USS forked or spawned procedure. * Error: the address space where the command originated has closed. **?** Error: inconsistent data (might be a transient condition). **!** Error: inconsistent data. **>** Error: should not occur.
C2B	Converts input binary to a Boolean string (EBCIDC "0" and "1" values)
C2D	Converts input to a string representing a decimal number.
C2X	Converts input to a string representing its hexadecimal notation.
CMDX	Inputs the first 88 (X'58') bytes of the IEZVX101 control block.
CONSAUTH	Indicates authority of the console issuing the command: **Value** **Description** **M** Master **I** I/O **S** SYS **C** CONSOLE
CONSNAME	Returns the issuing console name.

Table 4. Edit Orders (continued)

Edit Order	Description
D2C	Converts a signed integer number into a full-word.
D2X	Converts a signed decimal number to a hexadecimal representation
DELETE	Output order, binary input "Y" or "1" indicates that the command is to be deleted.
JOBNAME	Input order, specifies the 8-character JES job name of the originator of the command.
LEFT	Truncates or pads the input to the length specified. Characters are counted from the beginning, or left, of the input.
NEXT	Specifies that the input is to be placed into the output without an intervening blank.
NEXTWORD	Specifies that the input is to be placed into the output with an intervening blank.
NVABLE	Returns "Yes" if a NETVONLY action can succeed, otherwise returns an "No".
ONTO	Sets the logical end of command text for all input order.
PAD	Specifies the padding character to be used by subsequent orders. Examples of orders which use the padding character include the LEFT conversion order and the position output order.
PARSE	Specifies how the WORD input order counts words.
PREFIX	Conversion order; adds a literal string to the beginning of input text.
RESET	Cancels all previous SKIPTO and UPTO orders. The original input line is made available to input orders specified subsequent to RESET.
RIGHT	Truncates or pads the input to the length specified. Characters are counted from the end, or right, of the input.
RVAR	From the input revision variable name, returns the current value or a null string.
SKIPTO	Sets the logical start of the line for input orders position length and WORD to be a point other than the first character in the line.
STRIP	Specifies that padding characters at the start or end of the data are to be removed.
STRIPL	Specifies that padding characters at the beginning of the data are to be removed.
STRIPR	Specifies that any padding characters at the end of the data are to be removed.
SUBSTR	Specifies that a subset of the input data is to be selected.
SYSNAME	Specifies the 8–character name of the system from which the command originated.
UPTO	Redefines the logical end of the input line.
WORD	Specifies the subset of the input line to be processed. The subset is defined by specifying a starting word and the total number of words.
X2C	Converts a hexadecimal EBCDIC string into binary notation.
YESNO	Converts a 1-byte field to the character string Yes or No.

See the NetView online help or *IBM Tivoli NetView for z/OS Programming: Pipes* for more information on the PIPE EDIT orders.

Command Revision Table Example

This is an example of a CRT (CNMSCRT1 sample):

```
UPON(CMDVERB = 'V' | CMDVERB = 'VARY')
   SELECT
     WHEN(WORD 2 = 'NETVIEW')                    ! V NetView,(anytext)
        NETVONLY=CNMSRVMC                        ! handle in NetView
     OTHERWISE                ! all other VARY cmds untouched
   END

UPON (CMDVERB='T' | CMDVERB='SET')
   SELECT
    WHEN (W2 next W3 = 'MPF NO')                    ! SET MPF=NO ?
      WTO("What do you think you're doing, Dave?")  ! Think about it
    OTHERWISE
   END
UPON (CMDCONS='ROOT')              ! For special console...
*   Note this "empty" UPON is tested first by cmd revision.  For this
*   special console is thus exempt from all CMDVERB & OTHERCMD actions
UPON (ALLCMD)              ! THIS applies even to console ROOT
  SELECT
    WHEN (CMDVERB ¬= 'SEND')
*  The above means that the following is only for the SEND cmd.
    WHEN (SKIPTO /USER=/ 1 FOUND ¬= 'Yes')        ! default = "ALL"
      WTO("TLH447E Please do not broadcast to all.")  ! explain to op
      REVISE('Y' DELETE)                          ! disallow default
    OTHERWISE
  END
  SELECT      ! Note: second, independent SELECT under the ALLCMD
    WHEN (CMDVERB ¬= 'IMSTDIS')      ! special to IMS cmd
    WHEN (W2 = 'CCTL')
      WTO("TLH851I Please review guidelines for SYSTAR1.")
      !! Adding a message to the command response
    OTHERWISE
  END
*  NOTE: The following depends on NetView having set a CHRON command
*  to issue SETRVAR (or clist containing SETRVAR) to establish a
*  value for SHIFT:  NORMAL (working hours), NIGHT (other times of
*  day, or HOLIDAY (non work days)
UPON (CMDVERB='S' | CMDVERB='START')
 SELECT
   WHEN (W2 ¬= /STATCOMP/)     ! starting special proc? see next WHEN
   WHEN ("SHIFT" RVAR = "NORMAL")  !
     NETVONLY=CNMSRVMC         ! double use of sample clist!
   WHEN ("SHIFT" RVAR = "HOLIDAY")
     ! cmd is allowed, no action here
   WHEN (SKIPTO "LIMIT=" 1 FOUND = 'Yes')   ! limit keyword present?
     ! at NIGHT, proc is allowed with LIMIT keyword specified
   OTHERWISE                  ! SHIFT assumed to be NIGHT, no LIMIT
     WTO('TLH722E Do not start' 1 WORD 2 NW
         'without LIMIT keyword, except holidays.')
     REVISE('Y' DELETE)
  END
* START T610EESS.SS,SUB=MSTR,PPIOPT=PPI
```

Usage Reports for Command Revision Tables

You can use the REVISE CMD REPORT command to gather statistics and usage information about the active revision table. If successful, a CNM014I message is issued. When the REPORT keyword is specified with the MEMBER operand, the information displayed is about the table being replaced and the time it was replaced.

See the NetView online help or the *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for more information about using the REVISE CMD command.

Command Revision Table Testing

You can use these steps to test your command revision table:

1. Issue a command using the MVS command. For example:

   ```
   MVS SET MPF=NO
   ```

2. Use the NetView command revision table to issue a WTO instead of running the command. A section of the command revision table to do this is:

   ```
   UPON (CMDVERB='T' | CMDVERB='SET')
      SELECT
       WHEN (W2 next W3 = 'MPF NO')                    ! SET MPF=NO ?
         WTO("This command has been blocked by the NetView program.")
       OTHERWISE
      END
   ```

Chapter 15. The Automation Table

This chapter describes:
- The NetView automation table
- The statements you can use in an automation table
- How to code an automation table
- The syntax of automation-table statements
- Design guidelines for automation tables
- Usage reports for automation tables

For information about using the automation table to automate messages and MSUs, see Chapter 22, "Automating Messages and Management Services Units (MSUs)," on page 317.

For information about testing the logic of the automation table, see Chapter 34, "Automation Table Testing," on page 471.

What Is the Automation Table?

The automation table enables you to respond automatically to messages and management services units (MSUs). This table contains statements that define actions that the NetView program takes when it receives specific messages and MSUs. For example, you can issue a response in the form of a command, command list, or command processor.

You can also set attributes and processing options. For example, you can suppress, log, or route messages and block, record, or highlight MSUs.

The automation table also processes commands that are echoed to the screen, treating them as messages. To stop the automation table from processing commands, add a statement at the top of the table that ends processing if the message type is an asterisk (HDRMTYPE = '*') or another command-related message type.

Elements of Automation-Table Statements

An automation table contains these elements:

- An *IF-THEN statement* enables you to specify messages and MSUs that you want the NetView program to automate. An IF-THEN statement contains a set of conditions followed by a set of actions that the NetView program is to perform when a message or MSU meets those conditions.
- A *BEGIN-END section* enables you to group statements together for processing. A BEGIN-END section starts with a BEGIN option on an IF-THEN statement and ends with an END statement.
- An *ALWAYS statement* enables you to specify actions to take place for all messages and MSUs that reach that statement in the table.
- A *%INCLUDE statement* enables you to include separately coded and maintained sections of the automation table to divide your automation-table maintenance among several groups or individuals. You can view your INCLUDE structure using the automation-table management function (AUTOMAN).
- A *SYN statement* enables you to define synonyms for use later in the table. Each SYN statement includes a name and an associated value.

You store automation-table statements in member DSIPARM. You can store the statements that make up an automation table in a single member or in a set of members that you include in a main automation-table member with the %INCLUDE statement.

Automation-Table Processing

You can use either the AUTOMAN or AUTOTBL command to activate, deactivate, test, list, or check the status of an automation table or set of tables. You can also enable or disable individual statements or groups of statements in an automation table that has been defined to provide this functionality.

For more information about AUTOMAN, see "Managing Multiple Automation Tables" on page 248.

For the syntax of the AUTOTBL command and detailed information, refer to the NetView program online help. "Example of an Automation-Table Listing" on page 236 shows the results of using the AUTOTBL command to list an automation table.

When you activate an automation table, the NetView program first resolves all %INCLUDE and SYN statements by incorporating all included members and substituting synonym values for synonym names. Only IF-THEN statements, BEGIN-END sections, and ALWAYS statements directly affect the processing of messages and MSUs.

Automation-Table Searches

When the NetView program receives a message or MSU and an automation table is active, the NetView program searches the active automation table sequentially, looking for:

- Conditions that match the received message or MSU
- An ALWAYS statement, which matches unconditionally

When a match is found, the NetView program performs the actions that the matching statement specifies. If the matching statement specifies CONTINUE(Y), the NetView program continues searching for an additional match. If the matching statement does not specify CONTINUE(Y), the NetView program ends its search of the automation table for the message or MSU.

Types of Automation-Table Statements

Not all of your automation-table statements apply to all incoming data. When a message is processed, the NetView program checks only the automation statements that apply to messages. When an MSU is processed, the NetView program checks only the automation statements that apply to MSUs.

An IF-THEN or ALWAYS statement must be one of three types: message, MSU, or both.

- A message-type statement applies only to messages.
- An MSU-type statement applies only to MSUs.
- A both-type statement applies to either messages or MSUs

The type of an IF-THEN statement depends on the types of condition items and actions the statement contains. The type of an ALWAYS statement depends on the types of actions the statement contains.

A condition item or an action can be of three types: message, MSU, or both. To determine the types of condition items or actions, see the descriptions of the specific items or actions in this chapter. "Condition Items" on page 157 describes condition items, and "Actions" on page 209 describes actions.

Determining the Type of Statement

The rules for determining the type of an IF-THEN or ALWAYS statement are:

- If all condition items and actions are of type message, the statement type is message.
- If all condition items and actions are of type MSU, the statement type is MSU.
- If all condition items and actions are of type both, the statement type is both.
- If some condition items and actions are of type both and some are message, the statement type is message.
- If some condition items and actions are of type both and some are MSU, the statement type is MSU.
- If some condition items and actions are of type message and some are of type MSU, the statement is not valid.
- A statement with no condition items or actions, such as `ALWAYS`, is of type both.
- If any parts of a statement are not valid, the statement is not valid.

Statement Types and Processing

The statement type also affects the processing of BEGIN-END sections. A BEGIN-END section is the same type as the statement that contains the BEGIN keyword and begins the section.

- A BEGIN-END section that starts with a message statement type is type message and can contain statements or other BEGIN-END sections whose types are message or both.
- A BEGIN-END section that starts with an MSU statement type is type MSU and can contain statements or other BEGIN-END sections whose types are MSU or both.
- A BEGIN-END section that starts with a both statement type is type both and can contain statements or other BEGIN-END sections whose types are message, MSU, or both.
- A message-type BEGIN-END section containing MSU-type statements or an MSU-type BEGIN-END section containing message-type statements is not valid.

You cannot activate an automation table that contains statements or BEGIN-END sections that are not valid.

When the automation table receives a message, the NetView program processes only statements and BEGIN-END sections of type message or both. When the automation table receives an MSU, the NetView program processes only statements and BEGIN-END sections of type MSU or both.

Coding an Automation Table

These directions and restrictions apply to coding the automation table.

- You must store the automation table in a member that has a fixed 80-character format. You can code statements in columns 1–72.
- Columns 73–80 are for sequence numbers.

  Sequence numbers are optional, but if they are used they:

  - Must begin in column 73
  - Must consist of alphanumeric characters, but can also include the characters @, $, and #
- You must code a semicolon (;) at the end of each statement except the %INCLUDE statement.
- The automation table can be coded in mixed case. The case is preserved for:
  - Comments
  - Character literals (quoted strings)
  - Synonym names
  - Synonym values
  - The member name on a %INCLUDE statement

  Other statement components are internally changed to uppercase during processing of the table. This might result in error messages displayed as uppercase statements.
- You can use blanks to indent lines and to separate keywords, logical operators, and parentheses.

  However, blanks used within a comparison string are considered characters in that string.
- You can continue a statement on as many lines as needed, using columns 1–72. You can code a statement through column 72 of one line and continue the same statement in column 1 of the next line. In doing so, the lines are concatenated together. This allows long symbols to be split across lines.

  You can stop a line after any logical operator, a parenthesis, a completed condition, or an operand, and resume the statement anywhere in the first 72 columns of the next line.
- You must use single quotation marks as the delimiters for comparison text and for synonym values.
  - If a synonym value or comparison text contains a single quotation mark ('), you must represent it as two consecutive single quotation marks ('').
  - Do not substitute a double quotation mark for two single quotation marks.
- Place comments on separate lines for automation-table members.
  - Do not put comment lines between the beginning and end of a continued automation-table statement.
  - Each comment line must contain an asterisk (*) in the first column.
- System symbolic substitution is performed on automation-table statements read from an automation-table member in the DSIPARM data set.

  The &DOMAIN symbolic that is supplied with the NetView program is also included in the substitution process. The substitution is performed after comment removal but before record processing. Comments are also removed after substitution. Substitution is always performed on the &DOMAIN symbolic (unless substitution was disabled when the NetView program was started).

  For MVS and user-defined system symbolics, substitution is not performed if you are running on an MVS system prior to MVS Version 5 Release 2.
- Japanese double-byte characters are not supported in the automation table.

BEGIN-END Section

BEGIN-END sections contain a series of automation-table statements. An END statement ends a series of statements started with the BEGIN option on an IF-THEN or ALWAYS statement. You can use BEGIN-END sections to logically segment an automation table or to help improve the performance of automation-table processing.

The syntax for a BEGIN-END section is:

BEGIN-END Section

Where:

IF Starts an IF-THEN statement, as described in "IF-THEN Statement" on page 152.

conditions
Are the conditions that determine whether the actions indicated by THEN are to be processed, as previously described.

THEN Starts the THEN part of an IF-THEN statement, as described previously.

ALWAYS
Starts an ALWAYS statement, as described in "ALWAYS Statement" on page 228. Starting a BEGIN-END section with the ALWAYS statement is equivalent to simply coding *statements* without a BEGIN-END section.

BEGIN
Indicates the beginning of a series of statements. A BEGIN statement cannot be on the same line as an END statement.

statements
Indicates any series of statements, which can include SYN, %INCLUDE, IF-THEN, and ALWAYS statements and other BEGIN-END sections.

END Indicates the end of a series of statements. An END statement cannot be on the same line as a BEGIN statement.

Usage notes:

1. You cannot combine BEGIN with actions on a single IF-THEN statement.
2. You must provide a matching END statement for each BEGIN statement.
3. If the conditions are true, automation-table processing continues with the first statement within the section (the statement after BEGIN).

   If the conditions are not true, automation-table processing continues at the next statement after the END statement that ends the section.
4. You can nest BEGIN-END sections. That is, a BEGIN-END section can contain other BEGIN-END sections.
5. The types of statements used within a BEGIN-END section must be consistent with each other and, for an IF-THEN statement, with the conditions specified in the IF part of the statement.

   You cannot mix MSU-type and message-type statements, although you can mix both-type statements with either MSU-type or message-type statements.

   See "Types of Automation-Table Statements" on page 148 for more information.

6. A variable set (in the *conditions* part of an IF-THEN statement that starts a BEGIN-END section) is accessible for use in EXEC actions throughout the BEGIN-END section.

   The conditions portion (of a lower-level IF-THEN statement within the section) can assign a value to the same variable name, temporarily overriding the value. At the end of the lower-level IF-THEN statement (or its BEGIN-END section), the variable reverts to the value defined in the higher-level IF-THEN statement.

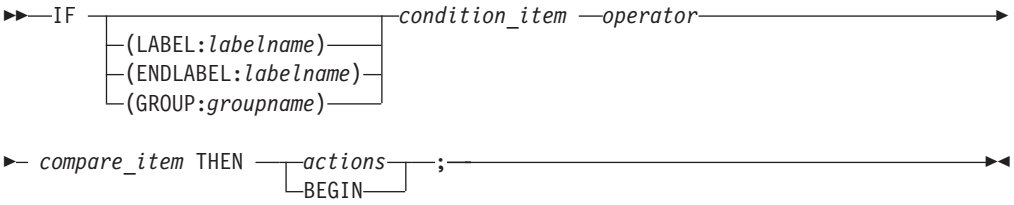
IF-THEN Statement

The IF-THEN statement enables you to specify messages and MSUs you want NetView automation to intercept and process. You can use the statement to code the conditions that a message or MSU must meet to be selected for automation, and the actions you want the NetView program to take if a message or MSU meets those conditions.

The NetView program evaluates the expressions stated before and after the operator in an IF statement. If the condition is true, the NetView program processes the THEN part of the statement. You can combine more than one condition with a logical-AND (&) operator, logical-OR (|) operator, and parentheses.

The syntax of the IF-THEN statement is:

IF-THEN Statement



Where:

IF The keyword you code at the beginning of each IF-THEN statement.

LABEL:*labelname*
The LABEL keyword identifies an automation-table statement or a BEGIN-END section to be specified with the DISABLE or ENABLE function of the AUTOTBL command.

- The *labelname* must be specified with alphanumeric characters, and can contain @, #, and $.
- The *labelname* can be up to 16 characters in length.

ENDLABEL:*labelname*
The ENDLABEL keyword identifies an automation-table statement or a BEGIN-END section to be specified with the DISABLE or ENABLE function of the AUTOTBL command.

Note:

- The *labelname* value must match the value on a previous LABEL keyword that is in the same member.
- If ENDLABEL is within a BEGIN-END section, the associated LABEL must be located within the same BEGIN-END section.

- The name used on the LABEL-ENDLABEL pair must be unique within the automation table.
- ENDLABEL must be specified with alphanumeric characters, and can contain @, #, and $.
- The *labelname* can be up to 16 characters in length.

The *labelname* value must match the value on a previous LABEL keyword which is in the same member.

GROUP:*groupname*
The GROUP keyword identifies an automation-table statement or a BEGIN-END section to be specified with the DISABLE or ENABLE function of the AUTOTBL command.

Note:

- One or more automation-table statements can be part of a named group of statements to be specified with the DISABLE or ENABLE function of the AUTOTBL command.
- The statements identified by a GROUP name can be in multiple members if desired.
- The *groupname* must be specified with alphanumeric characters, and can contain @, #, and $.
- The *groupname* can be up to 16 characters in length.

condition_item
The item being compared can be a bit string, character string, or a parse template.

See "Condition Items" on page 157 for more information about condition items.

operator
Indicates how the condition item is to be compared to the compare item.

= Indicates that if the condition item equals the compare item, the condition is true.

¬= Indicates that if the condition item does *not* equal the compare item, the condition is true.

< Indicates that if the condition item is less than the compare item, the condition is true.

Note:

- Variables and placeholders are not supported.
- Comparison values can differ in length.
- A null string is considered less than any other string.

<= Indicates that if the condition item is less than, or equal to, the compare item, the condition is true.

Note:

- You can specify => for the operator.
- Variables and placeholders are not supported.
- Comparison values can differ in length.
- A null string is considered less than any other string.

> Indicates that if the condition item is greater than the compare item, the condition is true.

Note:

- Variables and placeholders are not supported.
- Comparison values can differ in length.
- A null string is considered less than any other string.

>= Indicates that if the condition item is greater than, or equal to, the compare item, the condition is true.

Note:

- You can specify =< for the operator.
- Variables and placeholders are not supported.
- Comparison values can differ in length.
- A null string is considered less than any other string.

compare_item
The item to which the NetView program compares the condition item can be a bit string, character string, or a parse template.

See "Bit Strings as Compare Items" on page 204 for more information.

THEN The keyword coded on the second part of an IF-THEN statement.

actions Specifies actions for the NetView program to take when the IF conditions of the IF-THEN statement are true.

See "Actions" on page 209 for more information.

BEGIN
Specifies the start of a BEGIN-END section.

See "BEGIN-END Section" on page 151 for information.

Usage notes for IF-THEN Syntax:

1. You can include more than one condition in a statement. Link conditions with either a logical-AND (&) or a logical-OR (|) operator.
   - Ensure that there is a blank space proceeding and following the logical-AND (&).

     If the logical-AND (&) concatenates with other data, SYSCLONE support might change the logic of your IF-THEN statement.
   - When you link conditions with the logical-AND operator, *all* of the linked conditions must be true for the specified actions to be taken.
   - When you link expressions with the logical-OR operator, *at least one* of the linked conditions must be true for the specified actions to be taken.

   The IF-THEN statement in Figure 24 shows two conditions linked with the logical-AND operator. For the conditions to be true, the message must originate in domain CNM01, and its text must be `PURGE DATE IS LATER THAN TODAY'S DATE`.

```
IF DOMAINID='CNM01' &
   TEXT='PURGE DATE IS LATER THAN TODAY''S DATE' THEN
      EXEC (CMD('CLISTA') ROUTE (ONE * OPER1));
```

Figure 24. Example of Using the Logical-AND Operator

Figure 25 shows another example of two conditions linked with the logical-AND operator. In this example, the domain ID must be CNM02 and the MSU major vector key must be X'0000' (indicating an alert).

```
IF DOMAINID='CNM02' & MSUSEG(0000) ¬= '' THEN
      COLOR(YEL)
      CONTINUE(Y);
```

Figure 25. Additional Example of Using the Logical-AND Operator

The IF-THEN statement in Figure 26 shows two conditions linked with the logical-OR operator. If the message ID is IST051A or IST1311A, the NetView program takes the specified action.

```
IF MSGID='IST051A' | MSGID='IST1311A'
  THEN EXEC (CMD('CLISTA') ROUTE (ONE * OPER1));
```

Figure 26. Example of Using the Logical-OR Operator

2. The NetView program groups expressions linked with a logical-AND operator before those linked with a logical-OR operator.

   For example, the IF-THEN statement in Figure 27 has three conditions linked with logical-OR and logical-AND operators.

```
IF DOMAINID='CNM01' |
   TEXT='PURGE DATE IS LATER THAN TODAY''S DATE' &
   SYSID='MVS1' THEN
      EXEC (CMD('CLISTA') ROUTE (ONE * OPER1));
```

Figure 27. Example of Using the Logical-OR and Logical-AND Operator

   The NetView program evaluates the TEXT and SYSID conditions together (because a logical-AND operator links these two conditions). The TEXT and SYSID conditions must both be true or the DOMAINID condition must be true. The program then combines the result with the DOMAINID condition.

3. You can control the order in which the NetView program groups conditions by using parentheses around comparisons that you want the NetView program to evaluate together.

   This example presents the grouping of logical operators. If you want the NetView program to evaluate the DOMAINID and TEXT conditions together, place code parentheses around them, as shown in Figure 28.

```
IF (DOMAINID='CNM01' |
   TEXT='PURGE DATE IS LATER THAN TODAY''S DATE') &
   SYSID='MVS1' THEN
      EXEC (CMD('CLISTA') ROUTE (ONE * OPER1));
```

Figure 28. Example of Grouping Logical Operators

   When processing the IF-THEN statement in the previous example, the NetView program evaluates the DOMAINID and TEXT conditions together (because they are grouped within parentheses). The NetView program then combines the result with the SYSID condition. Either the DOMAINID or the TEXT condition must be true; and the SYSID condition must also be true.

   The NetView program ignores blank lines if they appear at the beginning of an MLWTO (multiline write-to-operator) message. The blank lines are retained for display purposes and can affect the location of lines when using GETMLINE in a command procedure.

An MLWTO message presented to MVS can have a control line (IEE932I), a sequential message identifier, or both appended to the message. The NetView program removes IEE932I to make the message more useful, but does not remove the sequential message identifier.

4. The series of IF-THEN statements in the next example shows automation-table statements that make up a block named VTAM.

   You can enable or disable the various statements by using the AUTOTBL ENABLE or DISABLE command with:

   - **LABEL=VTAM** to specify only the first statement in Figure 29
   - **ENDLABEL=VTAM** to specify only the last statement in Figure 29
   - **BLOCK=VTAM** to specify the entire block (all three statements) in Figure 29

   This series of IF-THEN statements is an example of using LABEL and ENDLABEL keywords.

```
IF (LABEL:VTAM) MSGID = 'IST051A'
   THEN EXEC (CMD('CLISTA') ROUTE (ONE * OPER1));
IF MSGID = 'IST052A'
   THEN EXEC (CMD('CLISTB') ROUTE (ONE * OPER1));
IF (ENDLABEL:VTAM) MSGID = 'IST053A'
   THEN EXEC (CMD('CLISTC') ROUTE (ONE * OPER1));
```

Figure 29. Example of Using LABEL and ENDLABEL Keywords

Figure 30, which presents the use of the GROUP keyword, includes several automation-table statements, some of which are part of a group of statements named VTAMX. You can use the AUTOTBL command to enable or disable all statements in the automation table that are part of this group by specifying a keyword of GROUP=VTAMX. In this example, the first and third statements are affected.

```
IF (GROUP:VTAMX) MSGID = 'IST054A'
   THEN EXEC (CMD('CLISTX') ROUTE (ONE * OPER1));
IF MSGID = 'IST055A'
   THEN EXEC (CMD('CLISTY') ROUTE (ONE * OPER1));
IF (GROUP:VTAMX) MSGID = 'IST056A'
   THEN EXEC (CMD('CLISTZ') ROUTE (ONE * OPER1));
```

Figure 30. Example of Using the GROUP Keyword

5. An MVS message issued by an unauthorized program has a plus sign added.

   The NetView program removes the plus sign and sets a field in the automation internal function request (AIFR) to indicate that the message was issued by an unauthorized program.

6. You can use the THEN keyword without either actions or a BEGIN-END section to indicate that the automation table is to take no further action for the message or MSU. Figure 31 presents the use of the THEN keyword without actions or a BEGIN-END section.

```
IF HDRMTYPE = '*' THEN ;
```

Figure 31. Example of Using THEN Keyword Without Actions

Condition Items

This section describes the condition items that you can use in an IF-THEN statement. Three tables show the condition items by type:

- Table 5 for messages
- Table 6 on page 158 for MSUs
- Table 7 on page 159 for messages and MSUs

With these exceptions, the text in these tables describes each condition item in alphabetic order:

- Display actions for messages are ignored unless the message is sent to the command facility for display.
- Display actions for MSUs are ignored unless the MSU contains an alert that is sent to the hardware monitor for display.

There are five types of MSUs:

- Control point management services units (CP-MSUs)
- Multiple domain support message units (MDS-MUs)
- Network management vector transports (NMVTs)
- Record maintenance statistics (RECMSs)
- Record formatted maintenance statistics (RECFMSs)

Table 5. IF Condition Items for Messages

Condition Item	Compare Item	Maximum Length	Description
ACTIONDL	Parse template	7 char	Tells why message was deleted
ACTIONMG	Bit String	1 bit	Indicates action message
AREAID	Parse template	1 char	MVS message area ID
AUTOTOKE[1]	Parse template	8 chars	MVS message processing facility automation token
CART[1]	Parse template	8 bytes	Command and response token
CORRELATED	Bit String	1 bit	Indicates the message is correlated
CORRFAIL	Bit String	1 bit	Indicates the correlation failed
DESC	Bit string	16 bits	MVS message descriptor codes
IFRAUWF1	Bit string	32 bits	MVS WTO information
JOBNAME	Parse template	8 chars	MVS originating job
JOBNUM	Parse template	8 chars	MVS assigned job number
KEY[1]	Parse template	8 chars	Key associated with a message
MCSFLAG	Bit string	16 bits	MVS multiple console support flags
MSGAUTH	Bit string	2 bits	Authorized program indicator
MSGCATTR[1]	Bit string	16 bits	MVS message-attribute flags
MSGCMISC[1]	Bit string	8 bits	MVS miscellaneous routing flags
MSGCMLVL[1]	Bit string	16 bits	MVS message-level flags
MSGCMSGT[1]	Bit string	16 bits	MVS message-type flags
MSGCOJBN[1]	Parse template	8 chars	Originating job name
MSGCPROD[1]	Parse template	16 chars	MVS product level
MSGCSPLX	Parse template	8 chars	Name of sysplex sending message
MSGDOMFL[1]	Bit string	8 bits	MVS delete operator message (DOM) flags

Table 5. IF Condition Items for Messages (continued)

Condition Item	Compare Item	Maximum Length	Description
MSGGBGPA[1]	Parse template	4 bytes	Background presentation attributes
MSGGDATE[1]	Parse template	7 chars	Date associated with a message
MSGGFGPA[1]	Parse template	4 bytes	Foreground presentation attributes
MSGGMFLG[1]	Bit string	16 bits	MVS general message flags
MSGGMID[1]	Parse template	4 chars	MVS message ID
MSGGTIME[1]	Parse template	11 chars	Time that the message was issued
MSGID	Parse template	255 chars	Message ID
MSGSRCNM[1]	Parse template	17 chars	Source name from source object
MVSRTAIN	Bit string	3 bits	MVS automation message retention facility (AMRF) AMRF retain flags
NVDELID	Parse template	24 char	NetView program deletion ID
ROUTCDE	Bit string	128 bits	MVS routing codes
SESSID	Parse template	8 chars	Terminal access facility session ID
SYSCONID	Parse template	8 chars	System console name
SYSID	Parse template	8 chars	ID of originating MVS system
TEXT	Parse template	255 chars	Message text
TOKEN	Parse template	255 chars	In a message text, a string delimited by blanks
Note: [1] This condition item does not have a value unless the message being processed was originally a message data block (MDB).			

Table 6. IF Condition Items for MSUs

Condition Item	Compare Item	Maximum Length	Description
CORRELATED	Bit String	1 bit	Indicates the MSU is correlated
CORRFAIL	Bit String	1 bit	Indicates the correlation failed
HIER	Parse template	See "HIER" on page 168.	Resource hierarchy associated with an MSU
HMASPRID[2]	Parse template	9 chars	Returns the alert sender product ID
HMBLKACT[2]	Parse template	5 chars	Returns the block ID and action code of an MSU
HMCPLINK[2]	Bit string	1 bit	Returns an indicator that specifies whether a complex link exists
HMEPNAU[2]	Parse template	16 chars	Returns the network addressable unit (NAU) name of the entry point node where the MSU originated for MSUs forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol or the NV-UNIQ/LUC alert forwarding protocol.
HMEPNET[2]	Parse template	16 chars	Returns the netid name of the entry point node where the MSU originated for MSUs forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol.
HMEPNETV[2]	Bit String	1 bit	Returns an indicator that specifies whether the entry point node 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.
HMEVTYPE[2]	Parse template	4 chars	Returns the event type of an MSU

Table 6. IF Condition Items for MSUs (continued)

Condition Item	Compare Item	Maximum Length	Description
HMFWDED[2]	Bit string	1 bit	Returns an indicator that specifies whether an MSU was forwarded from another node over the NV-UNIQ/LUC alert forwarding protocol
HMFWDSNA[2]	Bit string	1 bit	Returns an indicator that specifies whether an MSU was forwarded from a remote entry point node using the SNA-MDS/LU 6.2 alert forwarding protocol
HMGENCAU[2]	Parse template	1 char	Returns the general cause code of an MSU, in hexadecimal
HMONMSU[2]	Bit string	1 bit	Returns an indicator that specifies whether an MSU was submitted to automation by the hardware monitor
HMORIGIN[2]	Parse template	8 chars	Returns the name of the resource sending the MSU
HMSECREC[2]	Bit string	1 bit	Returns an indicator specifying whether the hardware monitor performs secondary recording
HMSPECAU[2]	Parse template	2 chars	Returns the specific component code of an MSU, in hexadecimal
HMUSRDAT[2]	Parse template	5 chars	Returns the user data from subvector 33 of an MSU
MSUSEG	Parse template or bit string	See "MSUSEG" on page 195.	MSU data
Note: [2] This condition item returns a null value if the MSU was not submitted for automation by the hardware monitor.			

Table 7. IF Condition Items for Messages and MSUs

Condition Item	Compare Item	Maximum Length	Description
ACQUIRE	Parse template	See "ACQUIRE" on page 160.	Value determined by the edit specification
ATF	Parse template or bit string	See "ATF" on page 162.	Value determined by a specified ATF program
ATF(DSICGLOB)	Parse template	See "DSICGLOB" on page 163.	Value of a common global variable
ATF(DSITGLOB)	Parse template	See "DSITGLOB" on page 164.	Value of a task global variable
ATTENDED	Bit string	1 bit	Attended task indicator
AUTOMATED	Bit string	1 bit	Significant action indicator
AUTOTASK	Bit string	1 bit	Autotask indicator
CURRDATE	Parse template	8 chars	Current date
CURRTIME	Parse template	8 chars	Current time of day
CURSYS	Parse template	8 chars	Current MVS operating system name
DISTAUTO	Bit string	1 bit	Distributed autotask indicator
DOMAIN	Parse template	5 chars	Current NetView program domain name
DOMAINID	Parse template	8 chars	Originating NetView program domain
HDRMTYPE	Parse template	1 char	Message type
IFRAUIND	Bit string	16 bits	AIFR indicator flags
IFRAUIN3	Bit string	8 bits	Indicator-bit field
IFRAUI3X	Bit string	32 bits	Indicator bits

Table 7. IF Condition Items for Messages and MSUs (continued)

Condition Item	Compare Item	Maximum Length	Description
IFRAUSB2	Parse template	2 chars	AIFR user field
IFRAUSC2	Bit string	128 bits	AIFR user field
IFRAUSDR	Parse template	8 chars	Name of originating NetView program task
IFRAUSRB	Bit string	16 bits	AIFR user field
IFRAUSRC	Parse template	16 chars	AIFR user field
IFRAUTA1	Bit string	48 bits	AIFR control flags
INTERVAL	Bit string	1 bit	Occurrence interval detection
LINEPRES	Parse template	4 bytes	Presentation attributes of first text buffer
LINETFLG	Bit string	16 bits	Presentation override flag (bit 16) and other flags
MVSLEVEL	Parse template	8 chars	Current MVS product level
NETID	Parse template	8 chars	VTAM network identifier
NETVIEW	Parse template	4 chars	NetView program version and release
NUMERIC	Parse template	255 chars	Numeric value of a variable
NVCLOSE	Bit String	1 bit	NetView program CLOSE processing flag
OPID	Parse template	8 chars	Operator or task ID
OPSYSTEM	Parse template	7 chars	Operating system
SYSPLEX	Parse template	8 chars	Local MVS sysplex name
TASK	Parse template	3 chars	Type of task
THRESHOLD	Bit string	1 bit	Occurrence threshold detection
VALUE	Parse template	255 chars	Value of a variable
VTAM	Parse template	4 chars	VTAM level
VTCOMPID	Parse template	14 chars	VTAM component identifier
WEEKDAYN	Parse template	1 char	Day of the week

This is an alphabetical list of the condition items.

ACQUIRE ('*edit_specification***')**

A condition item that enables you to extract AIFR data using the syntax and function provided by the PIPE EDIT stage.

Only the first line of the returned message buffer is used for comparison. The AIFR path) continues unaltered through the automation process.

While *edit_specification* must be enclosed in single quotation marks (in the form '*edit_specification*'), you cannot use single quotation marks within the *edit_specification* itself.

For specific information about the *edit_specification*, refer to the *IBM Tivoli NetView for z/OS Programming: Pipes*.

ACTIONDL [(*pos* **[***len***])]**

The reason for deleting the NetView program action message. The reason is expressed in an EBCDIC string that is 1-8 characters in length.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

If the value of ACTIONDL is not null (''), the automation table is processing a DOM (Delete Operator Message), as contrasted to a message or an alert.

Valid values are as follows:

'' Null; the message is not a DOM.

ASID The message was deleted because the address space ended that issued the message.

INVALID
The DOM contained an unrecognizable combination of bit settings.

LOCAL
The message was deleted by an operator overstrike or by the CONSOLE DELETE stage.

NETVIEW
The message was deleted by the NetView program DOM command using the NVDELID option, or internally by the NetView program.

SMSGID
The message was deleted by an MVS DOM-by-SMSGID. A single message was deleted by its specific identifier.

TCB The message was deleted because the task ended that issued the message.

TOKEN
The message was deleted by an MVS DOM-by-token.

Maximum length: 7 characters

Type: Message

Usage notes for ACTIONDL::

1. MVS might convert TCB and ASID conditions to DOM-by-SMSGID.
2. SMSGID is the most frequent type of MVS DOM.
3. Related condition items are ACTIONMG and NVDELID. Also see "DOMACTION" on page 212.

ACTIONMG
Indicates whether the message is treated by the NetView program as an MVS action message. Values for ACTIONMG are:
1 The message is an action message.
0 The message is not an action message.

Maximum length: 1 bit

Type: Message

Usage notes for ACTIONMG::

1. Action messages are WTORs or those marked with a Descriptor code that matches one of those specified on the MVSPARM.ActionDescCodes CNMSTYLE statement.
2. Related condition items are ACTIONDL and NVDELID. Also see "DOMACTION" on page 212.

AREAID [(*pos* **[***len***])]**
The one-letter identifier (A–Z), on the multiple console support console that displays the message.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of AREAID evaluates to null ('') if the value is B'0' or blank. You can test for these cases by comparing to the null ('') keyword.

Maximum length: 1 character

Type: Message

ATF ([BIT] '*cmdstring*')
Identifier for a program that is called to perform an automation-table function (ATF).

For a description of how to write your own ATF programs, refer to the *IBM Tivoli NetView for z/OS Programming: Assembler*.

The condition item is a value that the program returns.

The compare item is either a bit string or a parse template.

BIT Indicates that the compare item is a bit string. If you do not specify BIT, the compare item is a parse template.

cmdstring
The command string that calls the program.

The text of the string up to the first blank (or the whole string, if there are no blanks) is the program name. Any text after the first blank is passed as parameters to the called program.

The program name must be specified with a literal quoted string. However, variable values can be passed as ATF program parameters using the VALUE (*varname*) syntax.

After the program name is specified, the parameters can be specified by any combination of literals and VALUE specifications.

Variables that are passed must meet these criteria:

- Variables that were passed as ATF must be previously defined in the statement or BEGIN hierarchy.
- Variables that have not been set are treated as a null literal.
- A variable cannot be subscripted with position or length.

Maximum length: 256 bytes

Type: Both

Usage notes for ATF:

1. These criteria apply to ATF and *cmdstring*:
   - The length of *cmdstring* with its parameters is limited to 256 bytes (less the length of BUFHDR).
   - The ATF program name in *cmdstring* has a maximum length of 8 characters.
   - The length of the value returned by the ATF is limited to 256 bytes (minus the length of BUFHDR).
2. The interface is based on a parameter list whose address is in register 1.

   The register contains pointers to the control work block (CWB) and to the AIFR being automated.

3. The ATF return codes in register 15 are:

Code	Meaning
0	Normal
1–8	Indicates an error that causes the comparison to be evaluated as false
9 or greater	Indicates an error that results in error message CNM588E and a comparison evaluation of false

4. When you successfully activate an automation table with the AUTOTBL command, the NetView program loads all of the ATF programs your table uses.

   The NetView program does not reload the ATF program into main storage every time a message or MSU goes through the automation table.

5. The NetView program samples provide OPERID (CNMS4295) as an example of an ATF program.
6. ATF does not support a length specification.

   You can assign ATF to a variable and then use that variable (including *pos* and *len*) in a VALUE conditional statement.

ATF ([BIT] 'DSICGLOB *varname***')**

DSICGLOB is an ATF program that is supplied with the NetView program. If a command list or command processor has previously established a value for the common global variable, DSICGLOB returns that value. If the value is longer than 256 characters minus the length of BUFHDR, the value is truncated. If no value has been established for the variable, DSICGLOB does not return a variable value.

The compare item is either a bit string or a parse template. Use a parse template because the value of a global variable is a string of EBCDIC characters.

For information about how to specify *varname*, see the description of *cmdstring* for generic ATFs.

Any error encountered by the ATF program forces the condition item to evaluate as false and elicits a CNM588E message containing a return code:

Code	Meaning
100	A variable name is not valid.
104	The variable name used is too long.
108	No variable name is specified.
112	A NetView program storage failure.
116	A NetView program internal error.

BIT Indicates that the compare item is a bit string. If you do not specify BIT, the compare item is a parse template.

varname
The name of the common global variable. The length of the name is from 1-31 characters, and the name must be a valid global variable name. Refer to the *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language* for restrictions on variable names.

Maximum length: 256 characters

Type: Both

Note: If the automation table calls DSICGLOB for a NetView program message sent to the immediate message area of the operator's screen (TVBINXIT bit is on), DSICGLOB does not return a variable value, and the condition evaluates as false.

ATF ([BIT] 'DSITGLOB *varname*')

DSITGLOB is an ATF program that is supplied with the NetView program. If a command list or a command processor has previously established a value for the task global variable, DSITGLOB returns that value. If the value is longer than 256 characters minus the length of BUFHDR, the value is truncated. If no value has been established for the variable, DSITGLOB does not return a variable value.

The compare item is either a bit string or a parse template. Use a parse template because the value of a global variable is a string of EBCDIC characters.

For information about how to specify the *varname*, see the description of *cmdstring* for generic ATFs.

Any error encountered by the ATF program forces the condition item to evaluate as false and elicits a CNM588E message containing a return code:

Code	Meaning
100	A variable name is not valid.
104	The variable name used is too long.
108	No variable name is specified.
112	A NetView program storage failure.
116	A NetView program internal error.

BIT Indicates that the compare item is a bit string. If you do not specify BIT, the compare item is a parse template.

varname
The name of the task global variable. The length of the name is 1 - 31 characters in length, and the name must be a valid global variable name. Refer to the *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language* for restrictions on variable names.

Maximum length: 256 characters

Type: Both

Usage notes for ATF:

1. The task global variable returned by DSITGLOB is the one for the task that invoked the automation table. If you cannot predict which task invokes the automation table and causes the evaluation of the ATF, use a common global variable and the DSICGLOB ATF instead.
2. If the automation table calls DSITGLOB for a NetView program message sent to the immediate message area of the operator's screen (TVBINXIT bit is on), DSITGLOB does not return a variable value, and the condition evaluates as false.

ATTENDED [(*pos* [*len*])]

Describes the NetView program task that is automating a message or MSU. It is a bit string with a value of 1 or 0. The values for ATTENDED are:

1 Indicates that the task is one of the following situations:

- 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 situations:

- An autotask without an associated MVS console assigned using the AUTOTASK command
- Another type of task, such as a DST or an OPT task

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: Both

Usage notes for ATTENDED:

1. If the associated operator is an autotask, the presentation data is not eligible for display unless the autotask is associated with an active MVS console.
2. You can use ATTENDED in conjunction with DISTAUTO or AUTOTASK condition items to further define the characteristics of the task that is automating the message or MSU. For example, if ATTENDED is 1, DISTAUTO is 0, and AUTOTASK is 1, the task is an autotask with an associated MVS console.

AUTOMATED [(*pos* [*len*])]

Describes the automation indicator of the AIFR containing the message or MSU.

It is a one-bit indicator that specifies whether the AIFR has been automated by a previous significant action. Values for AUTOMATED are as follows:

1 The AIFR has been automated.

0 The AIFR has not yet been automated.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Automation treats an AIFR as AUTOMATED if a match occurs other than the ALWAYS or CONTINUE(YES) statements, unless the AUTOMATED action is used to override these defaults.

Maximum length: 1 bit

Type: Both

AUTOTASK [(*pos* [*len*])]

Condition item which describes the NetView program task that is automating the message or MSU. This is a one-bit indicator that specifies whether a task is an autotask. Values for AUTOTASK are:

1 The task is an autotask.

0 The task is not an autotask.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: Both

AUTOTOKE [(*pos* [*len*])]
Indicates the 1 to 8 character name of the MVS message processing facility (MPF) automation token.

If you specify AUTO(YES) or AUTO(NO) in the MPF table, the values YES and NO are not automation tokens.

AUTOTOKE has a value only if the message was originally a message data block (MDB).

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Message

CART [(*pos* [*len*])]
Specifies the 8-byte MVS command and response token (CART). The CART might contain characters that cannot be displayed.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of CART evaluates to null ('') if the field contains only binary zeros. You can test for this case by comparing the null ('') keyword.

Maximum length: 8 bytes

Type: Message

CORRELATED
Condition item that checks if a message or MSU is correlated. This is a one-bit indicator. Values for CORRELATED are:
1 The message or MSU is correlated.
0 The message or MSU is not correlated.

Maximum length: 1 bit

Type: Both

CORRFAIL
Condition item that checks if correlation failed. This is a one-bit indicator. Values for CORRFAIL are:
1 An internal error prevented correlation.
0 A problem was not encountered with correlation.

Maximum length: 1 bit

Type: Both

CURRDATE [(*pos* [*len*])]
Indicates the 1-8 character current date (*yyyy/mm/dd*).

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Both

CURRTIME [(*pos* **[***len***])]**
Indicates the 1-8 character current time of day (*hh:mm:ss*).

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Both

CURSYS [(*pos* **[***len***])]**
Indicates the 1-8 character current MVS operating system name.

The system name returned by CURSYS can be different than the system name returned by SYSID:

- CURSYS is the name of the system where the automation table is processing.
- SYSID is the name of the system where the message originated.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Both

DESC [(*pos* **[***len***])]**
Identifies 1–16 MVS descriptor codes assigned to the message. Refer to the MVS library for information about code values.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 16 bits

Type: Message

DISTAUTO [(*pos* **[***len***])]**
Indicates whether a task is a distributed autotask started with the RMTCMD command. The DISTAUTO condition item describes the task that is automating the message or MSU. The values for DISTAUTO are as follows:

1 The task is a distributed autotask.
0 The task is not a distributed autotask.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: Both

DOMAIN [(*pos* **[***len***])]**
Specifies the 1-5 character name of the current NetView program domain.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 5 characters

Type: Both

DOMAINID [(*pos* **[***len***])]**
Specifies the 1-8 character domain name of the NetView system that originated the message or MSU.

For messages, DOMAINID gives the name of the NetView system that first processed the message. Note that for messages BNJ030I and BNJ146I, which are generated based on alerts, the DOMAINID indicates the name of the NetView syhstem that generated these messages.

For forwarded alerts from a hardware monitor to another NetView program, DOMAINID gives the name of the distributed NetView program that originally processed and forwarded the alert. For other MSUs, DOMAINID gives the name of the local NetView program that is doing the automation-table search.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Both

HDRMTYPE [(*pos* **[***len***])]**
Specifies the 1-character buffer type of the received message or MSU. Buffer types are described in Appendix G, "NetView Message Type (HDRMTYPE) Descriptions," on page 557.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 character

Type: Both

HIER [(*indexnum***)]**
Specifies the NetView program hardware monitor hierarchy data associated with an MSU. The compare item is a parse template.

indexnum
The index number (1-5) of a specific resource name-type pair.

HIER is set only if the MSU is received from the hardware monitor. If you specify an *indexnum*, the value of HIER is the single, specified name-type pair in the form `aaaaaaaa1111`, where `aaaaaaaa` is the 8-character name and 1111 is the 4-character type. The names and types are padded on the right with blanks, if necessary. If an alert has fewer than *indexnum* resources, the

value is null. If you do not specify an *indexnum*, the value of HIER is equal to a concatenation of all existing name-type pairs. For example, if there are three name-type pairs, the value is in this format:

```
aaaaaaaa1111bbbbbbbb2222cccccccc3333
```

There can be up to five name-type pairs. If an MSU does not have hierarchy information, the value of HIER is null. See "Using the Resource Hierarchy" on page 331 for HIER examples.

HIER does not support a length specification. You can assign HIER to a variable, and then use that variable (including *pos* and *len*) in a VALUE conditional statement.

Maximum length: 60 characters

Type: MSU

HMASPRID [(*pos* [*len*])]

Returns the 9-character alert-sender product ID. This is the same alert-sender product ID returned with the *prodid* parameter on the SRFILTER command. The ID can be either of these items:

- A hardware product ID that has from 1 to 4 characters
- A software product ID has from 1 to 9 characters

Trailing blanks are not truncated.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

HMASPRID returns a null if an MSU is either:

- Not a generic record

  Note: The term generic refers to all MSUs that contain subvector 92. Generic MSUs include:
  - Alerts that contain subvector 92
  - Resolutions, which contain subvector 92

- Not submitted to automation by the hardware monitor

Maximum length: 9 characters

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example 1: Searching for a Device

```
IF HMASPRID = '3745' THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

This example specifies that if a hardware monitor MSU is generic and from a 3745 device, the automation table calls the CLISTA command list and routes it to operator AUTO1.

Example 2: Specifying a Generic MSU

```
IF HMASPRID ¬= '' THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

This example specifies that if a hardware monitor MSU is generic, the automation table calls the CLISTA command list and routes it to operator AUTO1.

HMBLKACT[(*pos* [*len*])]

Returns a 5-character value, including a 3-character block ID and a

2-character action code. This value is identical to the *code* value of the SRFILTER command. Values are returned only for nongeneric alerts (X'0000') and RECMSs and RECFMSs that are not statistics-only.

Refer to the NetView program online help for information about the SRFILTER command.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

HMBLKACT returns a null if an MSU is:

- A generic alert (X'0000')

  Note: The term generic refers to all MSUs that contain subvector 92. Generic MSUs include:
  - Alerts that contain subvector 92
  - Resolutions, which always contain subvector 92
- A resolution (X'0002')
- A PD statistic (X'0025')
- Link configuration data (X'1332')
- A statistics-only RECMS

  Note: Statistics-only RECMS refers to record maintenance statistics that contain only statistical data. These records have a recording mode of X'81', X'86', and X'87' in byte 8, offset 1 of the RECFMS. For X'87', only RECMSs that represent temporary errors (not permanent) are considered statistics-only.
- A statistics-only RECFMS

  Note: Statistics-only RECFMS refers to record formatted maintenance statistics that contain only statistical data. These records have a type of 1, 4, and 5 in byte 8, offset 1 of the RECFMS.
- Not submitted to the automation table by the hardware monitor

Maximum length: 5 characters

Type: MSU

Applies to: All MSUs except those that cause a null value to be returned

Example 1: Checking for a Block ID and Action Code That is Not Null

```
IF HMBLKACT ¬= '' THEN COLOR(RED);
```

This example checks for MSUs with a block ID and action code that is not null, and colors them red.

Example 2: Checking for a Specific Block ID and Action Code

```
IF HMBLKACT = HEX'FFD03' THEN COLOR(RED);
```

This example checks for MSUs with a block ID of X'FFD' and an action code of X'03', and colors them red.

Example 3: Checking for a Specific Block ID

```
IF HMBLKACT = HEX('FFD') . &
   HMBLKACT = MYVAR THEN
     EXEC(CMD('CLISTA 'MYVAR) ROUTE(ONE AUTO1));
```

This example checks for MSUs with a block ID of X'FFD'. It does not check for a specific action code. The automation table calls the CLISTA command

list for MSUs with a block ID of X'FFD'. The block ID and action are passed to the CLISTA command list in variable MYVAR, and the command list is routed to operator AUTO1.

HMCPLINK[(*pos* **[***len***])]**
Returns a one-bit indicator, either 1 or 0, that specifies whether a complex link exists.

Indicator	Description
1	Indicates that 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 condition item. Use a system schematic to determine the complete hierarchy configuration when a complex link is present. For more information about the HIER condition item, see "HIER" on page 168. Hardware monitor panels, such as the Most Recent Events panel, indicate a complex link exists by placing an asterisk (*) in the pictorial resource hierarchy at the top of the panel, and displaying message BNJ1538I on the message line near the bottom of the panel.
0	Indicates that a complex link does not exist or that the hardware monitor did not submit the MSU to automation.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example 1: Checking for a Complex Link

```
IF HMCPLINK = '1' THEN COLOR(RED);
```

This example specifies that hardware monitor MSUs with a complex link are colored red.

Example 2: Checking for an MSU with No Complex Link

```
IF HMONMSU = '1' &
   HMCPLINK = '0' THEN COLOR(RED);
```

This example checks for an MSU that was forwarded by the hardware monitor and that has no complex link, and colors it red.

HMEPNAU[(*pos* **[***len***])]**
Returns the network addressable unit (NAU) name of the entry point node where the MSU originated. For local MSUs, HMEPNAU returns the local NAU (domain) name. For MSUs that were forwarded from a remote node entry point, the NAU name of the remote entry point is returned. This is true for both alert forwarding mechanisms: LU 6.2 and LUC.

For LU 6.2 forwarded alerts, the NAU name returned is the NAU name of the entry point node in which the MS application resides which first sent (forwarded) the alert to the ALERT-NETOP application. If the NetView program cannot determine with complete certainty that the NAU name

returned is the entry point NAU name (for example, it might be an intermediate node name) then the NAU name returned is preceded by an asterisk (*), for example, `*nauname`.

See Chapter 26, "Centralized Operations," on page 373 for more information about forwarding mechanisms.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 16 characters

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example: Searching for MSUs Forwarded from a Remote Entry Point
Checking for an MSU Forwarded from NETA.CNM01 Using LU 6.2

```
IF HMFWDSNA = '1' &
   HMEPNET  = 'NETA' &
   HMEPNAU = 'CNM01' THEN COLOR(RED);
```

This example specifies that hardware monitor MSUs that have been forwarded from remote entry point node NETA.CNM01 using the SNA-MDS/LU 6.2 alert forwarding protocol are to be colored red.

HMEPNET[(*pos* **[***len***])]**
Returns the netid name of the entry point node where the MSU originated. For local MSUs, HMEPNET returns the local netid name. For MSUs that were forwarded using LUC alert forwarding, HMEPNET returns an asterisk (*), because the NetView program cannot determine the netid name.

For MSUs that were forwarded using LU 6.2 alert forwarding, the netid name returned is the name of the entry point node where the MS application resides. If the NetView program cannot determine a netid name, HMEPNET returns an asterisk (*). If the NetView program can determine the netid name, but cannot with complete certainty determine that the netid name is the entry point netid name (for example it might be an intermediate node netid name) then HMEPNET returns the netid name preceded by an asterisk (*), for example `*netidnam`.

See Chapter 26, "Centralized Operations," on page 373 for more information about forwarding mechanisms.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 16 characters

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example: Checking for an MSU Forwarded from NETA.CNM01 Using LU 6.2

```
IF HMFWDSNA = '1' &
   HMEPNET  = 'NETA' &
   HMEPNAU = 'CNM01' THEN COLOR(RED);
```

HMEPNETV[(*pos* [*len*])]

Returns a one-bit indicator, either 1 or 0, that specifies whether the entry point node 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.

Indicator	Description
1	Indicates that the entry point was a NetView program.
0	Indicates that the entry point was not a NetView program or that the MSU was not forwarded using the SNA-MDS/LU 6.2 alert forwarding protocol.

See Chapter 26, "Centralized Operations," on page 373 for more information about forwarding mechanisms.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example: Searching for MSUs Forwarded from a Remote Node Entry Point Using LU 6.2

```
IF HMEPNETV = '1' THEN COLOR(RED);
```

This example specifies that hardware monitor MSUs, which have been forwarded from a remote entry point NetView program using the SNA-MDS/LU 6.2 alert forwarding protocol, are to be colored red.

HMEVTYPE[(*pos* [*len*])]

Returns a 4-character event type of the MSU. Trailing blanks are not truncated from the returned value.

The event types are:

AVAL	BYPS	CUST	DLRC
HMV	HELD	IMPD	IMR
INST	INTV	NTFY	PAFF
PERF	PERM	PROC	REDL
RSLV	RSNT	SCUR	SNA
TEMP	UNKN	USER	

Refer to the NetView online help (HELP NPDA '*event_type*') for more information.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

HMEVTYPE returns a null if an MSU is:

- Not submitted to automation by the hardware monitor
- A PD statistic (X'0025')
- Link configuration data (X'1332')
- A statistics-only RECMS

Note: Statistics-only RECMS refers to record maintenance statistics that contain only statistical data. These records have a recording mode of X'81', X'86', and X'87' in byte 8, offset 1 of the RECFMS. For X'87', only RECMSs that represent temporary errors (not permanent) are considered statistics-only.

- A statistics-only RECFMS

  Note: Statistics-only RECFMS refers to record formatted maintenance statistics that contain only statistical data. These records have a type of 1, 4, and 5 in byte 8, offset 1 of the RECFMS.

Maximum length: 4 characters

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example 1: Searching for Event Type PERM

```
IF HMEVTYPE = 'PERM' THEN COLOR(RED);
```

This example specifies that MSUs with an event type of PERM are colored red.

Example 2: Searching for Event Type SNA

```
IF HMEVTYPE = 'SNA' . THEN COLOR(RED);
```

These examples specify that MSUs with an event type of SNA are colored red. You do not have to check for the trailing blank.

Example 3: Extracting an Event Type

```
IF HMEVTYPE ¬= '' &
   HMEVTYPE = MYVAR THEN
     EXEC(CMD('CLISTA 'MYVAR) ROUTE(ONE AUTO1));
```

This example extracts the event type from the hardware monitor MSU, passes it to the CLISTA command list in variable MYVAR, and routes the command list to operator AUTO1.

HMFWDED[(*pos* **[***len***])]**

Returns a one-bit indicator, either 1 or 0, that specifies whether an MSU was forwarded from another node.

Indicator	Description
1	Indicates an MSU was forwarded from another node through one of these alerts: • NV-UNIQ/LUC alert forwarding protocol • SNA-MDS/LU 6.2 alert forwarding protocol
0	Indicates that the MSU was not forwarded through another node, was forwarded over LU 6.2, or that the hardware monitor did not submit the MSU to automation. An indicator of 0 is returned in these instances: • Local MSUs are received through the CNM interface. • Local MSUs are received from the operating system. • MSUs are received through the program-to-program interface. • MSUs are received through the SNA-MDS/LU 6.2 alert forwarding protocol.

Note: RECMSs and RECFMSs that are forwarded from entry points over LUC or LU 6.2 are not submitted to automation at the receiving focal point. RECMSs and RECFMSs are submitted to automation at the entry point, but not at the receiving focal point.

See Chapter 26, "Centralized Operations," on page 373 for more information about forwarding mechanisms.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example 1: Searching for MSUs Forwarded from an Entry Point

```
IF HMFWDED = '1' THEN COLOR(RED);
```

This example specifies that hardware monitor MSUs that have been forwarded from an entry point NetView program are to be colored red.

Example 2: Searching for MSUs Not Forwarded from an Entry Point

```
IF HMONMSU = '1' &
   HMFWDED = '0' THEN COLOR(RED);
```

This example checks for an MSU, which was forwarded by the hardware monitor but not from an entry point NetView program, and colors it red.

HMFWDSNA[(*pos* **[***len***])]**

Returns a 1-bit indicator, either 1 or 0, that specifies whether an MSU was forwarded from a remote entry point node using the SNA-MDS/LU 6.2 alert forwarding protocol.

Indicator	Description
1	Indicates that an MSU was forwarded from a remote entry point node using the SNA-MDS/LU 6.2 alert forwarding protocol.
0	Indicates that an MSU was not forwarded from a remote entry point node using the SNA-MDS/LU 6.2 alert forwarding protocol or that the hardware monitor did not submit the MSU to automation.

Refer to Chapter 26, "Centralized Operations," on page 373 for more information about forwarding mechanisms.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example: Checking for an MSU Forwarded from NETA.CNM01 Using LU 6.2

```
IF HMFWDSNA = '1' &
   HMEPNET  = 'NETA' &
   HMEPNAU = 'CNM01' THEN COLOR(RED);
```

HMGENCAU[(*pos* [*len*])]

Returns the 1-character hexadecimal general cause code of an MSU.

The general cause code indicates:
- The general classification
- The exception condition that caused the MSU to be created

For more information about general cause codes, refer to the information about basic alert (X'91') MS subvectors in the Systems Network Architecture library.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

HMGENCAU returns a value only for nongeneric alerts (X'0000') and RECMSs and RECFMSs that are not statistics-only. HMGENCAU returns a null if an MSU is:

- A generic alert (X'0000')

  Note: The term generic refers to all MSUs that contain subvector 92. Generic MSUs include:
  - Alerts that contain subvector 92
  - Resolutions, which always contain subvector 92
- A link event (X'0001')
- A resolution (X'0002')
- A PD statistic (X'0025')
- Link configuration data (X'1332')
- A statistics-only RECMS

  Note: Statistics-only RECMS refers to record maintenance statistics that contain only statistical data. These records have a recording mode of X'81', X'86', and X'87' in byte 8, offset 1 of the RECFMS. For X'87', only RECMSs that represent temporary errors (not permanent) are considered statistics-only.
- A statistics-only RECFMS

  Note: Statistics-only RECFMS refers to record formatted maintenance statistics that contain only statistical data. These records have a type of 1, 4, and 5 in byte 8, offset 1 of the RECFMS.
- Not submitted to automation by the hardware monitor

Maximum length: 1 hexadecimal character

Type: MSU

Applies to: All MSUs except those that cause a null value to be returned

Example 1: Checking for a General Cause Code That is Not Null

```
IF HMGENCAU ¬= '' &
   HMGENCAU = MYVAR THEN
     EXEC(CMD('CLISTA 'MYVAR) ROUTE(ONE AUTO1));
```

This example checks for a general cause code that is not a null, passes it to the CLISTA command list variable MYVAR, and routes the command list to operator AUTO1.

Example 2: Checking for a Specific General Cause Code

```
IF HMGENCAU = HEX('01') THEN COLOR(RED);
```

This example specifies that a hardware monitor MSU with a general cause code of X'01' is to be colored red.

HMONMSU[(*pos* **[***len***])]**

Returns 0 or 1 to indicate whether an MSU was forwarded to automation from the hardware monitor.

Indicator	Description
1	Indicates an MSU was forwarded from the hardware monitor.
0	Indicates that an MSU was not forwarded from the hardware monitor. It might have been submitted to automation by the generic receiver (NVAUTO), or by a user application that issued DSIAUTO or CNMAUTO.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: MSU

Applies to: All MSUs

Example 1: Checking for MSUs Submitted by the Hardware Monitor

```
IF HMONMSU = '1' THEN COLOR(RED);
IF HMONMSU ¬= '' THEN COLOR(RED);
```

These examples specify that MSUs submitted by the hardware monitor are to be colored red.

Example 2: Checking for MSUs Not Submitted by the Hardware Monitor

```
IF HMONMSU = '' THEN ;
IF HMONMSU = '0' THEN ;
```

These examples specify that MSUs not submitted by the hardware monitor are not sent to automation.

HMORIGIN[(*pos* **[***len***])]**

Returns the name of the resource sending the MSU.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Trailing blanks are not truncated from the value returned. The resource name returned by HMORIGIN is the same name displayed on the hardware monitor Alerts-Dynamic, Alerts-Static, and Alerts-History panels when ALT_ALERT=ORIGIN is specified in BNJMBDST.

Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information about the ALT_ALERT statement.

If a complex link does not exist in a resource hierarchy, the resource name returned with HMORIGIN is the same as the resource name returned with the HIER condition item. If a complex link does exist, the resource names might not be the same. Use the HMCPLINK condition item to determine whether a complex link exists. For more information, see "HMCPLINK" on page 171 and "HIER" on page 168.

HMORIGIN returns a null if the hardware monitor does not submit the MSU to automation.

Maximum length: 8 characters

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example 1: Checking for MSUs from GENALERT

```
IF HMORIGIN = 'GENALERT' THEN COLOR(RED);
```

This example specifies that MSUs sent from a resource named GENALERT are to be colored red.

Example 2: Extracting a Resource Name

```
IF HMORIGIN ¬= '' &
   HMORIGIN = MYVAR THEN
      EXEC(CMD('CLISTA 'MYVAR) ROUTE(ONE AUTO1));
```

This example extracts the resource name from the hardware monitor MSU, passes it to the CLISTA command list in variable MYVAR, and routes the command list to operator AUTO1.

HMSECREC[(*pos* **[***len***])]**

Returns a 0 or 1 to indicate whether the hardware monitor performs secondary recording for an MSU.

Indicator	Description
1	Indicates that secondary recording is performed for an MSU at the resource level returned by the HIER condition item. For more information, see "HIER" on page 168. See the NetView online help for information about secondary recording.
0	Indicates either: • 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. • The hardware monitor did not submit the MSU to automation.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 1 bit

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example: Checking for Secondary Recording

```
IF HMSECREC = '1' &
   HIER = MYHIER THEN
     EXEC(CMD('CLISTA 'MYHIER) ROUTE(ONE AUTO1));
```

This example checks for secondary recording on an MSU, passes the HIER resource hierarchy level data in variable MYHIER to the CLISTA command list, and routes the command list to operator AUTO1.

HMSPECAU[(*pos* **[***len***])]**

Returns 4 characters representing the 2-character hexadecimal specific component code of an MSU. A general cause code is returned.

The *pos* parameter is the position where the comparison begins. The default value is 1.

The *len* parameter is the length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The specific component code indicates the 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 information about specific component codes, refer to the information about basic alert (X'91') MS subvectors in the Systems Network Architecture library.

Values are returned only for nongeneric alerts (X'0000') and for RECMSs and RECFMSs that are not statistics-only. HMSPECAU returns a null if an MSU is:

- A generic alert (X'0000')

  Note: The term generic refers to all MSUs that contain subvector 92. Generic MSUs include:
  - Alerts that contain subvector 92
  - Resolutions, which always contain subvector 92
- A link event (X'0001')
- A resolution (X'0002')
- A PD statistic (X'0025')
- Link configuration data (X'1332')
- A statistics-only RECMS

  Note: Statistics-only RECMS refers to record maintenance statistics that contain only statistical data. These records have a recording mode of X'81', X'86', and X'87' in byte 8, offset 1 of the RECFMS. For X'87', only RECMSs that represent temporary errors (not permanent) are considered statistics-only.
- A statistics-only RECFMS

  Note: Statistics-only RECFMS refers to record formatted maintenance statistics that contain only statistical data. These records have a type of 1, 4, and 5 in byte 8, offset 1 of the RECFMS.
- Not submitted to the automation table by the hardware monitor

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 2 hexadecimal characters

Type: MSU

Applies to: All MSUs except those that cause a null value to be returned

Example 1: Checking for a Component Code That is Not Null

```
IF HMSPECAU ¬= '' &
   HMSPECAU = MYVAR THEN
      EXEC(CMD('CLISTA 'MYVAR) ROUTE(ONE AUTO1);
```

This example checks for a specific component code that is not null, passes the code to the CLISTA command list in variable MYVAR, and routes the command list to operator AUTO1.

Example 2: Checking for a Specific Component Code

```
IF HMSPECAU = HEX('0001') THEN COLOR(RED);
```

This example specifies that an MSU with a component code of X'0001' is colored red.

HMUSRDAT[(*pos* **[***len***])]**

Returns the 5-character user-specified data in subvector 33 of an MSU.

Trailing blanks are truncated 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 characters. The characters returned with HMUSRDAT reflect any translation done by the hardware monitor, and might not be the same characters in subvector 33. Use HMUSRDAT to determine whether the hardware monitor has translated any data in subvector 33 to underscores or uppercase.

You can also use MSUSEG to retrieve user-specified data from subvector 33 in an MSU. However, MSUSEG does not translate any characters.

For more information about subvector 33 data, the UDAT option of the GENALERT command, and the U option of the SRFILTER command, refer to the NetView online help

HMUSRDAT returns a null if an MSU:

- Does not contain subvector 33. Subvector 33 is never present in RECMS or RECFMS records. Only generic major vectors can contain subvector 33. The hardware monitor accepts and processes subvector 33 information in any of the generic major vectors submitted to automation.
- Is a frame relay (key X'1332').
- Is not submitted to automation by the hardware monitor.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 5 characters

Type: MSU

Applies to: All MSUs submitted to automation by the hardware monitor

Example 1: Checking for Specific User-Specified Data

```
IF HMUSRDAT = 'MYDAT' THEN COLOR(RED);
```

This example checks for hardware monitor MSUs with user-specified data of MYDAT in subvector 33, and colors them red.

Example 2: Checking for User-Specified Data

```
IF HMUSRDAT ¬= '' &
   HMUSRDAT = MYVAR THEN
     EXEC(CMD('CLISTA 'MYVAR) ROUTE(ONE AUTO1));
```

This example checks for hardware monitor MSUs with user-specified data in subvector 33, passes the data to the CLISTA command list in variable MYVAR, and routes the command list to operator AUTO1.

IFRAUIND [(*pos* [*len*])]
Indicates the AIFR indicator fields IFRAUIND and IFRAUIN2, which contain 16 bits.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Bit 16 indicates whether the message was solicited or unsolicited:
1 Unsolicited
0 Solicited

See Chapter 9, "NetView Program Information Routing for Automation," on page 81 for a discussion of solicited and unsolicited messages. Refer to the *IBM Tivoli NetView for z/OS Programming: Assembler* for a description of all other bits.

The value of IFRAUIND evaluates to null ('') if all bits are B'0'. You can test for this condition by comparing to the null ('') keyword.

Maximum length: 16 bits

Type: Both

IFRAUIN3 [(*pos* [*len*])]
The 8–bit AIFR field IFRAUIN3 mapped by DSIIFR.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The values for bits 1 and 2, which indicate the cross-domain priority, are:
B'00' A default priority
B'01' A low priority
B'10' A high priority
B'11' The receiver is to be tested for the priority

Maximum length: 8 bits

Type: Both

IFRAUI3X [(*pos* [*len*])]
The 32 bits of binary flags that are mapped by DSIIFR for a message.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The values for byte 3 (IFRAUI33) are:
X'80' Automate removal of message.
X'40' DOM is not expected.
X'20' DOM-by-token issued by MVS.
X'10' DOM issued for local copy only.

X'08' AIFR sent to AUTO(YES) console owner.

Maximum length: 32 bits

Type: Both

Usage notes for IFRAUI3X::

1. The first 8 bits of the 32-bit IFRAUI3X flags are IFRAUIN3 (see IFRAUIN3 for description).
2. For a detailed description of all the IFRAUI3X fields, browse the assembler macro DSIIFR that was shipped with your NetView program.

IFRAUSB2 [(*pos* **[***len***])]**
The 2-character AIFR user field IFRAUSRB mapped by DSIIFR.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of IFRAUSB2 evaluates to null ('') if the field contains all blanks or binary zeros in any combination. You can test for this case by comparing to the null ('') keyword.

Maximum length: 2 characters

Type: Both

Note: To compare using bits, use the IFRAUSRB condition item.

IFRAUSC2 [(*pos* **[***len***])]**
The 128-bit AIFR user field IFRAUSRC mapped by DSIIFR.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of IFRAUSC2 evaluates to null ('') if all bits are B'0'. You can test for this case by comparing to the null ('') keyword.

Maximum length: 128 bits

Type: Both

Note: To compare using characters, use the IFRAUSRC condition item.

IFRAUSDR [(*pos* **[***len***])]**
The name of the NetView program task that originated the message or MSU. IFRAUSDR is a 1-8 character name.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Both

IFRAUSRB [(*pos* **[***len***])]**
The 16-bit AIFR user field IFRAUSRB mapped in DSIIFR.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of IFRAUSRB evaluates to null ('') if all the bits are B'0'. You can test for this case by comparing to the null ('') keyword.

Maximum length: 16 bits

Type: Both

Note: To compare using characters, use the IFRAUSB2 condition item.

IFRAUSRC [(*pos* [*len*])]
The 16-character AIFR user field IFRAUSRC mapped in DSIIFR.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of IFRAUSRC evaluates to null ('') if all bytes are character blanks or binary zeros, in any combination. You can test for this case by comparing to the null ('') keyword.

Maximum length: 16 characters

Type: Both

Note: To compare using bits, use the IFRAUSC2 condition item.

IFRAUTA1 [(*pos* [*len*])]
Indicates the AIFR fields IFRAUTA1, IFRAUTA2, IFRAUTA3, IFRAUTA4, IFRAUTA5, and IFRAUTA6. See fields IFRAUTA1 through IFRAUTA6 in DSIIFR for more information.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Description
1, 2, 25	The HOLD action
5, 6, 26	The SYSLOG action
7, 8, 27	The NETLOG action
9, 10, 28	The HCYLOG action
11, 12, 29	The DISPLAY action
13, 14, 30	The BEEP action
20	Whether the message is from MVS
24	Whether the message is an action message, such as a WTOR
47	Whether automation vector extensions exist
48	Whether presentation vectors exist in data buffers

Refer to the *IBM Tivoli NetView for z/OS Programming: Assembler* for a description of all bits.

The value of IFRAUTA1 evaluates to null ('') if all bits are B'0'. You can test for this condition by comparing to the null ('') keyword.

Maximum length: 48 bits

Type: Both

IFRAUWF1 [(*pos* **[***len***])]**
Indicates the AIFR fields IFRAUWF1, IFRAUWF2, IFRAUWF3, and IFRAUWF4, mapped in DSIIFR, which contain 32 bits of MVS WTO information.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Meaning
6	Whether the message is a WTOR
7	Whether the message was suppressed
8	Whether the message was broadcast to all
9	Whether the job name is to be displayed
10	Whether the status is to be displayed
14	Whether the session is to be displayed

The value of IFRAUWF1 evaluates to null ('') if all bits are B'0'. You can test for this condition by comparing to the null ('') keyword.

Maximum length: 32 bits

Type: Message

INTERVAL(*occurrence_number***)**
Returns an indication of whether this condition item has been evaluated against a multiple of *occurrence_number* times. Use this condition item for specifying actions to take place when a condition occurs periodically. The *occurrence_number* parameter specifies the interval to be checked. The value can be 1–1000000000.

The values returned by INTERVAL are:

1 Indicates that the condition item being evaluated is a multiple of *occurrence_number*

0 Returned for all other occurrences

For example, INTERVAL(5) returns a value of 1 when the condition item is evaluated for the fifth occurrence, the tenth occurrence, the fifteenth occurrence, and so on, and returns a value of 0 for every other occurrence.

The count of evaluations is incremented only if the INTERVAL condition item is reached during the sequential search for matches through the automation table. The count of evaluations is not incremented if one of these is true:

- The statement with the INTERVAL condition item is not reached because of a prior statement match in the table.
- The BEGIN-END conditional logic that resulted in the statement not being evaluated.
- A prior condition in the automation-table statement that is linked with the logical-AND (&) operator evaluates as false.

Maximum length: 1 bit

Type: Both

Example: Statement evaluated by the INTERVAL keyword

```
IF MSGID = 'XYZ123I' &
     INTERVAL(5) = '1' THEN
       <actions>;
```

In this example, the evaluation count is incremented only if the sequential search through the active automation table reaches the statement and the message ID is XYZ123I. The automation actions are done only for the 5th, 10th, 15th (and so on) XYZ123I messages that reach this statement.

Usage notes for INTERVAL::

1. **Choosing a useful interval value** - The NetView program increments the evaluation count before determining whether the count has reached an interval multiple. Do not specify an interval value of 1 because the condition item always evaluates the same. For example, the statement `INTERVAL(1) = '1'` is always true.
2. **Reset of the evaluation count** - The evaluation count is reset to 0 if any of these events occur:
   - The active automation table is replaced using the AUTOTBL command.
   - The NetView automation-table function is turned off.
   - The NetView program ends.

JOBNAME [(*pos* [*len*])]

The name of the MVS job where the received message originated. JOBNAME is a 1-8 character name.

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 refers. The names can differ when MVS issues a message about the NetView program job. If the message is issued during job start-up or shutdown, JOBNAME can contain the name of an initiator (instead of the actual job name), and you must extract the job name from the message text rather than from the JOBNAME keyword.

The same information is available with the MSGCOJBN condition item.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of JOBNAME evaluates equal to null ('') if the message was not received from an MVS system, or has no associated job name. You can test for these cases by comparing to the null ('') keyword.

Maximum length: 8 characters

Type: Message

JOBNUM [(*pos* [*len*])]

The number assigned by MVS to the job where the received message originated. JOBNUM is an 8-character number that can include an alphabetic prefix and embedded blanks.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of JOBNUM evaluates to equal to null ('') if the message was not received from MVS, or has no associated job number. You can test these cases by comparing to the null ('') keyword.

Maximum length: 8 characters

Type: Message

KEY [(*pos* **[***len***])]**
The key associated with a message. KEY might contain nondisplayable values.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

KEY has a value only if the message was originally a message data block (MDB).

Maximum length: 8 characters

Type: Message

LINEPRES [(*pos* **[***len***])]**
Contains the values for four presentation attributes:
- Alarm control
- Color
- Highlighting
- Intensity

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

LINEPRES is a 4-byte value taken from the first buffer of a message or MSU.

If the value for LINEPRES is not null, and bit 16 in LINETFLG is set on, the LINEPRES values are used for message and MSU presentation. These LINEPRES values are taken from one of two sources:
- The presentation overrides specified in the message data buffer (MDB)
- The presentation overrides as specified by a previous automation-table action

When one or more presentation attributes are set by the automation table (with the COLOR, HIGHINT, or XHILITE actions), all four of the presentation attributes for the message or MSU are copied to the LINEPRES fields and used to display that message or MSU. Attributes that are not set by the automation table are taken from MDB override fields, the fields in MSGGFGPA, or the values specified with the OVERRIDE or DEFAULTS SCRNFMT commands.

If LINEPRES is null, the presentation attributes of the message or MSU are taken from one of three other sources:
- The fields in MSGGFGPA
- The values specified with the OVERRIDE or DEFAULTS SCRNFMT command
- For MSUs, the hardware monitor defaults

Even if LINEPRES is null, other presentation attributes apply to this message when it is displayed. When LINEPRES is null, you can check the fields in MSGGFGPA for presentation attributes.

The four LINEPRES characters have these meanings and possible values:

Byte **Description**

1 Control field

Value	Meaning
80	MVS alarm-on indicator
00	MVS alarm-off indicator

Byte 1 is an MVS indicator. The NetView program does not use it. The NetView program indicators that control this alarm are in IFRAUTA1 bits 13, 14, and 30. The IFRAUTA1 alarm indicators can be set by the BEEP action in the automation table.

2 Color field

Value	Meaning
00	The foreground color
F0	Presentation background. Black on display, white on printer.
F1	Blue
F2	Red
F3	Pink (magenta)
F4	Green
F5	Turquoise (cyan)
F6	Yellow
F7	Presentation neutral. White on display, black on printer.

This field can be set by the COLOR action in the automation table. If the value is 00, the specific foreground color is determined by the fields in MSGGFGPA or the values specified by the OVERRIDE or DEFAULTS SCRNFMT commands.

3 Highlighting field

Value	Meaning
00	No highlighting
F1	Blinking
F2	Reverse video
F4	Underscore

This field can be set by the XHILITE action in the automation table.

4 Intensity field

Value	Meaning
E4	Normal intensity
E8	High (bright) intensity

This field can be set by the HIGHINT action in the automation table.

Maximum length: 4 bytes

Type: Both

LINETFLG [(*pos* **[***len***])]**
Is a 16-bit value taken from the first text buffer of any message or MSU.

Bit 16 of LINETFLG indicates whether the presentation attributes described in LINEPRES apply to this message or MSU. These are the values for bit 16:

Value	Meaning
0	Attributes returned by LINEPRES do not apply to the message or MSU.
1	Presentation attributes have been set to override the attributes in MSGGFPA, and do apply to this message or MSU.
pos	The position where the comparison begins. The default is 1.
len	The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 16 bits

Type: Both

MCSFLAG [(*pos* [*len*])]
The 16-bit MVS multiple console support flag.

pos	The position where the comparison begins. The default is 1.
len	The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Meaning
2	The message is to be queued to the console if it is active
3	The message is a command response WTO
5	The message is a reply to a WTOR
6	The message is to be broadcast to all active consoles
7	The message is to be queued to hardcopy only
8	The message is to be queued unconditionally to the console
9	The message is not to be time-stamped
14	The message is not to be queued to hardcopy

The MCSFLAG values in REXX, high-level language (HLL), and NetView command list language (CLIST) return only eight of the possible 16 bits for MCSFLAG. The automation-table condition item MCSFLAG returns all 16 bits. Table 8 shows the difference between the automation-table condition item and the REXX, command list, and HLL variables. Do not use the bits that are not described for the automation table.

Table 8. The MCSFLAG Condition Item Compared to REXX, Command List, and HLL

Bit	MCSFLAG Condition Item	REXX, CLIST, and HLL
1		REG0
2	REG0	QREG0
3	RESP	RESP
4		REPLY
5	REPLY	BRDCST
6	BRDCST	HRDCPY
7	HRDCPY	NOTIME
8	QREG0	NOCPY

Table 8. The MCSFLAG Condition Item Compared to REXX, Command List, and HLL (continued)

Bit	MCSFLAG Condition Item	REXX, CLIST, and HLL
9	NOTIME	
10		
11		
12		
13		
14	NOCPY	
15		
16		

The value of MCSFLAG evaluates to null ('') if all bits are B'0'. You can test for this case by comparing to the null ('') keyword.

Maximum length: 16 bits

Type: Message

MSGAUTH [(*pos* [*len*])]

Indicates whether a message was issued from an authorized program. MSGAUTH is a two-bit indicator. The compare item is a bit string.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Values for MSGAUTH are:

B'00' The message is not from MVS
B'01' Not used
B'10' A WTO from an unauthorized program
B'11' A WTO from an authorized program

The value of the first bit of MSGAUTH evaluates to null ('') if the message is not from MVS. The value of the second bit evaluates to null ('') if the message is from an unauthorized MVS program. The value of both bits evaluates to null if the message is not from MVS. You can test for these cases by comparing to the null ('') keyword.

Maximum length: 2 bits

Type: Message

MSGCATTR [(*pos* [*len*])]

Indicates the MVS message-attribute flags. MSGCATTR is a 16-bit field.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Meaning
1	The message was suppressed
2	The message is a command response

3 The message was issued by an authorized program

4 The message is to be retained by the automation message retention facility (AMRF)

MSGCATTR has a value only if the message was originally a message data block (MDB).

Maximum length: 16 bits

Type: Message

MSGCMISC [(*pos* **[***len***])]**
Indicates the MVS miscellaneous routing flags. MSGCMISC is an 8-bit field.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Meaning
1	Whether undeliverable messages are to be displayed
2	Whether only undeliverable messages are to be displayed
3	Whether messages are to be queued by ID only
4	Whether the message has been marked in the message processing facility (MPF) table as eligible for NetView program automation

MSGCMISC has a value only if the message was originally a message data block (MDB).

Maximum length: 8 bits

Type: Message

MSGCMLVL [(*pos* **[***len***])]**
Indicates the MVS message-level flags. MSGCMLVL is a 16-bit field. The compare item is a bit string.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Meaning
1	A WTOR
2	An immediate action message
3	A critical eventual action message
4	An eventual action message
5	An informational message
6	A broadcast message

MSGCMLVL only has a value if the message was originally a message data block (MDB).

Maximum length: 16 bits

Type: Message

MSGCMSGT [(*pos* [*len*])]
Indicates the MVS message-type flags. MSGCMSGT is a 16-bit field. These bits apply to messages displayed on an MVS console; these bits are not used by the NetView program.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check these bits:

Bit	Meaning
1	Job names are to be displayed
2	Status is to be displayed

MSGCMSGT only has a value if the message was originally a message data block (MDB).

Maximum length: 16 bits

Type: Message

MSGCOJBN [(*pos* [*len*])]
Indicates the originating job name. MSGCOJBN is a name that contains 1-8 characters. (The same information is available with the JOBNAME condition item.)

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

MSGCOJBN only has a value if the message was originally a message data block (MDB).

Maximum length: 8 characters

Type: Message

MSGCPROD [(*pos* [*len*])]
Indicates the MVS product level. MSGCPROD is a 16-character string consisting of a 4-character MVS control program object version level, a 4-character control program name *MVS*, and an 8-character identifier for the originating system.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

MSGCPROD has a value only if the message was originally a message data block (MDB).

Maximum length: 16 characters

Type: Message

MSGCSPLX [(*pos* [*len*])]
The name of the MVS SYSPLEX where the received message originated.

MSGCOJBN is a name that contains 1–8 characters. The *pos* parameter is the position where the comparison begins and has a default value of 1. The compare item is a parse template.

The value of MSGCSPLX evaluates to equal to null ('') if the message was not received from an MVS SYSPLEX, has no associated SYSPLEX name, or the message was not originally a message data block (MDB). You can test these cases by comparing them to the null ('') keyword.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Message

MSGDOMFL [(*pos* **[***len***])]**
Indicates the MVS DOM flags. MSGDOMFL is an 8-bit field.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Check the following bits:

Bit	Meaning
1	A DOM by message ID (MSGID)
2	A DOM by system ID (SYSID)
3	A DOM by the NetView address-space ID (ASID)
4	A DOM by a job step TCB
5	A DOM by a token

MSGDOMFL has a value only if the message was originally a message data block (MDB).

multiple console support consoles are set up by default as DOM(NORMAL) receivers. As a result, the DOMs that are received from MVS by these consoles have a flag in bit 1. The SYSID, ASID, TCB, and TOKEN bit flags are not usually set on when the DOM is received from an MVS program.

Maximum length: 8 bits

Type: Message

MSGGBGPA [(*pos* **[***len***])]**
Indicates the background presentation attributes. MSGGBGPA is a 4-byte hexadecimal value. The compare item is a parse template.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

These are the byte descriptions:

Byte	Meaning
1	Background control
2	Background color
3	Background highlighting
4	Background intensity

For a description of the values for each byte, see "LINEPRES" on page 186.

MSGGBGPA has a value only if the message was originally a message data block (MDB).

Maximum length: 4 bytes

Type: Message

MSGGDATE [(*pos* **[***len***])]**
The date that the message originator placed in the MDB. MSGGDATE is a 7-character date in the form *yyyyddd* where *yyyy* is the year and *ddd* is the day of the year.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

MSGGDATE only has a value if the message was originally a message data block (MDB).

Maximum length: 7 characters

Type: Message

MSGGFGPA [(*pos* **[***len***])]**
Indicates the foreground presentation attributes. MSGGFGPA is a 4-byte hexadecimal value. The compare item is a parse template.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The byte descriptions are:

Byte	Meaning
1	Foreground control
2	Foreground color
3	Foreground highlighting
4	Foreground intensity

For a description of the values for each byte, see "LINEPRES" on page 186.

MSGGFGPA has a value only if the message was originally a message data block (MDB).

Maximum length: 4 bytes

Type: Message

MSGGMFLG [(*pos* **[***len***])]**
Indicates the MVS general message flags. MSGGMFLG is a 16-bit field. Bit 1 indicates a DOM. Do not test other bits.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

MSGGMFLG has a value only if the message was originally a message data block (MDB).

Maximum length: 16 bits

Type: Message

MSGGMID [(*pos* [*len*])]
The 4-character MVS message identifier. MSGGMID might contain nondisplayable characters.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

MSGGMID has a value only if the message was originally a message data block (MDB).

Maximum length: 4 characters

Type: Message

MSGGTIME [(*pos* [*len*])]
The time MVS associates with the message. MSGGTIME is 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 hundredths of seconds.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

MSGGTIME has a value only if the message was originally a message data block (MDB).

Maximum length: 11 characters

Type: Message

MSGID [(*pos* [*len*])]
The message identifier of the received message. MSGID is a 1–255 character ID. The message identifier is usually the first token of the message. If a REPLYID is sent with the message, the REPLYID is not used as the first token.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 255 characters

Type: Message

MSGSRCNM [(*pos* [*len*])]
Indicates the 1-17 character source name.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

This source name is an identifier from the source object which was provided by either the DSIMMDB or CNMPMDB application programming interface (API) invocation.

For more information about DSIMMDB, refer to *IBM Tivoli NetView for z/OS Programming: Assembler*. For more information about CNMPMDB, refer to *IBM Tivoli NetView for z/OS Programming: PL/I and C*.

The source name is selected from the source object by these rules:

1. The first alias, if any
2. The first network identifier concatenated to a network addressable unit (NAU) name, with a period (.) between them, if both exist in sequence
3. The first existing NAU name
4. The string N/A, if none of the other names in this list are specified in the source object
5. Null, if there is no source object

For more information about how the source object is defined and the DSIAIFRO mapping, refer to *IBM Tivoli NetView for z/OS Programming: Assembler*.

Note: This function has a value only if the message was originally an MDB with an associated source object.

Maximum length: 17 characters

Type: Message

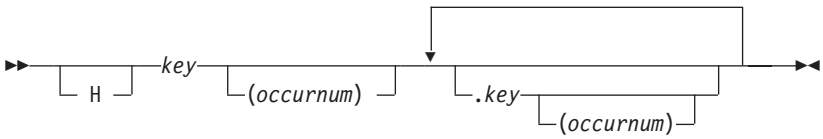
MSUSEG (*location* **[***byte* **[***bit***]])**

Indicates the contents of one segment of an MSU. The compare item can be a bit string or a parse template.

location

The location of the data to be compared. The syntax for the parameter is:

Location Parameter



H For an MDS-MU, indicates that the first *key* is to be obtained at the MDS-MU level, rather than the major-vector level. If you use this parameter and the MSU being processed is not an MDS-MU, MSUSEG returns a value of null.

key The 2-character or 4-character representation of the 1-byte or 2-byte hexadecimal ID of the generalized data stream (GDS) variable or key of the major vector, subvector, subfield, or sub-subfield.

You can use more than one *key*, separating them with periods. Each additional *key* specifies a lower-level structure within the structure identified by the preceding *key*.

occurnum

The occurrence number, counting from 1, of the GDS variable, major vector, subvector, subfield, or sub-subfield. An asterisk (*) means you want any occurrence. For example, used at the subvector level, an *occurnum* of 2 means you want the second instance of the *key* subvector. An *occurnum* of * means you want the first subvector with

a key of *key*, if any, that results in equality with the compare item you have specified. The maximum *occurnum* is 32767, and the default is 1.

byte The byte position within the lowest *key* specified in *location*. A position of 1, not a 0, designates the first byte. The maximum is 32767, and the default is 1.

bit The bit position within the byte specified by *byte*. The bit position can be any number from 1 to 8. Note that a position of 1, not a 0, designates the first bit. If you specify a bit position, the compare item is a bit string. Otherwise, the compare item is a parse template.

MSUSEG does not support a length specification. You can assign MSUSEG to a variable, and then use that variable (including *pos* and *len*) in a VALUE conditional statement.

Maximum length: Varies

Type: MSU

Usage notes:

1. See "Writing Automation Table Statements to Automate MSUs" on page 322 for examples of how to use MSUSEG.
2. The MSUSEG automation-table statement is not interchangeable with REXX's MSUSEG function or the NetView command list language's &MSUSEG control variable. The formats for specifying an MSU location are similar, but other syntax details vary.

MVSLEVEL [(*pos* [*len*])]

The 8-character string that identifies the level of MVS that is currently running.

You can use the LISTVAR command to determine the MVS level on your system.

In contrast to the MVSLEVEL condition item, the MSGCPROD condition item identifies the system level of MVS that the message came from.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of MVSLEVEL is null if the currently running system is not MVS.

Maximum length: 8 characters

Type: Both

NETID [(*pos* [*len*])]

Indicates the VTAM network identifier. This field has a maximum length of 8 characters.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

If VTAM has never been active when the NetView program is active, the value of NETID is null.

Maximum length: 8 characters

Type: Both

NETVIEW [(*pos* **[***len***])]**

Indicates the version and release of the currently running NetView program. The value of NETVIEW is a 4-character field in the form NV*vr* where *v* is the version number and *r* is the release number.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 4 characters

Type: Both

NUMERIC (*variable***[***pos* **[***len***]])**

Indicates a *variable* to convert from a text value to a numeric value and to compare to the numeric value of the literal specified in the parse template.

pos The position where the text to convert to a numeric begins within the variable value. The default value is 1.

len The length of the text value to convert to a numeric. This value can be positive or negative; decimal points are not supported. The default value is the remaining portion of the variable beginning with *pos*.

Maximum length: 255 characters

Type: Both

NVCLOSE [(*pos* **[***len***])]**

Indicates whether the NetView program is currently performing CLOSE processing. It is a 1-bit indicator. Values for NVCLOSE are as follows:

1 The NetView program is performing CLOSE processing

0 The NetView program is not performing CLOSE processing

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Note: Use the NVCLOSE check with caution. If NVCLOSE is used as the only condition item on an automation statement, a looping condition can occur. The intent is to use the NVCLOSE condition item in conjunction with other condition items as shown in this example.

Maximum length: 1 bit

Type: Both

Example:

If you use MYCMD to restart the task referred to in the DSI008I message, this automation statement can prevent attempts to restart tasks during CLOSE processing.

```
IF MSGID = 'DSI008I' & NVCLOSE ¬= '1' THEN
  EXEC(CMD('MYCMD')ROUTE(ONE AUTO1));
```

Usage notes for NVCLOSE::

1. The value of NVCLOSE evaluates to null ('') when CLOSE processing is not currently running. You can test for this case by comparing to the null ('') keyword.
2. If you have automation running on the PPT task, which determines whether tasks are active, and if the NetView program is using CLOSE STOP processing, a loop can occur. This loop can prevent the NetView program from completing CLOSE STOP processing. For example, if PPT is issuing EXCMD for various tasks, the NetView program does not end until the PPT has completed the task.

NVDELID [(*pos* [*len*])]

Indicates a 24-character EBCDIC value for a message that can be used as input by the DOM command to delete an action message.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 24 characters

Type: Message

OPID [(*pos* [*len*])]

Indicates the operator or task ID under which the automation table is processing. OPID is a 1-8 character ID.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Both

OPSYSTEM [(*pos* [*len*])]

Indicates the operating system for which the NetView program was compiled. This field has a maximum length of 7 characters.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 7 characters

Type: Both

ROUTCDE [(*pos* [*len*])]

Identifies one or more MVS routing-code bits assigned to the message. A message can have up to 128 routing-code bits.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Refer to the MVS library for information about code values.

Maximum length: 128 bits

Type: Message

SESSID [(*pos* [*len*])]
Indicates the 1-8 character identifier of the NetView terminal access facility (TAF) session that sent the received message.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of SESSID is a string of hexadecimal zeros (X'00') if the message did not come over a TAF session. You can test for this case by comparing to the null ('') keyword.

Maximum length: 8 characters

Type: Message

SYSCONID [(*pos* [*len*])]
Specifies the MVS system console name associated with the message. System console names are from 1 to 8 characters in length.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 8 characters

Type: Message

SYSID [(*pos* [*len*])]
Indicates the 1-8 character identifier of the MVS system that sent the message.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

One use for SYSID is in a sysplex. You can add SYSID to your existing automation-table statements to block messages from certain systems, or to invoke certain automation-table actions based on the system ID.

This example shows how you can use SYSID to process messages local to your system, whether the message originated from MVS or not. In the example, the local system name is SYSA:

```
IF (SYSID = 'SYSA' | SYSID = '') THEN
  BEGIN;
    .
    .
    .
  END;
```

Messages originating from MVS on system SYSA satisfies the check for SYSID because they have a SYSID value equal to 'SYSA'. Messages that did not originate from MVS but are local to system SYSA also match because they have a SYSID equal to null.

Maximum length: 8 characters

Type: Message

SYSPLEX [(*pos* [*len*])]
Identifies the name of the MVS SYSPLEX where the received message is being automated. SYSPLEX is a 1-8 character name.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of SYSPLEX evaluates to equal to null ('') if the message is not being automated on an MVS SYSPLEX or has no associated SYSPLEX name. You can test these cases by comparing to the null ('') keyword.

Maximum length: 8 characters

Type: Both

TASK [(*pos* **[***len***])]**
Specifies the type of task under which the automation table is processing. TASK is a 3-character string.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The values for TASK are:

HCT A hardcopy task.

DST A data services task. DST is an optional task that has MOD=DSIZDST specified in the CNMSTYLE member.

OPT An optional task. The CNMCSSIR task always evaluates to a value of OPT.

OST An operator station task. Automation tasks evaluate to a value of OST. You can use the AUTOTASK and DISTAUTO condition items to distinguish autotasks from other OSTs.

NNT A NetView-NetView task.

MNT The NetView program main task.

PPT The primary POI task.

Maximum length: 3 characters

Type: Both

TEXT [(*pos* **[***len***])]**
Specifies the text of the received message. TEXT is a 1-25 character string that contains the entire message text, including the MSGID.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The compare item is a parse template.

Maximum length: 255 characters

Type: Message

THRESHOLD(*occurrence_number* **[***time_period***])**
Returns an indication of whether the threshold condition item has been evaluated against at least *occurrence_number* of times during the prior *time_period*. The THRESHOLD condition item is useful for specifying particular actions to take place when a condition has happened at least a

specified number of times within a specified time period. See Figure 60 on page 233 for an example of these occurrence-detection condition items:

occurrence_number
: Specifies the number of occurrences within the specified time period that cause the threshold condition to be reached. The value can be 1–1000.

time_period
: Specifies the time interval of the threshold. The default is 24 hours. The time period is specified as *ddd hh:mm:ss*, where:

  ddd
  : The number of days in the range of 0–365. If you specify *ddd*, you must also specify *hh:mm:ss*.

  hh:mm:ss
  : The hours (ranging 00–23), minutes (ranging 00–59), and seconds (ranging 00–59).

  You cannot specify a time period of zero. If you specify only one numeric value for *time_period*, without any colon delimiters (:), the NetView program assumes it to be a value for minutes.

Table 9 shows examples of valid THRESHOLD specifications.

Table 9. Examples of Valid THRESHOLD Specifications

SPECIFICATION	OCC.#	DAYS	HOURS	MIN.	SEC.
`THRESHOLD(3) = '1'`	3	1			
`THRESHOLD(4 1 00:00:00) = '1'`	4	1			
`THRESHOLD(5 1:00) = '1'`	5		1		
`THRESHOLD(6 0 1:00:00) = '1'`	6		1		
`THRESHOLD(7 10) = '1'`	7			10	
`THRESHOLD(8 :30) = '1'`	8				30
`THRESHOLD(9 10 10) = '1'`	9	10		10	
`THRESHOLD(10 10 00:10:00) = '1'`	10	10		10	

Unless you are accepting the default time period of 24 hours (one day), specify all the elements of *time_period* (days, hours, minutes, and seconds), even though they are not required, to avoid any misunderstanding of what the time period is.

The values returned by THRESHOLD are:

1
: Indicates the number of occurrences within the specified time period is equal to or greater than the value of *occurrence_number*

0
: Returned for all other occurrences

The count of evaluations is incremented only if the THRESHOLD condition item is reached during the sequential search (for matches) of the automation table. The count of evaluations is not incremented if one of these situations is true:

- The statement with the THRESHOLD condition item is not reached because of a prior statement match in the table.
- The BEGIN-END conditional logic that resulted in the statement was not evaluated.
- A prior condition in the automation-table statement that is linked with the logical-AND (&) operator evaluates to false.

In Figure 32, the evaluation count is incremented only if the sequential search through the active automation table reaches this statement *and* if the message ID is XYZ123I.

The automation actions are done only for the fifth (or more) XYZ123I message that reaches this statement during the automation table search for any 3-hour time period.

```
IF MSGID = 'XYZ123I' &
   THRESHOLD(5 0 3:00:00) = '1' THEN
     <actions>;
```

Figure 32. Statement Evaluated by the THRESHOLD Keyword

Maximum length: 1 bit

Type: Both

Usage notes for THRESHOLD::

1. **Elapsed time and the time period** - For every evaluation of the THRESHOLD condition, the number of occurrences during the prior time period (specified by *time_period*) is examined to see if the threshold has been reached. As time passes, prior occurrences might no longer be within *time_period*.
2. **Choosing a useful occurrence value** - The NetView program increments the evaluation count before determining whether the threshold has been reached. Do not specify an occurrence value of 1 because the condition item always evaluates the same. For example, the condition item `THRESHOLD(1 x x:xx:xx) = '1'` is always true.
3. **Reset of the evaluation count** - The evaluation count is reset to 0 if any of these events occur:
   - The active automation table is replaced using the AUTOTBL command.
   - The NetView automation-table function is turned off.
   - The NetView program is ended.
4. **Defining limits on actions** - You can define an ending occurrence number (a point at which you no longer want to take a certain action) by combining two THRESHOLD condition items on one statement. If, for example, you want certain actions to occur only on the 3rd through 6th occurrence of message XYZ123I within any one-hour time period, you can use this automation-table statement:

   ```
   IF MSGID = 'XYZ123I' &
      THRESHOLD(3 0 01:00:00) = '1' &
      THRESHOLD(5 0 01:00:00) = '0' THEN
          <actions>;
   ```

   The second THRESHOLD condition item in the statement is evaluated only after the first threshold is met (starting with the third occurrence of the XYZ123I message within a one-hour period). The fifth evaluation of the second THRESHOLD condition item is the seventh occurrence of the XYZ123I message.
5. **Processing of immediate messages** - If the automation table evaluates THRESHOLD for a NetView message sent to the immediate message area of the operator's screen (that is, if TVBINXIT is on), the THRESHOLD occurrence count is not incremented, and the condition evaluates as false.

TOKEN [(*token-number* **[***pos* **[***len***]])]**
Indicates a particular word or phrase within the message. The NetView

program uses the blank spaces between words and phrases to divide a message into tokens. A token consists of all the characters between two nonadjacent blank spaces. The compare item is a parse template.

token-number The number of the token you want to compare. It must have a numeric value; the default value is 1.

pos Indicates the position, within the specified token where comparison begins. The default value is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 255 characters

Type: Message

VALUE (*variable* **[***pos* **[***len***]])**

Indicates the name of the variable whose value is to be used in a comparison.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

Maximum length: 255 characters

Type: Both

VTAM [(*pos* **[***len***])]**

Indicates the version and release of VTAM. VTAM is a 4-character string in the form VT*vr* or V*vrm*, where *v* is the version number, *r* is the release number, and *m* is the modification number.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of VTAM evaluates to null ('') when VTAM is inactive. You can test for this case by comparing to the null ('') keyword.

Maximum length: 4 characters

Type: Both

VTCOMPID [(*pos* **[***len***])]**

Indicates the VTAM component identifier. VTCOMPID is a 14-character string.

You can use the LISTVAR command to determine the VTAM component identifier.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The value of VTCOMPID evaluates to null ('') when VTAM is inactive. You can test for this case by comparing to the null ('') keyword.

Maximum length: 14 characters

Type: Both

WEEKDAYN [(*pos* **[***len***])]**

Is a numeric value 1–7 representing the day of the week.

pos The position where the comparison begins. The default is 1.

len The length of the string to be compared. The default value is the remaining portion of the string beginning with *pos*.

The possible character values for WEEKDAYN are:

1 Monday
2 Tuesday
3 Wednesday
4 Thursday
5 Friday
6 Saturday
7 Sunday

Maximum length: 1 character

Type: Both

Bit Strings as Compare Items

Compare items that you can use in an IF-THEN statement include bit strings and parse templates.

A bit string is either a sequence of one or more bits to be compared, or a null. Enclose a bit string in single quotation marks. The string can have any combination of the values 1, 0, and X:

0 Tells the NetView program to check for a value of B'0'
1 Tells the NetView program to check for a value of B'1'
X Tells the NetView program not to check the value of the bit

For example, if you check for a bit string of 0X1, the bit strings 011 and 001 both match.

The example in Figure 33 tells the NetView program that when the message routing code has the bits 10011 starting in position 3 or when the message descriptor code has the bits 110 starting in position 6, the message is to be routed to operators whose identifiers are OPER1, OPER4, and OPER6, and an audible alarm is to be sounded when the message is displayed.

```
IF ROUTCDE(3) = '10011' | DESC(6) = '110' THEN
      EXEC (ROUTE(ALL OPER1 OPER4 OPER6)) BEEP(Y);
```

Figure 33. Example of Comparing Bits

The NetView program compares a compare-item bit string to a bit-string *of equal length* taken from the condition item, starting with the position you indicate (the default is to start with position 1). If there are not enough bits available in the condition item, an error results. For example, the statement in Figure 34 directs the NetView program to compare 10X10 to bits 14 through 18 of the descriptor codes. Because descriptor codes can only contain 16 bits, an error results.

```
IF DESC(14) = '10X10' THEN ...
```

Figure 34. Example of Comparing Bits of Unequal Length

A bit string of null ('') works differently, and its function depends on the condition item. DESC and ROUTCDE equal null if all of the bits (beginning with the position

you specify, if any) are zero. In Figure 35, the comparison is true if all of the DESC bits are zeros.

```
IF DESC = '' THEN ...
```

Figure 35. Example of Comparing Null Bit Strings

For MSUSEG and ATF, null bit strings work like null parse templates. MSUSEG is null if the location you specify does not exist in the MSU being processed. ATF is null if the ATF you call sends back a compare item with a length or zero. The precise meaning of a zero-length compare item depends on the ATF. DSICGLOB and DSITGLOB, which are the ATFs that are supplied with the NetView program, give a value of null if you request a global variable to which you have not yet assigned a value, or to which you have assigned a value of null.

If the NetView program is not using multiple console support consoles, the condition items that have values only if the message was originally a message data block evaluate to null for MVS system messages. See “Condition Items” on page 157 for a list of these condition items. These condition items can have a value if the MDB was received from the CNMPMDB or DSIMMDB application programming interface.

Parse Templates as Compare Items

For a parse template, you can use any combination of literals, variable names, variable values, and placeholders. Alternatively, you can use a null.

Literals

A literal indicates that you want to compare to a specified string. A literal is either a character or a hexadecimal string. The maximum length for a literal is 255 characters. The maximum length for a hexadecimal literal is 255 hexadecimal digits.

Try to keep literals on one line. If you use more than one line for a single literal, do not indent the continuation lines. End each line in column 72, and begin each continuation line in column 1.

You can continue a literal compare item by breaking it into smaller literal compare items on several lines. Consecutive literal compare items are concatenated without extra blanks, so you do not have to end the lines in column 72 and begin them in column 1. For example, you can continue a literal compare item on several lines, as shown in Figure 36.

```
IF TEXT='PURGE DATE IS LATER '
   'THAN TODAY''S DATE'
```

Figure 36. Example of a Multiline Literal Compare Item

A ***character literal*** is a string of alphanumeric characters enclosed in single quotation marks.

For the DSI146I message, the example in Figure 37 compares the sixth token starting at the 5th character to the character literal `AUTO`.

```
IF MSGID = 'DSI146I' & TOKEN (6 5) = 'AUTO' THEN
      EXEC(ROUTE(ALL * OPER1));
```

Figure 37. Example of Comparing Character Literals

When a single quotation mark is part of a character literal, you must code a second single quotation mark after the first, as shown in Figure 38.

```
'PURGE DATE IS LATER THAN TODAY''S DATE'
```

Figure 38. Example of Using Single quotation marks in a Character Literal

You can use system symbolics as a character literal, as shown in Figure 39.

```
IF DOMAINID = '&DOMAIN' & MSGID = 'DSI146I' THEN
      EXEC(ROUTE(ALL * AUTO1));
```

Figure 39. Example of Using System Symbolics as a Character Literal

A ***hexadecimal literal*** is a string of hexadecimal digits enclosed in single quotation marks within a HEX() keyword, such as `HEX('A1')`. The single quotation marks distinguish a hexadecimal literal from a hexadecimal variable.

If you specify an odd number of hexadecimal digits, the NetView program adds a leading zero. For example, the NetView program interprets `HEX('1AB')` the same as `HEX('01AB')`.

Variable Names

Variable names designate parts of a message or MSU that you want the NetView program to ignore (when doing the comparison), but to store those parts for use during action processing. You can use the stored variables as command string parameters on an EXEC action with CMD.

During comparison processing, the NetView program ignores any parts designated by variable names and stores each part ignored in the variable name you specify. After setting variables in the IF part of an IF-THEN statement, you can use them in the THEN part of the statement or within a BEGIN-END section for that IF-THEN statement.

A variable name can have up to 16 alphanumeric characters. However, the first character cannot be numeric. You can code up to 25 variable names in an IF-THEN statement, using any names that are not automation-table functions, actions, or keywords. Do not use the same variable name more than once in any one IF condition.

After you define a variable in the IF part of an IF-THEN-BEGIN structure, the variable maintains its value throughout the BEGIN-END section. However, an individual IF-THEN statement within the section can temporarily redefine the value of the variable for its own use by making a comparison to the same variable name.

A variable name can be either character or hexadecimal. A ***character variable name*** is one whose value is a set of characters.

The IF-THEN statement in Figure 40 contains the character variable name DATEVAR.

```
IF DOMAINID='CNM01' &
   TEXT='DATABASE HASN''T BEEN PURGED SINCE' DATEVAR THEN
      EXEC (CMD('CLISTA ' DATEVAR) ROUTE (ALL OPERA OPERB));
```

Figure 40. Example of Using a Character Variable Name

If the NetView program receives the message DATABASE HASN'T BEEN PURGED SINCE 12/3/10, the NetView program puts the value of the text following the word SINCE, which is 12/3/10, into the variable DATEVAR. Then the NetView program runs CLISTA under both OPERA and OPERB using the value of DATEVAR as a parameter.

The IF-THEN statement in Figure 41 contains the variable name DOMID.

```
IF TEXT='PURGE DATE IS LATER THAN TODAY''S DATE' &
   DOMAINID=DOMID THEN
      EXEC (CMD('CLISTA ' DOMID) ROUTE (ALL OPERA OPERB));
```

Figure 41. Example of Using Character Variable Name DOMID

If the NetView program receives the message PURGE DATE IS LATER THAN TODAY''S DATE from domain CNM01, the statement says to put the value of DOMAINID, which is CNM01, into the variable DOMID and run CLISTA under both operators OPERA and OPERB using the variable DOMID as a parameter.

A ***hexadecimal variable name*** is one whose value is the hexadecimal representation of the data you assign to it. Specify a hexadecimal variable name with the HEX() keyword.

The example is Figure 42 extracts the generic alert data from an MSU in the variable GENERICDATA. The examples pass the data to a POWEROUT command list in hexadecimal format. Unlike the hexadecimal literal HEX('14'), the hexadecimal variable HEX(GENERICDATA) does not have single quotation marks.

```
IF MSUSEG(0000.92 6) = HEX('14') .
   & MSUSEG(0000.92) = HEX(GENERICDATA) THEN
      EXEC (CMD('POWEROUT 'GENERICDATA) ROUTE (ONE AUTO1 *));
```

Figure 42. Example of Using a Hexadecimal Variable Name

The NetView program expands the data assigned to GENERICDATA into a string of EBCDIC characters representing hexadecimal digits (0–9 and A–F) before passing the data to the POWEROUT command list.

Variable Values

You can use the value of a variable in a parse template using the VALUE() function. In this case, the value is used rather than being set. A variable that has no value is treated as a NULL literal. The variable must be specified in either the BEGIN block or the IF-THEN statement. A variable cannot be set and subsequently referenced in the same parse template. The variable cannot be subscripted with position or length. The IF-THEN statement in Figure 43 contains a parse template to use the domain name from variable DOM1.

```
IF DOMAINID=DOM1 &
    TEXT = 'WORD1 ' VALUE(DOM1) ' WORD3' THEN
    EXEC (CMD('CLISTA DOM1') ROUTE (ALL OPERA));
```

Figure 43. Example of Using the Value of Variable DOM1

Placeholders

Placeholders cause the NetView program to skip over parts of a message or MSU that you do not want to use in the comparison. Placeholders are similar to variable names, but the NetView program does not store the text skipped by a placeholder for use in an action. Designate a placeholder with a period (.).

If you code a period within single quotation marks, the NetView program treats the period as part of a string, not as a placeholder.

The IF-THEN statement in Figure 44 a placeholder to cause the NetView program to skip over parts of the message text.

```
IF TEXT = . 'SENSE CODE=' SENSE . THEN
      EXEC (CMD('CLIST1 'SENSE) ROUTE (ONE OPERA OPERB));
```

Figure 44. Example of Using a Placeholder

If the NetView program receives the message `RESOURCE LU1 SENSE CODE=08 NOT ACTIVATED`, the statement in Figure 44 directs the NetView program to skip over all of the text preceding `SENSE CODE=`, store the value 08 in the variable name `SENSE`, and skip over all of the text following the variable name SENSE. Without the leading placeholder, the example message does not fulfill the conditions of the statement. Without the trailing placeholder, the variable `SENSE` takes on the value 08 NOT ACTIVATED.

The IF-THEN statement in Figure 45 uses a placeholder to cause the NetView program to select a single character from the message text.

```
IF MSGID = 'IST105I' &
   TOKEN(2 4) = 'A' . THEN
            NETLOG(N);
```

Figure 45. Example of Using a Placeholder to Select a Single Character

If the NetView program receives the message `IST105I A01A425  NODE NOW INACTIVE`, the message identifier and the 4th character of the resource name (the second token) are compared. If the values specified in the automation-table statement match, the message does not appear in the network log.

Nulls

You can use a parse template of null ('') to check if information is absent in a message or MSU. You cannot use the null in conjunction with literals, variable names, or placeholders in a single comparison. You can use the logical-AND (&) and logical-OR (|) operators to join null comparisons with other comparisons.

A parse-template compare item has a value of null if the message or MSU being processed does not have any value for that item. The precise meaning of the null varies from compare item to compare item. A bit string compare item has a value of null if all bits in that bit field have a value of B'0'.

Some condition items have values only if the message was originally a message data block (MDB) (see “Condition Items” on page 157). These condition items evaluate to null for MVS system messages if the NetView program is not using multiple console support consoles. These condition items can have a value if the MDB was received from the CNMPMDB or the DSIMMDB API.

Textual compare items give a value of null if the position you specify is beyond the length of the compare item. For example, TOKEN(8) yields null for a message that has only six tokens. Some compare items in the textual compare category are:

- DOMAINID
- MSGID
- TEXT
- TOKEN

An ATF call gives a value of null if the ATF sends back a compare item with a length of zero. The specific meaning of a zero-length compare item depends on the ATF. DSICGLOB and DSITGLOB, which are the ATFs that are supplied with the NetView program, give a value of null if you request a global variable to which you have not yet assigned a value, or to which you have assigned a value of null.

MSUSEG gives a value of null if the field you specify does not exist in the MSU being processed. HIER gives a value of null if the MSU does not have any resource- hierarchy information or if you specify a name-type pair that the MSU does not have.

Some functions are set only if a message is received from MVS, or from the CNMPMDB or DSIMMDB APIs. Otherwise, these functions give null values. For example:

- JOBNAME
- SYSID
- ROUTCDE
- AREAID

SESSID gives a value of null if the message being processed was not received over a TAF session.

HDRMTYPE does not return null values.

Comparing to null is not always the same as comparing to a string of hexadecimal zeros.

The example in Figure 46 shows how you might use the null ('') keyword to check for the presence of a resolution major vector (key X'0002') within an MSU.

```
IF MSUSEG(0002) ¬= '' THEN
   BEGIN;
⋮
   END;
```

Figure 46. Example of Using Nulls as a Variable

Actions

This section describes the actions that you can use in IF-THEN and ALWAYS statements. Table 10 summarizes these actions. The actions are organized according to type (message, MSU, or both) in the table and alphabetically in the description section.

Table 10. IF-THEN and ALWAYS Actions

Action	Description
Messages	
DISPLAY	Displays the message
DOMACTION	Specifies action for operator message deletion
HCYLOG	Logs the message in hardcopy log
HOLD	Holds the message on the screen
NETLOG	Logs the message in the network log
SYSLOG	Logs the message in the system log

MSUs

SRF	Sets recording-filter attributes for the MSU
XLO	Specifies external logging only

Messages and MSUs

AUTOMATED	Sets the significant action indicator for the AIFR
BEEP	Sounds an audible alarm
CNM493I	Specifies whether CNM493I messages are written to the network log for this statement
COLOR	Sets foreground color
CONTINUE	Continues table processing for the message or MSU
EDIT	Specifies an edit specification that alters an AIFR that is being automated
EXEC	Issues a command or, for messages, controls routing
HIGHINT	Sets high-intensity 3270 mode
TRACE	Sets tracing on
XHILITE	Sets foreground highlighting

AUTOMATED (Yes | No | Ignore)
Sets the specified value of the significant action indicator for any message or MSU that matches the statement. Possible values are as follows:

YES | Y
Forces the AUTOMATED status ON for an AIFR matching this statement, regardless of the actions for this statement, to indicate this AIFR has been automated. This is the default.

NO | N
Forces the AUTOMATED status OFF for an AIFR matching this statement, regardless of the actions, to indicate this AIFR has not been automated.

IGNORE | I
Leaves the indicator in the state prior to this statement.

This indicator can be queried by the AUTOMATED condition item.

Type: Both

BEEP (Y | N)
Determines whether an audible alarm sounds when the message or MSU is displayed. For MSUs, BEEP applies only to alert major vectors displayed by the hardware monitor. If you do not specify a BEEP value in the automation table or elsewhere, the default is BEEP (N) for messages.

See Note 5 on page 226 for information about MSU defaults.

Type: Both

CBE ('*edit_specification*')

The CBE action enables you to use information from a message or MSU to create an XML version of a Common Base Event and sent it to the Common Event Infrastructure server. The XML is constructed using the syntax and functions provided by the CBETEMP global order in the PIPE

EDIT stage. With the CBE action, data from messages and MSUs can be used to fill in values from predefined XML templates; the resulting XML sent to the Common Event Infrastructure server. The original AIFR continues through automation. For information about what you can include in the *edit_specification*, refer to *IBM Tivoli NetView for z/OS Programming: Pipes*. The edit specification does not support variables.

While *edit_specification* must be enclosed in single quotation marks (in the form '*edit_specification*'), you cannot use single quotation marks (' ') within the *edit_specification* itself.

CNM493I (Y | N)
Indicates whether CNM493I messages are written to the network log for this automation-table statement. A CNM493I message is generated and written to the log to serve as an audit trail for a command or command list that is run from the automation table. If you do not specify a CNM493I value in the automation table, the DEFAULTS command, or the OVERRIDE command, the default is for the NetView program to generate CNM493I messages.

Usage notes:

1. Be careful if you specify CNM493I(N) for statements that are not stable and might need the debugging capability provided by CNM493I messages (to indicate when a command or command list has been run from the automation table). In some cases, the function provided by the detailed automation usage report is sufficient to provide information about the number of times a particular command or command list was run from the automation table. In these cases, consider preventing CNM493I messages from being generated. You can do this by:
   - Using CNM493I(N) for particular automation statements
   - Using OVERRIDE CNM493I=NO for a particular NetView program task
   - Using DEFAULTS CNM493I=NO for all NetView programs
2. If you specify the CNM493I action and you do not specify EXEC actions with CMD keywords for a message or MSU, the CNM493I action is ignored and processing continues.

Type: Both

COLOR (BLU | GRE | PIN | RED | TUR | WHI | YEL)
Specifies the foreground color for display on color terminals. Color is set for all lines of an MLWTO, but only the first line of the message is available to the automation table. The hardware monitor uses the specified color if it displays the MSU. The command facility uses the specified color if it displays the message.

See Note 8 on page 227 for information about hardware monitor defaults for MSUs.

BLU Specifies blue
GRE Specifies green
PIN Specifies pink
RED Specifies red
TUR Specifies turquoise
WHI Specifies white
YEL Specifies yellow

Type: Both

CONTINUE (Yes | No | Stop)
Specifies whether messages and MSUs that match the statement must continue through automation-table processing, possibly matching another statement farther down in the current table or another table. Possible values are:

YES | Y
If a match occurs, processing continues to the next statement in the current automation table and subsequent tables, if present.

NO | N
If a match occurs, processing continues with the first statement in the next automation table, if present. This is the default.

STOP | S
If a match occurs, processing ends and no further statements in the current table or subsequent tables, if present, are evaluated.

Type: Both

DISPLAY (Y | N)
Determines whether the NetView program displays the message if the message reaches a task capable of display. If you do not specify a DISPLAY value in the automation table or elsewhere, the default is DISPLAY (Y).

NetView program messages sent to the immediate message area of the operator's screen are always displayed, regardless of the setting for the DISPLAY action.

Type: Message

DOMACTION (A | D | N)
Specifies the type of delete operator message (DOM) processing that the NetView program does with regard to this message. The default value for action messages is DELMSG; the default for other messages is NODELMSG. Action messages are WTORs or those having a Descriptor code matching the setting of MVSPARM.ActionDescCodes in CNMSTYLE. The DOMACTION specification enables you to tailor DOM processing as follows:

A | AUTOMATE
Specifies that a DOM is expected for this message and requests that when a DOM is received for this message, a modified copy of the original message is sent through automation with modified values that identify it as a DOM. In order for this automation to occur, the original message must be held at an operator station or at an autotask. Wherever the message is held, the automation of the DOM occurs. When the DOM is received, automation for the message is redriven. The IFRAUDOM bit is set on and other DOM-related bits are copied from the DOM request. The action message is removed from internal storage and deleted from operator consoles.

Note: Note: You can differentiate the original message from the DOM copy by checking IFRAUDOM (if you are using an ATF) or by checking the automation value ACTIONDL. A command driven by the automation of the DOM can use SMSGID to correlate the instance of the message with the instance of the DOM.

D | DELMSG
Specifies that a DOM is expected for this message, and that when a

DOM is received for this message, the NetView program deletes the action message from internal storage and from operator consoles, but does not send a copy of the message through automation. DELMSG is the default value for action messages.

N | NODELMSG
Specifies that a DOM for this message is not expected. The NetView program does not keep any internal record of it for future deletion by a DOM request. If the message is an unsolicited system message, the NetView program does not process the DOM if one is sent. This setting is appropriate for situations when messages are issued by applications with Descriptor codes listed on the MVSPARM.ActionDescCodes CNMSTYLE statement, but the application does not issue a DOM. Alternatively, an operator or autotask can use the NetView DOM command to delete the message. NODELMSG is the default value for non-action messages.

Type: Message

Usage notes:

1. If the value of DOMACTION after automation is AUTOMATE or DELMSG, the NetView program allocates resources in expectation of a DOM. If many such messages are processed and the DOMs are not forthcoming, then both storage and processing time are negatively affected. To address this concern, you can use the DOM command to remove records associated with the message.
2. Action messages are held by default. When a message that is not an action message is automated, a DOMACTION(AUTOMATE) or DOMACTION(DELMSG) action, by default, sets the message as *held*. Consider adding a HOLD action in order to differentiate between HOLD(YES) and HOLD(LOCAL). A DOM can be automated only if the original message is still on the hold queue of some NetView task. Check the OVERRIDE setting for the task to which the primary copy of the message is to be sent in order to ensure that the setting for HOLD at that task allows the message to be held

EDIT ('*edit_specification***')**

The EDIT action enables you to make changes to an AIFR while it is in the automation table. The changes are made using the syntax and functions provided by the PIPE EDIT stage. With EDIT, messages and MSUs can be reformatted. The altered AIFR continues through automation. The original AIFR is no longer available.

While *edit_specification* must be enclosed in single quotation marks (in the form '*edit_specification*'), you cannot use single quotation marks (' ') within the *edit_specification* itself.

Edit specifications define the action to be taken on the AIFR data. For information about what you can include in the *edit_specification*, refer to *IBM Tivoli NetView for z/OS Programming: Pipes*. The edit specification does not support variables.

EXEC ([CMD(*cmdstring***)] [ROUTE(***routeparms***)])**
Indicates an action to be processed. You can specify CMD, ROUTE, or both with each EXEC action. You can code more than one EXEC action on a single statement to process more than one command or route extra copies of a message.

CMD Indicates a command, command list, or command processor you want the NetView program to run.

cmdstring
Is a command string of up to 2000 characters. The string can be literals, a variable, or a combination of literals and variables. Enclose literals in single quotes, but do not enclose variables in quotes.

Because variable values could contain leading or trailing blanks which could cause the command to fail, you can use the STRIP function to strip off leading and trailing blanks or hexadecimal zeros. The syntax for the STRIP function is

```
STRIP(varname)
```

where *varname* is the variable name whose value, without leading and trailing blanks and hexadecimal zeros, is to be inserted into the command.

This string contains the complete command syntax of the command or command list, including the command name and any parameters. If you have defined automation table variables, you can use them for parameters. See "Parse Templates as Compare Items" on page 205.

Japanese double-byte characters are not supported in NetView command strings.

Special rules apply to command strings that are longer than 255 characters. See Note 9 on page 218 for a description of these rules.

If you are using the automation table to convert alerts to Event Integration Facility (EIF) events or traps and forward the events or traps to the designated event server or an SNMP manager, add the `TECROUTE` keyword to the beginning of *cmdstring* as a prefix. Only one command prefixed by `TECROUTE` can be run for a specified alert; code all needed command actions in the same command.

See "Event/Automation Service" on page 404 for more information.

ROUTE
Instructs NetView to route *cmdstring* or the message to the operators whose identifiers are specified in *routeparms*.

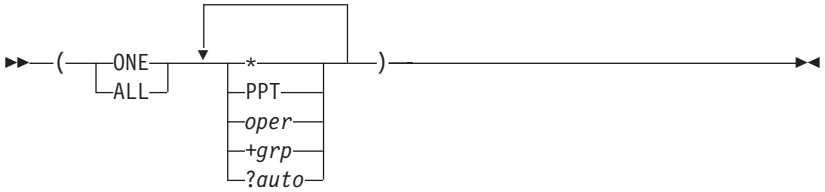
You can also route messages by using any of these methods:

- Use the MSGROUTE command in a command list that you issue from the automation table. Use the MSGROUTE command if, for example, you want to check part of a multiline message other than the first line before deciding where to route the message. For information about the MSGROUTE command, refer to *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language*.
- Use the EXCMD command. Information about the EXCMD command can be found in *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)*.
- Use the PIPE ROUTE stage command. Information about the PIPE ROUTE stage command can be found in *IBM Tivoli NetView for z/OS Programming: Pipes*.

routeparms

Specifies the operators or groups of operators to whom the message or *cmdstring* is routed for processing. The syntax for the parameter is:

Routeparms



Where:

ONE Routes the message or *cmdstring* to the first logged-on operator in the list or to the first operator who is assigned to a group appearing in the list and who is logged on.

ALL Routes the message or *cmdstring* to all the operators and groups of operators in the list who are logged on to the NetView system.

***** Indicates that the NetView program routes the message or *cmdstring* to the current operator task (the task that sent the message to the automation table). In cases where the current task cannot process the message or *cmdstring*, the NetView program routes to the primary autotask, defined by the FUNCTION.AUTOTASK.PRIMARY statement in the CNMSTYLE member.

For additional information about command routing, see Note 6 on page 217.

PPT Indicates that the NetView program routes the message or *cmdstring* to the PPT for processing.

oper The identifier of an operator to whom the NetView program routes the message or *cmdstring*. The operator identifier must be defined to the NetView program. The maximum length of an operator identifier is 8 characters. You can code as many operator identifiers as needed.

+grp The identifier of each group of operators to whom the NetView program routes the message or *cmdstring*. The maximum length of a group identifier is 8 characters, and it must begin with a plus sign (+). You can code as many group identifiers as required. Define group identifiers with the ASSIGN command.

Refer to the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* for information about the ASSIGN command.

?auto The *function_name* of an autotask defined with a *function.autotask.function_name* to which the NetView program routes the message or cmdstring. The autotask must be defined to the NetView program. You can code as

many as necessary in this manner. This value is resolved to the appropriate autotask name when the automation table is compiled.

Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about the function.autotask command.

Type: Both, except that EXEC with only the ROUTE option is type Message.

Usage notes:

1. The target task for a ROUTE action can only be those that are shown in the syntax diagram; you cannot route to an optional task.
2. The NetView program processes each EXEC action on each matching statement individually, except for ROUTE-only actions with the ALL option. To eliminate duplicate task IDs, the NetView program combines all ROUTE-only actions with the ALL option.
3. When you specify CMD on an EXEC action, enclose the command and any literal parameters in single quotes. Delimiters, such as spaces, that are required in a command string must also be enclosed in single quotes.

   To use variables as parameters, do not enclose the variable names in single quotes. You can define the variables in the *conditions* portion of the IF-THEN statement.

   Alternatively, if your statement is enclosed in a BEGIN-END section, you can define the variables in the IF-THEN statement that begins the section. If your definition gives a variable a value of null, you can pass the null value to the command list. However, passing a variable that you have not defined at all results in a syntax error.

   In Figure 47, the space within the single quotes after CLISTB separates the command name from the parameter value in VARPARM1. The space enclosed in single quotes between VARPARM1 and VARPARM2 delimits those two variable names. Also, a space within the single quotes before LITPARM sets the literal parameter off from VARPARM2.

```
EXEC (CMD('CLISTB ' VARPARM1 ' ' VARPARM2 ' LITPARM'))
```

Figure 47. Example of Specifying a CMD in an EXEC

   If VARPARM1 is OPER4 and VARPARM2 is OPER5, the resulting command is `CLISTB OPER4 OPER5 LITPARM`.
4. If you specify CMD on an EXEC action, also specify ROUTE.

   When you specify both CMD and ROUTE, the NetView program processes the CMD actions under the tasks specified in *routeparms*. Autotasks, in many cases, are ideal command destinations. You cannot route commands to SYSOP or the network log. Any BEEP, DISPLAY, HCYLOG, and HOLD actions you specify do not apply to the CMD action but to the incoming message.

   In Figure 48 on page 217, The NetView program sends the RUNNING command list to the first task in group +GRP01 that is logged on. If none of the +GRP01 operators are logged on, the command list goes to AUTOMGR instead. In any case, the NetView program does not display the message that triggers the entry.

```
IF MSGID='DSI530I' & TEXT(10) = TASKNAME ''' :' . THEN
      EXEC (CMD('RUNNING ' TASKNAME) ROUTE(ONE +GRP01 AUTOMGR))
      DISPLAY(N);
```

Figure 48. Example of Using the CMD and ROUTE Keywords

5. When an unsolicited message matches an ASSIGN PRI condition, the message is first routed to the autotask or operator station that is specified by the ASSIGN command, where automation continues. If the designated task is not active, routing does not occur. This affects these kinds of messages:
   - Unsolicited MVS messages, when automation is requested in MPF or MRT and the message is not directed to a specific console owned by some NetView task
   - Unsolicited VTAM messages received through the Program Operator Interface (POI)
   - Messages designated as *authorized receiver* by the message sender
   - Command output from a command running under the PPT
   - Messages originating at the PPT
   - Messages queued to the PPT

   Except for an unsolicited MVS message, if the message does not match an ASSIGN PRI condition, the message is routed to an authorized receiver, if one is logged on, and otherwise to the PPT. Automation then continues at the task where the message was routed. Unsolicited MVS messages not routed by an ASSIGN PRI condition are automated at the CNMCSSIR task.
6. When you specify CMD in an EXEC statement, always code a ROUTE action for that EXEC and, unless you are certain that the automation occurs at an autotask or an operator's OST, do not code an asterisk (*) for the ROUTE action. In the event that a ROUTE action of asterisk is encountered or no ROUTE action is coded, the following rules apply to the routing of the command:
   - If the automation occurs at an autotask or an operator's OST, the command is routed to that task.
   - If the following message types do not match any ASSIGN PRI specification, then they are automated at an optional task:
     - MSUs from the hardware monitor that are automated at the BNJDSERV task.
     - Unsolicited messages from the MVS program, automated at the CNMCSSIR task.
     - Unsolicited messages resulting from an XCF transmission, automated at the DSIXCFMT task.
     - Other optional tasks, if the task issues a message using DSIPSS TYPE=OUTPUT.

     For all these tasks, commands without an explicit ROUTE specification are queued to the primary autotask, defined by the FUNCTION.AUTOTASK.PRIMARY statement in the CNMSTYLE member.
   - At the PPT, commands without an explicit ROUTE are queued to the primary autotask, defined by the FUNCTION.AUTOTASK.PRIMARY statement in the CNMSTYLE member.

7. If the automation table cannot find an active task to run a command, the NetView program sends a DWO032E message through the automation table and to the network log to indicate the problem.

   You might want to have a statement in the automation table that automates message DWO032E, because it indicates that an automation statement is failing to function as expected. DWO032E is always sent to the network log; the automation-table NETLOG action does not affect it. This message can occur if none of the tasks to which you route a command are active.

   Message DWO032E is not displayed to an operator by default, but it can be routed to an operator from an automation-table statement to indicate to an operator that there is a problem with automation. If you do route the DWO032E message from an automation-table statement, ensure that at least one operator, such as the PPT, in the list of operators specified is logged on (to avoid the possibility of a looping condition).

   To avoid the problem of commands that cannot be run, consider including a stable autotask somewhere in your ROUTE list when issuing a command from the automation table.

8. Using EXEC with both CMD and ROUTE only routes the command you specify with the CMD keyword.

   It does not affect the routing of the message that is undergoing automation-table processing. To change the routing of the message itself, use an EXEC action with ROUTE, but not CMD.

   When you specify ROUTE but not CMD on an EXEC action, the NetView program routes the message to the operators specified in *routeparms* using the BEEP, DISPLAY, HCYLOG, and HOLD actions specified in the IF-THEN statement. Otherwise, the NetView program processes the BEEP, DISPLAY, HCYLOG, and HOLD actions under the current operator task (the task that sent the message to the automation table).

   For example, the statement in Figure 49 displays a message to operator OPER1.

```
IF MSGID='DSI530I' & TEXT(10)= . ':' . THEN
      EXEC (ROUTE(ONE OPER1))
      DISPLAY(Y);
```

Figure 49. Example of Using EXEC Action with the ROUTE Keyword

9. Commands issued from the automation table can be up to 2000 characters. However, command parse tokens in the NetView program have a maximum length of 255 characters.

   A command parse token is made up of all characters between parse delimiters in a command. Parse delimiters are commas, equal signs, parentheses, and blanks. In Figure 50 on page 219, the parse tokens are:
   - MYCMD
   - KEYWORD1
   - VALUE1
   - KEYWORD2
   - VALUE1
   - VALUE2

Each parse token can be up to 255 characters.

```
MYCMD KEYWORD1=VALUE1,KEYWORD2=(VALUE1,VALUE2)
```

Figure 50. Example of Using A Parse Token

In an automation-table statement, use two single quotes to represent one single quote within a literal string. If a parse token has a pair of single quotes in it, any parse delimiters between the single quotes are ignored. For example, if the command shown in Figure 51 is issued from the automation table, `'LITERAL, STRING'` is one parse token. The comma is not used as a parse delimiter.

```
IF MSGID='DSIxxxI'
  THEN EXEC(CMD('MYCMD KEYWORD1='LITERAL, STRING',
                KEYWORD2=(VALUE1,VALUE2)')
            ROUTE(ALL OPER1));
```

Figure 51. Example of Ignoring Parse Delimiters

If there is an unbalanced set of single quotes, everything from the extra single quote to the end of the command is considered one parse token. For example, if the command shown in Figure 52 is issued from the automation table, `'LITERAL, STRING, KEYWORD2=(VALUE1,VALUE2)'` is one parse token.

```
IF MSGID='DSIxxxI'
  THEN EXEC(CMD('MYCMD KEYWORD1='LITERAL, STRING,
                KEYWORD2=(VALUE1,VALUE2)')
            ROUTE(ALL OPER1));
```

Figure 52. Example of Unbalanced Parse Tokens

10. Using the REFRESH command, you can dynamically delete operators, and dynamically add operators without predefining the operators to the NetView program.

    The automation table is activated successfully even if operators targeted by the ROUTE keyword in automation statements are not presently defined to the NetView program.

    If you issue the AUTOTBL command to activate or test an automation table, and the ROUTE keyword on an EXEC action specifies an operator that is not defined to the NetView program, you receive a message informing you that the operator ID specified on the ROUTE keyword is unknown. The automation table is then activated successfully.

    Regardless of whether an operator is defined to the NetView program, messages routed to operators that are not logged on are delivered to the next assigned operator, or to the original destination.

    If an operator definition is deleted using the REFRESH command, the operator session continues until the operator logs off. Messages routed to operators that are logged on but no longer defined to the NetView program are still delivered to that operator.

HCYLOG(Y|N)

Determines whether the message is placed in the hardcopy log, if the hardcopy log task is active. If you do not specify an HCYLOG value in the automation table or elsewhere, the default is HCYLOG(Y).

Type: Message

HIGHINT(Y | N)
Specifies a high-intensity 3270 setting for terminals that support high intensity. The hardware monitor uses the setting if it displays the MSU. The command facility uses the setting if it displays the message.

See Note 8 on page 227 for information about hardware monitor defaults for MSUs.

Type: Both

HOLD(YES | NO | LOCAL |)
Determines whether the NetView program holds the message on the operator's screen after display. The HOLD parameter also determines whether queued action messages are rerouted to the authorized receiver when the operator logs off.

HOLD(YES)
The message is held on the NetView screen and is kept on a queue for both operators and autotasks.

If it is an action message, it is rerouted upon logoff.

HOLD(LOCAL)
The message is held as with HOLD(YES). Queued action messages are not be rerouted upon logoff.

HOLD(NO)
Prevents a message from being held on the screen.

Action messages marked HOLD(NO) are not queued for later processing unless the message is sent to an autotask. If the message is an Action message that is sent to an autotask, it is queued and rerouted to the authorized receiver after the operator logs off.

Note: Action messages, such as WTORs, that are marked HOLD(NO) are not processed by a subsequent DOM, such as a reply. Therefore, the highlighting does not change, but the messages do scroll off the screen.

Note: Any of the HOLD values can be abbreviated to one letter. For example, HOLD(LOCAL) and HOLD(L) are synonymous.

Type: Message

NETLOG(Y | N [*indicator-number***] [*] [***oper***[,...]] [+***grp***[,...]])**
Determines whether the NetView program places the message in its network log and whether the message activates a status monitor important message indicator for specified operators or groups of operators. If you do not specify a NETLOG value in the automation table or elsewhere, the default is NETLOG(Y).

indicator-number
Identifies the status monitor important message indicator.

*
Indicates that the NetView program routes the message or *cmdstring* to the current task (the task that sent the message to the automation table). If the current task is CNMCSSIR, message routing can differ.

oper [,...]
Specifies the operator identifier of the operators for whom the

message is logged as important. The operator identifier must be defined to the NetView program. The maximum length of an operator identifier is 8 characters. You can code as many operator identifiers as needed.

+grp [,...]
Specifies the group identifier of the groups of operators for whom the message is logged as important. The maximum length of a group identifier is 8 characters, and it must begin with a plus (+) sign. The ASSIGN command is used to define the group identifiers. Refer to the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* for more information about the ASSIGN command.

Type: Message

Usage notes:

1. You can also use the F (FILTER) statement to define important message indicators. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about the F (FILTER) statement.
2. For NETLOG, if only an indicator number is specified, the message is logged as important for the authorized receiver. If an indicator number and a list of operators or groups of operators are specified, the message is logged as important for the operators and groups of operators.

   This is an IF-THEN statement that is coded with only an indicator-number for NETLOG

   ```
   IF MSGID='IST105I' THEN
         NETLOG(Y 2);
   ```

   .

   Message IST105I is defined as an important message with a status monitor important message indicator number of 2. Because no operators or groups of operators were specified, when the NetView program encounters message IST105I, the message is logged as important for the authorized receiver.

   This shows how the IST105I message is logged when an indicator number and a list of operators and groups of operators are specified for NETLOG. The IST105I message is defined as an important message with a status monitor important message indicator number of 2.

   ```
   IF MSGID='IST105I' THEN
         NETLOG(Y 2 * OPER1 +GRP5 OPER6);
   ```

   The NetView program logs the IST105I message as important with defined highlighting for OPER1, OPER6, all operators assigned to group +GRP5, and the current operator. Duplicate highlighting and logging do not occur if specified operators are also assigned to a specified group. If operators OPER1 and OPER6 are assigned to group +GRP5, each operator receives only one copy of message CNM039I, which is displayed if an operator is not in the status monitor or log browse.
3. You can assign a status monitor important-message indicator in the automation table to an operator that is not presently defined to NetView. The automation table activates successfully when you use the AUTOTBL command. When you dynamically add the operator, the operator can see any indicators for messages that are processed by NetView automation after the operator logs on, but is not able to see indicators for messages processed before the operator logs on. The message indicator is set when the operator logs on.

If you dynamically delete an operator definition, but the operator remains logged on, you can still assign a status monitor important-message indicator to that operator.

SRF (*filterlevel [setting] [filterscope]***)**

Indicates that recording filters are to be set for the MSU to control the recording of data in the hardware monitor database.

If you do not specify a filter setting for an MSU in the automation table or elsewhere, the default for ESREC is PASS. The default for AREC depends on the MSU. See the SRFILTER command in the *IBM Tivoli NetView for z/OS Command Reference Volume 1 (A-N)* for more information. For MSUs that pass AREC, the default for ROUTE is PASS, the default for TECROUTE is BLOCK, the default for TRAPROUT is BLOCK, the default for OPER is BLOCK.

filterlevel

Indicates the recording filter you want to set for the MSU.

ESREC

Determines whether the hardware monitor records an MSU as an event.

AREC Determines whether the hardware monitor records an event as an alert.

OPER Determines whether the hardware monitor generates BNJ146I and BNJ030I messages from an alert and sends them to the authorized receiver.

ROUTE

Determines whether the hardware monitor routes an alert to the system acting as the NetView focal point for alerts.

TECROUTE

Determines whether the hardware monitor converts an alert to an EIF event and routes the event to the designated event server.

TRAPROUT

Determines whether the hardware monitor converts an alert to a trap and routes the trap to the SNMP manager.

setting Specifies whether an MSU matching the conditions is to be blocked from (or passed through to) one of the following events:

- The hardware monitor database
- A NetView operator (as a message to the authorized receiver)
- The hardware monitor focal point

BLOCK

The data is blocked. This is the default.

PASS The data is passed.

filterscope

Specifies whether filter settings apply to the primary or secondary event, if the hardware monitor records a secondary event.

PRI Indicates that the filter settings apply to the primary event.

SEC Indicates that the filter settings apply to the secondary event, if one exists.

BOTH Indicates that the filter settings apply both to the primary event and to any secondary event. This is the default.

Type: MSU

Usage notes:

1. Hardware monitor filters set to BLOCK with the hardware monitor SRFILTER (SRF) command do not prevent MSUs from coming to the automation table. SRF actions in the automation table can override the filter settings that the SRFILTER command establishes.

   The SRF action cannot set color and highlighting options as the SRFILTER command can. Instead, use the BEEP, COLOR, HIGHINT, and XHILITE actions to set color and highlighting options from the automation table.

   See "Filtering Alerts" on page 299 for more information about SRFILTER and SRF.

2. The default *filterscope* of BOTH is sufficient for most MSUs. Secondary event recording is a rare case in which the hardware monitor determines that the affected resource differs from the resource causing the failure; therefore, the NetView program creates two events from a single problem record. The two events are similar, but they specify different resource names, and the primary event has a shorter resource hierarchy than the secondary event.

   The hardware monitor SRFILTER command can affect each of the events separately. The NetView program uses only the primary event to search for a match in the automation table, but the filtering options you specify in the table can apply to either event or to both.

   By default, the filtering options apply to both events. You must specify a *filterscope* if you want the automation table to filter the primary and secondary events separately.

3. ESREC or AREC filter settings (BLOCK or PASS) are valid for alerts forwarded from a NetView or non-NetView remote node entry point over the SNA-MDS/LU 6.2 alert forwarding protocol.

   For example, if the SRF action is used to set the ESREC filter level to BLOCK and the AREC filter level to PASS for non-LU 6.2 forwarded alerts, hardware monitor considers ESREC/BLOCK and AREC/PASS an improper setting and resets AREC to BLOCK. Therefore, ESREC and AREC are both set to BLOCK and no data is recorded to the database.

   However, for LU 6.2 forwarded alerts, if the SRF action is used to set both ESREC to BLOCK and AREC to PASS, hardware monitor accepts this setting, and only an alert record is recorded to the database. This is *alert-only* recording, which is illustrated in this example:

```
*====================================================================*
* Was the MSU forwarded over LU 6.2, and if so                       *
* then record it in the hardware monitor database as alert-only by   *
* BLOCKing ESREC and PASSing AREC.                                   *
*====================================================================*
IF HMFWDSNA = '1' THEN
      SRF(ESREC BLOCK)
      SRF(AREC  PASS);
```

   As explained in Chapter 26, "Centralized Operations," on page 373, default alerts received over LU 6.2 from NetView entry points are recorded to the database as alert-only, but alerts received over LU 6.2 from non-NetView entry points go through the normal ESREC and AREC filters.

Data is recorded to the database in accordance with how these filters are passed. The SRF action enables you to override these defaults. For example, you can set the AREC filters level to PASS and the ESREC filter level to BLOCK to record non-NetView alerts as alert-only, or you can set the ESREC filter level to PASS to record events or statistical data for alerts forwarded from entry point NetView systems.

SYSLOG (Y|N)
Determines whether the NetView program sends the message to the MVS system log. If you do not specify a SYSLOG value in the automation table or elsewhere, the default is SYSLOG(N). SYSLOG has no effect on messages received from the subsystem interface. Messages received from the subsystem interface are unconditionally placed in the MVS system log before being sent to the NetView program.

Type: Message

TRACE ('*tracetag*')
Sets the tracing indicator and tag for the message or MSU so that automation-table processing can be traced. The *tracetag* tag must be enclosed in quotes, as in '*tracetag*', must be no more than 16 characters long, and must not include any blanks. A CONTINUE(Y) action is implied with the TRACE action. See "Using NetView Automation Table Tracing" on page 484 for more information about tracing AIFRs through the automation table.

Type: Both

XHILITE (BLI|REV|UND|NONE)
Specifies foreground extended highlighting. See Note 8 on page 227 for information about defaults for MSUs.
BLI Specifies blinking
REV Specifies reverse-video highlighting
UND Specifies underscoring
NONE
Specifies no extended highlighting

Type: Both

XLO (Y|N)
Specifies external logging only. When you set XLO to N, the recording filters set by the hardware monitor and the automation table take effect. When you set XLO to Y, only external logging occurs, and the NetView program ignores the recording-filter settings. If you do not specify an XLO value in the automation table, the XITCI installation exit for BNJDSERV, or installation exit DSIEX16B, the default value is **N** (to allow the recording filters to take effect).

Type: MSU

Usage notes:

1. You cannot combine actions with BEGIN on a single automation statement. The rules for specifying an action more than once for a single message or MSU depend upon the action.

   You can use the EXEC action as many times as you want for a single message or MSU. Each of the EXEC actions is performed. For example, the statement in Figure 53 on page 225 routes incoming DSI530I messages to the tasks in group +GRP01 and also runs the RUNNING command list under autotask AUTOMGR.

```
IF MSGID='DSI530I' & TEXT(10)= TASKNAME ''' :' . THEN
      EXEC (ROUTE(ALL +GRP01))
      EXEC (CMD('RUNNING ' TASKNAME) ROUTE(ONE AUTOMGR))
      DISPLAY(Y);
```

Figure 53. Example of Performing Multiple EXECs for a Message or MSU

The NetView program processes each EXEC action of each matching statement individually. The exception is EXEC actions that use the ROUTE keyword with the ALL option (but without the CMD keyword). If you specify more than one EXEC(ROUTE(ALL *parm1 parm2 parmx*)) action for a single message, the NetView program merges the task lists and does not route the message to any task more than once.

You can also use the SRF action more than once for a single MSU. If you give conflicting settings for a filter, whether in a single statement or in separate statements that are processed because of a CONTINUE action, the NetView program uses the last setting given.

If you specify an action (other than EXEC or SRF) more than once in a single automation statement, the first occurrence of the action takes precedence. For example, in the statement in Figure 54, the first occurrence of the HOLD action is Y and the first occurrence of the COLOR action is RED. Therefore, operators OPER1, OPER2, OPER3, and OPER4 receive message IEE136I held in red.

```
IF MSGID = 'IEE136I' THEN
      EXEC (ROUTE(ALL OPER1 OPER2)) HOLD(Y)
      EXEC (ROUTE(ALL OPER3 OPER4)) HOLD(N) COLOR(RED);
```

Figure 54. Example of Specifying an Action More than Once

You can use the CONTINUE(Y) action any number of times on separate statements to continue automation-table processing after matches are found. A message or MSU continues processing until it matches a statement that does not have a CONTINUE(Y).

For actions other than EXEC, SRF, and CONTINUE, if use of the CONTINUE action results in the application of conflicting actions to a single message or MSU, the NetView program uses the value given last. For example, a BEEP(Y) action can override a BEEP(N) action given earlier in the automation table. However, if a NETLOG(Y) action without an important-message indicator follows a NETLOG(Y) action with an important-message indicator, with no intervening NETLOG(N), the indicator from the first NETLOG(Y) is retained as shown in Figure 55:

```
*** First statement for IEE136I ***
IF MSGID = 'IEE136I' THEN
      EXEC (ROUTE(ALL OPER1 OPER2)) HOLD(Y) NETLOG(Y)
      EXEC (ROUTE(ALL OPER3 OPER4)) COLOR(RED)
      CONTINUE(Y)


*** Second statement for IEE136I ***
IF MSGID = 'IEE136I' THEN
      EXEC (ROUTE(ALL OPER5 OPER6))
      EXEC (ROUTE(ALL OPER7 OPER8)) COLOR(BLU) NETLOG(N);
```

Figure 55. Example of Conflicting Action for a Message Using CONTINUE

If message IEE136I is issued, a match occurs on the first statement. Because CONTINUE(Y) is coded, the automation table is searched for additional matches. When the second statement is found, the actions set in the first

statement can optionally be altered because of the CONTINUE(Y) that was coded. The HOLD action is established in the first statement and unchanged in the second. The COLOR action is set in the first statement, but then altered in the second. Finally, the NETLOG action is ultimately determined by the second statement. The result is that OPER1 through OPER8 each displays message IEE136I that is held and has a color of blue. The message is not sent to the NETLOG.

2. For actions coded yes or no (Y|N), you can code YES or Y for yes, and NO or N for no.
3. For a message action, the default value indicated is the NetView system default. Use this list to determine override defaults. Each item in the list can override the items preceding it in the list.
   a. System default.
   b. DEFAULTS command.
   c. Installation exit DSIEX02A, if used to replace DEFAULTS settings.
   d. Action on a matching statement in the automation table.
   e. Installation exit DSIEX16, if used to replace DEFAULTS settings.
   f. OVERRIDE command. SCRNFMT specifications for message color and highlighting do not override automation-table specifications.
   g. Installation exit DSIEX02A or DSIEX16, if used to replace OVERRIDE settings.
4. A YES or NO setting for CNM493I, DISPLAY, HCYLOG, NETLOG, or SYSLOG on the OVERRIDE command overrides the setting specified for the action in the automation table, if any. A DISABLE setting for BEEP or HOLD on the OVERRIDE or DEFAULTS command means that the NetView program does not use the setting specified in the automation table. The BEEP keyword on the DEFAULTS and OVERRIDE commands affects only message processing.

   Refer to the NetView online help for more information about the DEFAULTS and OVERRIDE commands.
5. For MSUs, the filtering and highlighting actions apply only to alert major vectors coming through the hardware monitor. Filtering and highlighting actions include SRF, XLO, COLOR, HIGHINT, and XHILITE.
6. Use this list to determine override filtering and highlighting options. Each item in the list can override the items preceding it in the list.
   a. Except for XLO, hardware monitor filter settings specified with the SRFILTER (SRF) command
   b. For XLO, the return code from BNJDSERV's installation exit XITCI
   c. Action on a matching statement in the automation table
   d. Installation exit DSIEX16B

   If the final XLO value after DSIEX16B is YES, the MSU goes to external logging only. Otherwise, if the final ESREC value is BLOCK, the NetView program ignores the AREC, ROUTE, and OPER filters. The hardware monitor does not record the MSU as an event or an alert. If the ESREC value is PASS but the AREC value is BLOCK, the NetView program ignores the ROUTE and OPER filters.

   Setting filters to BLOCK in the hardware monitor or setting XLO to YES in XITCI does not prevent an MSU from going to the automation table. The automation table still processes the MSU and has an opportunity to override the previous XLO setting and other filter settings for the MSU.

7. For MSUs, use this list to determine CNM493I message generation override options. Each item in the list can override the items preceding it in the list.
   - System default (which is to generate the CNM493I messages).
   - DEFAULTS command.
   - Action on a matching statement in the automation table.
   - OVERRIDE command for those MSUs sent to automation by an OST, such as with DSIAUTO. You cannot use the OVERRIDE command on DSTs such as BNJDSERV, which is the NetView task that delivers MSUs to the automation table for processing from the hardware monitor.
8. If an alert major vector passes through all of the steps listed in note 5 on page 226 without obtaining any color or highlighting option, the hardware monitor uses color maps and your SRFILTER COLOR DEFAULTS settings to control display of the alert.

   With the initial settings, the alert displays in white or high intensity when first appearing on the Alerts-Dynamic panel, but otherwise appears in turquoise or low intensity. BEEP is set to YES and XHILITE to NONE. (Refer to the *IBM Tivoli NetView for z/OS Customization Guide* for information about color maps and *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)* for information about the SRFILTER COLOR DEFAULTS settings.)

   If, however, the alert major vector obtains a value for any one of the color and highlighting options, SRFILTER COLOR DEFAULTS does not apply. Instead, the alert receives default settings for any unspecified options:

   BEEP YES
   COLOR TUR (turquoise)
   HIGHINT NO
   XHILITE NONE

   If you do not specify a value in the automation table or elsewhere (see Note 5 on page 226), the default filter settings are:
   - The default for XLO is NO.
   - The default SRFILTER settings for the hardware monitor control the other filter settings.
   - The default for ESREC is PASS.
   - The default for AREC depends on the alert type (refer to *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)*).
   - The default for ROUTE is PASS if an alert passes AREC.
   - The default for OPER is BLOCK.
9. The NetView program processes HOLD and BEEP actions for a message only if you code DISPLAY(Y) in the automation table. HOLD and BEEP do not work for messages routed to an MVS console. HOLD, DISPLAY, and BEEP are the only automation actions that are preserved when messages are forwarded across domains over OST-NNT sessions.
10. Do not automate messages that were assigned to SYSOP in another NetView program on your system. Doing so might cause both NetView programs to loop.

    See Chapter 32, “Running Multiple NetView Programs Per System,” on page 455 for valid methods of communicating between two NetView programs in a system.

ALWAYS Statement

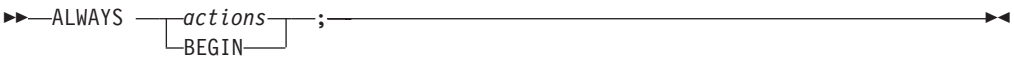
The ALWAYS statement enables you to specify actions or a series of statements that the NetView program processes for all messages and MSUs that reach that point in the table.

You can use the ALWAYS statement with CONTINUE at the beginning of an automation table or a BEGIN-END section to set defaults for the table or section.

You can also use the ALWAYS statement at the end of an automation table or a BEGIN-END section to handle messages and MSUs that do not match any other statement in the table or section.

The syntax for the ALWAYS statement is:

ALWAYS Statement



Where:

ALWAYS
The keyword coded at the beginning of each ALWAYS statement.

actions Specifies actions for the NetView program to take. For information about actions, see "Actions" on page 209.

BEGIN
Specifies the beginning of a BEGIN-END section. For more information, see "BEGIN-END Section" on page 151.

Usage notes:

1. Like other statements, an ALWAYS statement can be message-type, MSU-type, or both-type.

   Any message-type action makes the ALWAYS statement a message-type statement and prevents the statement from affecting MSUs. Any MSU-type action makes the ALWAYS statement an MSU-type statement and prevents the statement from affecting messages.

   See "Types of Automation-Table Statements" on page 148 for more information about statement types.
2. The statements indicated by ALWAYS statement are processed only when the ALWAYS statement is reached through logical and sequential automation-table processing. If automation-table processing stops because of a match, any ALWAYS statements after the match are not processed.
3. Exercise caution when using an ALWAYS statement that issues a command or command list.

   Usually, such statements occur only in BEGIN-END sections. Inappropriate use of such a statement can affect a large number of messages and MSUs.

%INCLUDE Statement

The %INCLUDE statement enables you to keep portions of your automation table in separate files or members.

The syntax for the %INCLUDE statement is:

%INCLUDE Statement

Where:

%INCLUDE
The keyword coded at the beginning of each %INCLUDE statement.

membername
The name of the member to be included. The member must be in the DSIPARM data set.

&varname
The name of an existing local or global variable, preceded by the ampersand (&) character.

Usage notes:

1. Each %INCLUDE statement can be no longer than one line.
2. Unlike other automation-table statements, the %INCLUDE statement does not end with a semicolon (;).
3. A member that has been included can contain %INCLUDE statements as well as other automation-table statements.
4. A member that has been included cannot include itself either directly or indirectly.
5. If you specify a variable name, the NetView program includes the designated member or file when you issue the AUTOTBL command. The NetView program searches for the variables in this order:
   - If the AUTOTBL command is issued from a command procedure, the NetView program searches first for a local variable of the name *varname*.
   - If the AUTOTBL command is not issued from a command procedure or there is no local variable of the name *varname*, the NetView program searches next for a task global variable, and finally for a common global variable.

   If you change the value of the variable after activating the automation table, the member that is included does not change, unless you reissue the AUTOTBL command.

   For example, you might use *&varname* to include table segments that are tuned to the message and MSU traffic you expect during certain shifts. Based on the time of day, a command procedure might update the variable before loading the automation table.

The %INCLUDE statement is not exclusive to the automation table; you can use it in other DSIPARM members also. For a full description of the %INCLUDE statement, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

SYN Statement

The SYN statement enables you to define synonyms for use later in the automation table. A synonym has a name and a value. After defining a synonym, you can use the name of the synonym elsewhere in the table. When you activate the table, the NetView program substitutes the synonym value for the name.

Synonyms enable you to provide a shorthand notation for long, repetitive strings. Synonyms can also help you modify and maintain an automation table, because you can change a value throughout a table by changing it in one place.

The syntax for the SYN statement is:

SYN Statement

Where:

SYN The keyword coded at the beginning of each SYN statement.

synname
The name of the synonym, up to 256 characters.

synvalue
The value of the synonym.

Usage notes:

1. The definition of a synonym must precede the use of the synonym in the automation table. You can define a synonym's value only once in the table, but thereafter you can use the synonym as often as you like. Consider defining all synonyms at the beginning of the table.
2. You cannot nest a synonym inside another synonym.
3. You can use blanks, alphanumeric characters, and other characters in synonym names and synonym values except as follows:
   - Synonym names cannot contain a percent sign (%) or a semicolon (;).
   - Synonym values cannot contain a semicolon (;).
   - Because single quotation marks are used as the delimiter for the synonym value, if a synonym value is to contain a single quotation mark ('), you must represent it as two consecutive single quotation marks (''). Do not substitute a double quote for two single quotes. For example, the synonym in Figure 56 contains single quotation marks.

```
  SYN %LogFullCondition% = 'MSGID= ''IFB040I''';
:
IF %LogFullCondition% THEN
      EXEC(CMD('MVS S CLRLOG') ROUTE(ONE AUTO1));
```

Figure 56. Example of Using a SYN Statement

4. Substitution is not performed on synonyms found within quotes. Synonyms found within quotes are treated as literal strings. For example, consider the SYN statement and automation table entry in Figure 57.

```
SYN %LDOMAIN% = '''CNM01''';
:
IF MSGID = 'DSI530I' & DOMAINID ='%LDOMAIN%' THEN
      EXEC (CMD('CLIST1 ')ROUTE(ONE AUTO1));
```

Figure 57. Example of Incorrect Synonym Substitution

Although the statement in Figure 57 uses correct syntax, no substitution occurs for the synonym %LDOMAIN% because it is coded within single quotes. If you want single quotes to be included as part of the synonym, code the SYN

statement and automation table as shown in Figure 58.

```
SYN %LDOMAIN% = '''CNM01''';
.
.
.
IF MSGID = 'DSI530I' & DOMAINID = %LDOMAIN% THEN
      EXEC (CMD('CLIST1 ')ROUTE(ONE AUTO1));
```

Figure 58. Example of Correctly Using Synonym Substitution

5. Consider using a naming convention for synonyms.

Design Guidelines for Automation Tables

When you are designing or coding an automation table, consider the techniques listed in this section.

Limit System Message Processing

Limit the number of system messages processed by the NetView program. Use operating system facilities such as MPF to avoid sending messages to the NetView program when you do not want NetView to automate or display the messages. This practice enhances performance by reducing the number of times the NetView program must search the automation table. See "Suppressing System Messages" on page 299 for more information.

Streamline the Automation Table

Make the automation table readable and consistent, and therefore easier to maintain, by performing these activities:

- Use comments at the beginning of each automation table member or file to describe the statements in that member.
- Use comments to describe what messages or MSUs an automation statement should match and what the statement actions are to accomplish.
- Use indentation.

  For example, indent the actions for IF-THEN statements, BEGIN keywords and their corresponding END statements, and statements in BEGIN-END sections.

  Although comments must start in column 1, statements do not have to start there. You can use as many blanks as you want within a statement; you must, however, end the statement with a semicolon.
- Use blank lines and comments within an automation member to separate statements and groups of statements.
- Use a naming convention for automation-table members.
- Separate automation-table logic into multiple members. You can then enable or disable this logic as needed.
- Define automation-table statements or groups of statements using LABEL-ENDLABEL or GROUP keywords to allow enabling or disabling automation-table logic as needed.

Group Statements with BEGIN-END Sections

Use BEGIN-END sections to put easily identifiable types of messages and MSUs into their own sections of the table. By doing so, you enhance the performance of the automation table. You can also make your automation table easier to read, understand, and maintain.

You can use an ALWAYS statement at the end of each section to specify the handling of messages and MSUs that do not have a specific statement within the section. Figure 59 uses ALWAYS statements in this way to prevent further processing for messages and MSUs that do not have specific statements.

```
* Statements for major-vector X'0000' MSUs
IF MSUSEG(0000) ¬= '' THEN
   BEGIN;
   :
      ALWAYS;
   END;
* Statements for VTAM messages
IF MSGID = 'IST' . THEN
   BEGIN;
   :
      ALWAYS;
   END;
* Statements for JES2 messages
IF MSGID = '$HASP' . THEN
   BEGIN;
   :
      ALWAYS;
   END;
* Statements for command-facility DSI messages
IF MSGID = 'DSI' . THEN
   BEGIN;
   :
      ALWAYS;
   END;
* Statements for hardware monitor messages
IF MSGID = 'BNJ' . THEN
   BEGIN;
   :
      ALWAYS;
   END;
```

Figure 59. Example of Grouping Statements

Dividing the table into sections minimizes the number of statements that must be processed to find a match for a message or MSU. In Figure 59, for a NetView command-facility DSI message, the NetView program needs to check only two statements before reaching the section for command-facility DSI messages:

- The `MSGID = 'IST'` statement
- The `MSGID = '$HASP'` statement

The NetView program does not check the `MSUSEG(0000) ¬= '` statement for a message, because MSUSEG is for MSUs only. Within each section, additional BEGIN-END sections can also help reduce the number of automation-table comparisons that must be made.

Arrange BEGIN-END sections so that the most frequently used sections come earlier in the table. You can use the AUTOCNT command to generate automation-table usage reports, which you can use to analyze message and MSU frequency.

The sample NetView automation table DSITBL01 (CNMS1015) uses BEGIN-END sections for groups of messages.

You can use the occurrence-detection condition items (THRESHOLD and INTERVAL) within a BEGIN-END section to indicate different actions taken depending on whether the occurrence-detection threshold has been reached, as

shown in Figure 60.

```
IF MSGID = 'IST102I' THEN
   BEGIN;
     IF THRESHOLD(3 7 00:00:00) = '1' THEN
        EXEC(CMD('MSG SYSOP VTAM DOWN - ASSISTANCE REQUIRED')
             ROUTE(ONE AUTOVTAM AUTO1 PPT *));
     ALWAYS
        EXEC(CMD('VTAMSTRT')
             ROUTE(ONE AUTOVTAM AUTO1 PPT *))
        EXEC(CMD('MSG SYSOP AUTOMATION RESTARTING VTAM')
             ROUTE(ONE AUTO1 PPT *));
   END;
```

Figure 60. Example of Occurrence-Detection Condition Items

Using the example in Figure 60, the NetView program does one of these functions if VTAM becomes inactive:

- If this has happened at least two other times within the past seven days, the NetView program sends a message to the system operator informing the operator of the problem.
- If this is the first or second time this has happened within the past seven days, the NetView program performs these activities:
  - Tries to restart the VTAM program.
  - Notifies the system operator that the automation table had sent a request for an autotask to restart the VTAM program.

Isolate Complex Compare Items

Some Compare items take longer to evaluate than others. Compare items with the potential to be relatively slow include:

- MSUSEG compare items that specify complex locations
- The DSICGLOB ATF program that is supplied with the NetView program
- Lengthy ATF programs that you write for yourself

You can isolate these items by placing them in BEGIN-END sections started with an IF-THEN statement, so that the NetView program evaluates the items only when the comparison in the IF-THEN evaluates as true.

You can also isolate items by placing them after a logical-AND operator (&). In this case, the NetView program evaluates the items only if the conditions before the AND operator are met. For example, the statement in Figure 61 isolates the DSICGLOB ATF so that the NetView program retrieves the value of common global variable REQUIREDSTATUSB only when a message with an ID of XYZ123 comes in.

```
IF MSGID = 'XYZ123' &
   ATF('DSICGLOB REQUIREDSTATUSB') = 'ACTIVE' THEN
      EXEC (CMD('RESTARTB'));
```

Figure 61. Example of Isolating a Complex Compare Item

Include Other Automation Tables

To make automation tables easier to update, create an automation table with several automation members by using the %INCLUDE statement. For example, you can define a separate member for each class of messages and MSUs. You can also put automation-table statements that are common to several automation tables into a single member and include that member in each of the automation tables.

Figure 62 shows the beginning of an automation table that includes other automation members.

```
* Main automation table
* Synonym definitions
%INCLUDE ATSYNS
* Statements for major-vector X'0000' MSUs
%INCLUDE AT0000
* Statements for VTAM messages
%INCLUDE ATVTAM
* Statements for JES2 messages
%INCLUDE ATJES2
* Statements for NetView command-facility DSI messages
%INCLUDE ATNVDSI
* Statements for NetView hardware monitor messages
%INCLUDE ATNVBNJ
```

Figure 62. Example of Including Other Automation Tables

In Figure 62, automation-table members ATSYNS, AT0000, ATVTAM, ATJES2, ATNVDSI, and ATNVBNJ contain the statements for synonym definitions, X'0000' major vectors, VTAM messages, JES2 messages, NetView DSI messages, and NetView BNJ messages, respectively.

Note: BEGIN-END sections can be used in the included members or files to increase the efficiency of the automation table.

Tailor Automation Tables for Your Operation

Write different included sections for different phases of your operation. For example, you can write a table for each shift. You can tune each table to the message and MSU traffic you expect for that shift.

Another approach is to use a series of automation tables concurrently. Each table can be loaded or dropped using the AUTOTBL or AUTOMAN command. When loading an automation table, you can specify the order in which to process the tables in relation to other active automation tables. This is useful for changing your automation policy at specific times such as off-shift and prime shift.

Use Synonyms

Use synonyms to define complex or repetitive strings within an automation table or to standardize using automation tables across several systems. You can define all system-dependent specifications as synonym values and place the synonyms at the beginning of the automation table or in an included member. Then copying an automation table to another system might require changing only the synonym values or the member containing the synonyms.

Note: Synonyms must be defined in the same table in which they are used. They cannot span multiple tables.

Place Statements Carefully

Be careful about how you order the statements in an automation table or set of automation tables. Incorrect placement or specification of a statement can result in:

- A message or MSU matching an unintended statement
- A message or MSU not matching an intended statement because of a misplaced BEGIN-END section

- A message or MSU matching several statements when only one was intended, if a CONTINUE action is misplaced

Use Automation-Table Listings

Use the automation-table listing facility to determine where to place new statements within existing automation tables. You can also use a listing for problem determination to find syntax errors or incorrectly placed statements within a single automation table. A listing shows all included members, synonym values for synonym names, the levels of BEGIN-END sections, and date-and-time stamps. It also lists any errors found in the table. See "Example of an Automation-Table Listing" on page 236 for more information.

Use the ALWAYS Statement

Use the ALWAYS statement for an action or list of actions, such as:

- To stop automation processing at a certain point in the table

  For example, use ALWAYS as the last statement in a BEGIN-END section to prevent a possible incorrect statement match and to enhance performance. You can code the ALWAYS statement without actions (`ALWAYS;`) to stop automation processing for a message or MSU.
- To set defaults for a section of the table

  For example, you can take certain actions for a group of messages. An example of using ALWAYS, together with CONTINUE, for that purpose is shown in "Set Automation-Table Defaults" on page 236
- To facilitate testing

  For example, to analyze message frequency or to obtain an audit trail, you can log all instances of a certain group of messages for a period of time.

Use the CONTINUE Action Carefully

Use the CONTINUE action with great caution. Inappropriate use of CONTINUE can result in unintended actions, such as several commands or command lists being processed when you intended for only one to be processed.

The CONTINUE action is useful if you want to perform several actions for a message or MSU. Figure 63 logs all occurrences of command facility messages to a sequential log file to facilitate a frequency analysis.

```
* Process NetView command facility DSI messages
IF MSGID = 'DSI' . THEN
   BEGIN;
*     Temporary statement to send the message ID to a
*     sequential log file for frequency analysis.
*
*     The statement extracts the message ID in the
*     variable MSGIDVAR and calls the command processor
*     LOGMSGID, which uses NetView program's sequential log
*     facility to write the ID to a sequential file.
      IF MSGID = MSGIDVAR THEN
            EXEC (CMD('LOGMSGID ' MSGIDVAR))
            CONTINUE(Y);
*     Permanent statements that automate various DSI messages.
   :
   END;
```

Figure 63. Example of Using the CONTINUE Keyword

Set Automation-Table Defaults

Use CONTINUE and ALWAYS to set defaults for an automation table or automation-table section that can be overridden by a specific entry.

```
ALWAYS SYSLOG(Y) NETLOG(N) CONTINUE(Y);
```

Figure 64. Example of Using the CONTINUE Keyword on an ALWAYS Statement

The statement in Figure 64 causes all messages to go to the system log but not to the network log by default. If a message matches a later statement in the table, that statement must explicitly specify SYSLOG(N) or NETLOG(Y) to override the defaults set by the ALWAYS statement. Because SYSLOG and NETLOG are message-only actions, the statement in Figure 64 does not affect MSUs.

Limit Automation of Command Responses

Limit the testing for command responses in your automation tables because command response messages cannot be reliably tested for in an automation table. Commands can be issued from a PIPE or a CLIST that processes the responses and does not expose them to the automation table. These same commands can also be issued from another source that does result in the messages being exposed.

Automation as the NetView Program Closes

After a CLOSE STOP or CLOSE IMMED command is issued, any existing global keeps end and commands defined by ENDCMD statements are queued to the task defined by the endcmd.AutoTask statement in the CNMSTYLE member. All commands must complete within the time specified on the endcmd.close.leeway statement in the CNMSTYLE member. Some functions are not supported during this period. Avoid using any commands that involve long or indeterminate waits (for example commands directed to NetView TSO servers, RMTCMD commands, or WTOR commands). Message automation can continue for the period defined on the endcmd.close.leeway statement; automation cannot schedule new commands during this period.

Example of an Automation-Table Listing

This section shows an example of an automation table that is composed of two members, shown in Figure 65 and Figure 66 on page 237.

Figure 67 on page 238 shows the automation-table listing that you can generate from the members with the AUTOTBL LISTING function. Figure 67 on page 238 illustrates many of the ways a listing can give you information to help you code, tune, or debug an automation table.

The first member, shown in Figure 65, contains automation-table synonym statements to be included by the second member.

```
* Set table synonyms
SYN %MYDOMAIN% = '''CNM01''';
SYN %NETL3% = 'NETLOG(YES 3 +STATGRP)';
```

Figure 65. Example of Automation-Table Synonym Statements

The second member, shown in Figure 66 on page 237, is the main automation-table member, the one you specify on an AUTOTBL command.

```
* Include the member that contains the synonym definitions
%INCLUDE EXSYNS

* Set table defaults and continue processing
ALWAYS SYSLOG(Y) NETLOG(N) DISPLAY(Y)
      CONTINUE(Y);

* All DSI messages go here
IF MSGID = 'DSI' . THEN
   BEGIN;

* Invoke the PDFILTER command list automatically when the hardware
* monitor completes initialization.  (Use a synonym to check domain.)
      IF MSGID ='DSI530I' &
         TEXT = . 'BNJDSERV' . &
         DOMAINID = %MYDOMAIN% &
         TEXT = MESSAGETEXT THEN
            EXEC(CMD('PDFILTER ' MESSAGETEXT));

* Handle the DSI701I message.  (Use a synonym to specify the action.)
      IF MSGID = 'DSI701I' THEN
          %NETL3%;
   END;

* Any statements for CNM messages go here
IF MSGID = 'CNM' . THEN
   BEGIN;

* Suppress the CNM094I message
      IF MSGID='CNM094I' THEN
            DISPLAY(N)
            NETLOG(N);
   END;

* This is not a valid statement for handling syntax errors.
IF BADFUNC = 'INFO' THEN
      DISPLAY(N);
```

Figure 66. Example of a Main Automation-Table Member

You can generate a listing of the table shown in Figure 66 by issuing an AUTOTBL command with the LISTING keyword. Figure 67 on page 238 shows the resulting listing. The listing gives you several types of information about the automation table:

- Header lines indicate the AUTOTBL command that you issued, the task that ran the command, and the time.
- Start and end lines indicate where each member in the table begins and ends.
- An asterisk (*) in column 1 marks each comment line.
- Any synonyms that you defined are resolved. In the example, %MYDOMAIN% and %NETL3% are replaced with their values.
- The listing describes each statement in the table:
  - Columns 1 through 4 indicate the statement number.
  - Columns 6 through 8 indicate the BEGIN-END nesting level. For example, an 001 indicates a statement that is not within a BEGIN-END section, and an 002 indicates a statement in a first-level BEGIN-END section.
  - Columns 10 through 72 show the statement text.
  - Columns 73 through 80 show the sequence number, if any.
  - An error message follows each statement that is not valid.
- At the end of the listing is a line stating the total number of errors.

```
* Set table synonyms
0001 001 SYN %MYDOMAIN% = '''CNM01''';
0002 001 SYN %NETL3% = 'NETLOG(YES 3 +STATGRP)';
----------------------------  END  OF 'EXSYNS  ' --------------------
* Set table defaults and continue processing
0003 001 ALWAYS SYSLOG(Y) NETLOG(N) DISPLAY(Y) CONTINUE(Y);
* All DSI messages go here
0004 001 IF MSGID = 'DSI' . THEN BEGIN;
* Invoke the PDFILTER command list automatically when the hardware
* monitor completes initialization.  (Use a synonym to check domain.)
0005 002 IF MSGID ='DSI530I' & TEXT = . 'BNJDSERV' . & DOMAINID =
            'CNM01' & TEXT = MESSAGETEXT THEN EXEC(CMD('PDFILTER '
            MESSAGETEXT));
* Handle the DSI701I message.  (Use a synonym to specify the action.)
0006 002 IF MSGID = 'DSI701I' THEN NETLOG(YES 3 +STATGRP);
0007 002 END;
* Any statements for CNM messages go here
0008 001 IF MSGID = 'CNM' . THEN BEGIN;
* Suppress the CNM094I message
0009 002 IF MSGID='CNM094I' THEN DISPLAY(N) NETLOG(N);
0010 002 END;
* This invalid statement demonstrates handling of syntax errors
0011 001 IF BADFUNC = 'INFO' THEN DISPLAY(N);
ERROR    CNM505E INVALID FUNCTION NAME "BADFUNC" SPECIFIED IN
            CONDITIONAL
----------------------------  END  OF 'EXMAIN  ' ----------------------------
1  STATEMENT(S) IN ERROR
```

Figure 67. Example of an Automation-Table Listing

Automation-Table Usage Reports

NetView automation-table processing maintains a set of counters that track how many events are compared against a certain set of criteria, and how many cause automation actions to be run. You can use this information to fine-tune the automation table for your environment. Frequently matched statements can be moved toward the beginning of the table so that less checking takes place. You can examine statements that are never matched to determine whether they are to be deleted or changed because of logic errors.

You can also use the usage report to determine the level of automation taking place in your system.

The AUTOCNT Command

The AUTOCNT command produces a report describing the use of automation-table statements in either an active NetView automation table or the NetView automation table that is being tested with the AUTOTEST command. You can also use the AUTOCNT command to reset the automation-table statement usage counters.

The AUTOCNT command can request information and statistics for message-type automation statements, MSU-type automation statements, or both. You can request summary information or detailed information. The detailed information describes how many messages and MSUs were compared to each automation-table statement, and how many matched.

"Example of Usage Reports Output" on page 239 contains an example of an automation-table usage report and illustrates the differences between a summary

report and a detailed report. You can also view this information using the automation-table management (AUTOMAN) function.

You can display the information and statistics as multiline messages. You can also place the information in a file.

Use the online command help for the syntax and parameter descriptions of the AUTOCNT command.

Example of Usage Reports Output

This section includes an example of an automation-table usage report. The source automation-table member is shown, followed by detailed and summary usage reports. You can use the source automation member to determine which statements the detailed usage statistics refer to by matching the sequence number (SEQ NUMBER) and member name (MEMBER NAME) fields of the usage report with the source member. You can also determine what sections of an active automation table have been disabled. You can use an automation-table listing for statement correlation by matching the statement number (STMT NUMBER) field with the listing statement numbers.

A separate multiline message can be generated for each of these types of reports:

- A detailed report for all message-type statements in the active automation table
- A detailed report for all MSU-type statements in the active automation table
- A summary report for all message-type statements in the active automation table
- A summary report for all MSU-type statements in the active automation table

The detailed reports show usage information for each statement in the table and can be used for:

- Tuning the automation table
- Identifying statements with logic errors that cause them to never match or always match
- Testing new statements
- Determining specific message and MSU traffic

The summary reports show total usage information for the entire active automation table and can be used for:

- Capacity planning
- Determining the results of adding new automation
- Trend analysis
- Determining general message and MSU traffic

You can generate usage reports by using the AUTOCNT command for message-type statements, MSU-type statements, or both. Specifying STATS=SUMMARY on the AUTOCNT command provides summary-only reports. Specifying STATS=DETAIL on the AUTOCNT command provides both detailed and summary reports.

Figure 68 on page 240 is an example of an automation-table member (showing sequence numbers) for an automation table that was activated one hour ago. Figure 69 on page 241 is the automation-table listing (showing statement numbers).

```
*************************************************************************
*        BEGINNING OF 'IST' MSGID'S                                     *
*************************************************************************
*                                                                       *
IF MSGID = 'IST' . THEN
  BEGIN;
    IF MSGID = 'IST097I' |
       MSGID = 'IST314I' |
       MSGID = 'IST526I' THEN
         DISPLAY(N) NETLOG(N) SYSLOG(N);
    IF MSGID = 'IST259I' THEN
         EXEC(CMD('INOPRU') ROUTE(ONE AUTOVTAM AUTO1 * PPT));
    IF (LABEL: REACTIVATEVTAM)
       MSGID = 'IST102I' &
       ATF('DSICGLOB VTAMDESIRED') = 'ACTIVE' THEN
         COLOR(PIN)
         EXEC(CMD('VTAMSTRT') ROUTE(ONE AUTOVTAM AUTO1 * PPT));
    ALWAYS;
  END;
*                                                                       *
*************************************************************************
*      BEGINNING OF 'DSI' MESSAGES                                      *
*************************************************************************
*                                                                       *
IF (LABEL: DSIPREFIX) MSGID = 'DSI' . THEN                               SEQ00001
  BEGIN;
    IF (LABEL: ALTDSIPREFIX)
       MSGID = 'DSI034I' |
       MSGID = 'DSI201I' |
       MSGID = 'DSI208I' |
       MSGID = 'DSI633I' THEN
         DISPLAY(N) NETLOG(N) SYSLOG(N);
    IF (ENDLABEL: ALTDSIPREFIX)
       MSGID='DSI374A' THEN
         HOLD(Y) BEEP(Y) DISPLAY(Y)
         EXEC(ROUTE(ALL * +GRPOPS));
    ALWAYS;
  END;
*                                                                       *
*************************************************************************
*      BEGINNING OF MSU SECTION                                         *
*************************************************************************
*                                                                       *
  IF MSUSEG(0000) ¬= '' THEN                                               SEQ00002
    BEGIN;
*   PROBABLE CAUSE: COMMUNICATION CONTROLLER OR TERMINAL CONTROL UNIT
    IF (GROUP: MSUSTATEMENT)
       MSUSEG(0000.93 3) = HEX('3111') . |
       MSUSEG(0000.93 3) = HEX('3121') . THEN
        COLOR(RED) XHILITE(REV)
        SRF(ROUTE PASS);
*   PROBABLE CAUSE: LAN COMPONENT OR LAN ADAPTER
    IF (GROUP: MSUSTATEMENT)
       MSUSEG(0000.93 3) = HEX('37') . |
       MSUSEG(0000.93 3) = HEX('332') . THEN
        COLOR(PIN) XHILITE(REV)
        SRF(ROUTE PASS);
    ALWAYS;
  END;
```

Figure 68. Automation-Table Member

```
---------------------------- START  OF 'AUTOSEG1' ----------------------------
*************************************************************************
*        BEGINNING OF 'IST' MSGID'S                                     *
*************************************************************************
*                                                                       *
0001 001 IF MSGID = 'IST' . THEN BEGIN;
0002 002 IF MSGID = 'IST097I' | MSGID = 'IST314I' | MSGID = 'IST526I'
            THEN DISPLAY(N) NETLOG(N) SYSLOG(N);
0003 002 IF MSGID = 'IST259I' THEN EXEC(CMD('INOPRU') ROUTE(ONE
            AUTOVTAM AUTO1 * PPT));
0004 002 IF (LABEL: REACTIVATEVTAM) MSGID = 'IST102I' & ATF('DSICGLOB
         VTAMDESIRED') = 'ACTIVE' THEN COLOR(PIN)
         EXEC(CMD('VTAMSTRT') ROUTE(ONE AUTOVTAM AUTO1 * PPT));
0005 002 ALWAYS;
0006 002 END;
*                                                                       *
*************************************************************************
*      BEGINNING OF 'DSI' MESSAGES                                      *
*************************************************************************
*                                                                       *
0007 001 IF (LABEL: DSIPREFIX) MSGID = 'DSI' . THEN BEGIN;               SEQ00001
0008 002 IF (LABEL: ALTDSIPREFIX) MSGID = 'DSI034I' | MSGID =
            'DSI201I' | MSGID = 'DSI208I' | MSGID = 'DSI633I' THEN
             DISPLAY(N) NETLOG(N) SYSLOG(N);
0009 002 IF (ENDLABEL: ALTDSIPREFIX) MSGID='DSI374A' THEN HOLD(Y)
         BEEP(Y) DISPLAY(Y)  EXEC(ROUTE(ALL * +GRPOPS));
0010 002 ALWAYS;
0011 002 END;
*                                                                       *
*************************************************************************
*      BEGINNING OF MSU SECTION                                         *
*************************************************************************
*                                                                       *
0012 001 IF MSUSEG(0000) ¬¬= '' THEN BEGIN;                               SEQ00002
*   PROBABLE CAUSE: COMMUNICATION CONTROLLER OR TERMINAL CONTROL UNIT
0013 002 IF (GROUP: MSUSTATEMENT) MSUSEG(0000.93 3) = HEX('3111') . |
            MSUSEG(0000.93 3) = HEX('3121') . THEN COLOR(RED)
            XHILITE(REV) SRF(ROUTE PASS);
*   PROBABLE CAUSE: LAN COMPONENT OR LAN ADAPTER
0014 002 IF (GROUP: MSUSTATEMENT) MSUSEG(0000.93 3) = HEX('37') . |
         MSUSEG(0000.93 3) = HEX('332') . THEN COLOR(PIN)
         XHILITE(REV) SRF(ROUTE PASS);
0015 002 ALWAYS;
0016 002 END;
----------------------------  END  OF 'AUTOSEG1' ----------------------------

0  STATEMENT(S) IN ERROR
```

Figure 69. Automation-Table Listing for the Sample Member

Assumptions of Message and MSU Processing for This Example

For this example, it is assumed that during the past hour, while automation-table usage statistics were being kept, no statements were disabled. The automation table processed these messages:

Message Type	Number of Messages
IST097I messages	154
IST314I messages	20
IST526I messages	3
IST259I messages	9
IST102I messages	0

Message Type	Number of Messages
Other IST prefix messages	612
DSI034I messages	3
DSI201I messages	3
DSI208I messages	39
DSI633I messages	7
DSI374A messages	1
Other DSI prefix messages	107
Messages not prefixed by IST or DSI	1346

The automation table also processed these MSUs:

MSU Type	Number of MSUs
Alert major vectors with a probable cause of X'3111'	2
Alert major vectors with a probable cause of X'3121'	3
Alert major vectors with a probable cause of X'37'	14
Alert major vectors with a probable cause of X'332'	3
Other alert major vectors not including the above	3211
Nonalert MSUs	130

Detailed Automation-Table Usage Report

Detailed automation-table usage reports contain this information for each automation-table statement in the active NetView automation table.

- Statement number (STMT NUMBER)

  The sequential number of the statement in the automation table. You can also find this number in the automation-table listing, which provides correlation if you have a matching listing.
- Label indicator (L I)

  This column contains a character that indicates the type of label specified or whether a sequence number is specified if no label is present. These values for the indicator are possible:

  (blank) No label or sequence number is specified for this statement.

  S No label is specified for this statement. However, a sequence number is specified. The sequence number is found in the next column.

  L Regardless of any sequence number specified, there is a LABEL specification for this statement. The LABEL name is found in the next column.

  E Regardless of any sequence number specified, there is an ENDLABEL specification for this statement. The ENDLABEL name is found in the next column.

B Regardless of any sequence number specified, there is a LABEL specification for this statement that matches a subsequent ENDLABEL specification and, therefore, specifies a BLOCK of statements. The BLOCK name is found in the next column.

G Regardless of any sequence number specified, there is a GROUP specification for this statement. The GROUP name is found in the next column.

- Sequence number or label name (SEQUENCE NUMBER/ LABEL NAME)

  This column contains one of these values:

  (blank) No sequence number or label name was specified for this statement.

  (sequence number) A sequence number was specified for this statement without a label name specification.

  (label name) A label name that shows the value specified on the LABEL, ENDLABEL, or GROUP specification for this statement.

- Member name (MEMBER NAME)

  The member name where the statement is located. This, along with the sequence number, provides correlation with the source automation-table members or files.

- Conditional comparisons (COMPARE COUNT)

  The counter that is incremented when the associated conditional statement is selected for evaluation.

- Evaluation matches (MATCH COUNT)

  The counter that is incremented when the associated conditional statement is evaluated as true, resulting in performance of all automation actions specified on the statement.

- Executed commands (E C)

  This column reports the number of commands that are run for this automation statement when there is an evaluation match. If the number of EXEC actions with CMD keywords is greater than 99, an asterisk (*) appears in the column.

- Continue indicator (C I)

  A report column marked X indicates that the conditional statement contained a CONTINUE action, causing the NetView program to continue to scan the automation table. CONTINUE(Y) actions cause additional conditional processing for later statements in the table, and can enable a conditional match on additional statements.

- Always statement indicator (A I)

  A report column marked X indicates that the statement was an ALWAYS. For ALWAYS statements, the MATCH/COMP field is always 100%.

- Disable indicator (D I)

  This column describes whether the statement is currently part of a DISABLE request or whether it was part of a DISABLE request since the last time usage statistics were reset. The possible values are:

  (blank) The statement has not been part of a DISABLE request since the last time usage statistics were reset.

  d The individual statement has been disabled since the last time usage statistics were reset, but is not currently disabled.

b The block of statements has been disabled since the last time usage statistics were reset, but is not currently disabled.

S The statement is currently disabled using its sequence number.

L The statement is currently disabled using a LABEL request.

E The statement is currently disabled using an ENDLABEL request.

B The statement is currently disabled using a BLOCK request.

G The statement is currently disabled using a GROUP request.

Note: To view the status of individual automation-table statements, blocks, or groups, use the automation-table management (AUTOMAN) function.

- Match to compare percentage (MATCH/COMP)

  A statistic calculated by dividing the ratio of MATCH COUNT by the COMPARE COUNT of the conditional statement, multiplied by 100. If the number of matches and the number of comparisons are both zero, the ratio is shown as **-.-** to indicate division by zero.
- Compare percentage (COMP/TOTAL)

  A statistic calculated by dividing the ratio of COMPARE COUNT of the conditional statement by the total number of messages (or MSUs), multiplied by 100. If the number of comparisons against this statement and the total number of messages or MSUs processed by automation are both zero, the ratio is shown as **-.-** to indicate division by zero.
- Match percentage (MATCH/TOTAL)

  A statistic calculated by dividing the ratio of MATCH COUNT of the conditional statement by the total number of messages (or MSUs), multiplied by 100. If the number of matches for this statement and the total number of messages or MSUs processed by automation are both zero, the ratio is shown as **-.-** to indicate division by zero.

Any numeric column value that exceeds 99999999 is overwritten with eight asterisks (*).

Figure 70 and Figure 71 on page 245 illustrate the output of a detailed report.

```
----------------( AUTOSEG1 MESSAGE DETAILS  09/02/10 14:21:46 )----------------
                                                              |-- PERCENTAGES --|
STMT L SEQUENCE NUMBER/ MEMBER    COMPARE    MATCH  E C A D MATCH/ COMP/ MATCH/
NUMB I LABEL NAME       NAME      COUNT      COUNT  C I I I COMP   TOTAL TOTAL
--------------------------------------------------------------------------------
0001                    AUTOSEG1     2304      798  0        34.6  100.0  34.6
0002                    AUTOSEG1      798      177  0        22.2   34.6   7.7
0003                    AUTOSEG1      621        9  1         1.4   27.0   0.4
0004 L REACTIVATEVTAM   AUTOSEG1      612        0  1         0.0   26.6   0.0
0005                    AUTOSEG1      612      612  0   X   100.0   26.6  26.6
0007 L DSIPREFIX        AUTOSEG1     1506      160  0        10.6   65.4   6.9
0008 L ALTDSIPREFIX     AUTOSEG1      160       52  0        32.5    6.9   2.3
0009 E ALTDSIPREFIX     AUTOSEG1      108        1  0         0.9    4.7   0.0
0010                    AUTOSEG1      107      107  0   X   100.0    4.6   4.6
--------------------------------------------------------------------------------
```

Figure 70. MSG Detail Report

```
----------------( AUTOSEG1 MSU DETAILS  09/02/10 14:21:46 )------------------
                                                            |-- PERCENTAGES --|
STMT L SEQUENCE NUMBER/ MEMBER    COMPARE    MATCH  E C A D MATCH/ COMP/ MATCH/
NUMB I LABEL NAME       NAME      COUNT      COUNT  C I I I COMP   TOTAL TOTAL
--------------------------------------------------------------------------------
0012 S SEQ00002         AUTOSEG1     3363     3233           96.1  100.0  96.1
0013 G MSUSTATEMENT     AUTOSEG1     3233        5            0.2   96.1   0.1
0014 G MSUSTATEMENT     AUTOSEG1     3228       17            0.5   96.0   0.5
0015                    AUTOSEG1     3211     3211      X   100.0   95.5  95.5
--------------------------------------------------------------------------------
```

Figure 71. MSU Detail Report

The detailed statistics can indicate the effect of each statement on automation processing. Examine the comparison and match counts to determine the optimal order of automation statements. Generally, the statements with the highest match counts should be near the beginning of their BEGIN-END sections. Likewise, BEGIN-END sections with the highest total match counts for all statements within the BEGIN-END section should be near the beginning of the automation table.

Move statements with care, ensuring that the sequential logic of the table is not affected.

```
IF MSGID = 'XYZ123I' & TOKEN(5) = 'DEVICE1' THEN
    EXEC(CMD('COMMAND1'));
IF MSGID = 'XYZ456I' & DOMAINID = 'CNM99' THEN
    EXEC(CMD('COMMAND2'));
IF MSGID = 'XYZ123I' | MSGID = 'XYZ456I' THEN
    EXEC(CMD('COMMAND3'));
```

Figure 72. Statements Evaluated with Usage Statistics

Consider the statements in Figure 72. Even if the usage statistics show that the third statement is matched more frequently than the first and second statements, moving the third statement sequentially ahead of the first two affects the automation actions performed. The statements that call COMMAND1 and COMMAND2 never match, because the statement that calls COMMAND3 always matches the messages first.

Examine IF-THEN statements that always match to determine whether there are logic errors in the statement specifications. Examine IF-THEN statements that never match to determine whether:

- There is a logic error in the statement specification
- There is a statement preceding the statement in question that prevents this statement from getting its intended messages or MSUs
- The statement is no longer required, and therefore can be removed

Examine the comparison and match counts to determine the optimal order of automation statements. When you add new automation statements, the COMPARE and MATCH COUNTS can indicate part of the effect of the addition.

A high number of matches for a statement that contains one or more command invocations can indicate excessive CPU processing for issuing the commands. If the commands being issued are command lists, consider preloading them using the LOADCL command.

Summary Automation-Table Usage Report: Summary automation-table usage reports contain this information for all message-type or MSU-type statements in the active NetView automation table.

- Date and time of usage report generation

  The date is in the format *mm/dd/yy*. The time is in the format *hh:mm:ss*, where *hh* is based on a 24-hour clock. The date and time are reported in the label message for the SUMMARY statistics (messages DWO810I and DWO811I).
- Date and time of start of usage count monitoring

  The date is in the format *mm/dd/yy*. The time is in the format *hh:mm:ss*, where *hh* is based on a 24-hour clock. The date and time are reported in message DWO812I.
- Total number of messages or MSUs processed

  A count of all the messages or MSUs that have passed through the automation table.
- Total number of messages or MSUs matched

  The number of messages or MSUs that were acted upon by at least one automation-table statement. An ALWAYS statement causes a message or MSU to be considered a match.
- Number of messages or MSUs resulting in command execution

  A count of the number of messages or MSUs that resulted in one or more commands being run from automation-table statements.
- Total commands run for messages or MSUs

  The total number of commands run by all automation-table statements during the period when statistics were taken. The EXEC action with the CMD keyword indicates a command that is run from the automation table.
- Total routes run for messages

  The total number of routes run by all automation-table statements during the period when statistics were taken. The EXEC action with the ROUTE keyword (and without the CMD keyword) indicates that a route is run from the automation table.
- Average number of compares per message or MSU

  The number of compares divided by the number of messages or MSUs that had passed through the automation table.
- Average number of messages or MSUs processed per minute

  The number of messages or MSUs processed by the NetView automation table divided by the number of minutes since the last reset or load of the automation table.
- Number of minutes elapsed

  The amount of time, in minutes, since the last AUTOCNT RESET command or since the current active automation table was activated.

Figure 73 on page 247 and Figure 74 on page 247 illustrate the output of a summary report:

```
- AUTOMATION TABLE MSG SUMMARY REPORT BY OPER1
 ------------( AUTOSEG1 MESSAGE SUMMARY  09/02/10 14:21:46 )------------
 STATISTICS STARTED          = 09/02/10  13:58:34
 TOTAL MSGS PROCESSED        =     2304
 MSGS MATCHED                =      958
 MSGS RESULTING IN COMMANDS  =        9
 TOTAL COMMANDS EXECUTED     =        9
 TOTAL ROUTES EXECUTED       =        1
 AVERAGE COMPARES/MSG        =        2.58
 TOTAL MSGS/MINUTE           =       38
 MINUTES ELAPSED             =       60
 ---------------------------------------------------------------------------
```

Figure 73. MSG Summary Report for Message Automation

```
-  AUTOMATION TABLE MSU SUMMARY REPORT BY OPER1
  --------------( AUTOSEG1 MSU SUMMARY  09/02/10 14:21:46 )--------------
 STATISTICS STARTED          = 09/02/10  13:58:34
 TOTAL MSUS PROCESSED        =     3363
 MSUS MATCHED                =     3233
 MSUS RESULTING IN COMMANDS  =        0
 TOTAL COMMANDS EXECUTED     =        0
 AVERAGE COMPARES/MSU        =        2.92
 TOTAL MSUS/MINUTE           =       56
 MINUTES ELAPSED             =       60
 ---------------------------------------------------------------------------
```

Figure 74. MSU Summary Report for MSU Automation

The summary statistics can indicate how effective and efficient your automation processing is:

- The number of messages or MSUs per minute can indicate the automation processing load.
- The average compares per message or MSU indicate how much automation processing time is taken to determine what, if any, automation-table actions to take.

  The smaller the average compares figure is, the smaller the CPU use by automation processing of messages and MSUs. You can generally reduce the average compares figure by adding BEGIN-END sections or combining multiple statements.

  A high number of messages that are not matched might indicate that one of these activities is to be performed:

  – Add automation statements
  – Improve efficiency of the operating system message processing facility to prevent messages from undergoing automation processing
  – Suppress more messages

The summary statistics are especially useful for historical purposes so that you can see the effect of:

- Adding more devices to the network
- Adding more automation statements to your automation table
- Using different automation tables
- Changes in shifts or days on your overall automation processing

Keeping historical statistics can be useful for capacity planning and system stress analysis.

General Reminders about Automation-Table Usage Reports: The automation-table usage report is based on the usage for the current automation table, such as:

- Table activated by the AUTOTBL command
- Table being tested using the AUTOTEST command

To ensure usage reports are correlated with automation-table statements:

- Allow changes to the source members only at certain times and save a backup copy before making changes.
- Generate an automation-table listing when you activate a new automation table.

When you activate an automation-table, the counters that the AUTOCNT command uses are set to zero. If you generate an automation-table listing when you activate the automation table and if no AUTOCNT RESET command is issued (between the time when the automation table is activated and the time when a usage report is generated, the date and time indicated in the listing match the STATISTICS STARTED date and time in the summary usage report. Comparing the dates and times is one way you can verify that you have a correlation between the detailed usage report statements and the actual automation statements.

Some statements can be both message-type and MSU-type. If this is the case, the statement has both message and MSU usage statistics associated with it, and is listed separately in both the message detailed usage report and the MSU detailed usage report.

The usage report statistics might not be exact, because messages and MSUs continue to be processed by the automation table when the AUTOCNT command is run. Messages or MSUs currently undergoing automation processing might be reflected in the detailed usage statistics. The usage statistics are used to identify general trends, not as precise data.

Managing Multiple Automation Tables

The AUTOTBL command enables you to load multiple automation tables. An automation table, typically, is made up of many included members. The automation-table management (AUTOMAN) command enables you to make changes to selected tables or changes that have an effect on all automation tables. To help you work with automation tables, AUTOMAN provides a full-screen panel interface.

AUTOMAN and the full-screen panel interface enable you to do the following tasks:

- View and manage single or multiple automation tables
- Enable or disable individual automation tables or statements
- View existing tables and their status

Getting Started

AUTOMAN provides individual table commands and global commands. The individual table commands apply to one or more selected tables, and global commands apply to all automation tables. See these features and options of each type of command:

- With individual table commands, you can enable or disable automation tables. You can also enable or disable automation-table statements, as shown here:
  - Sequence number

- Label
- End label
- Block
- Group
- Include

With individual table commands, you can also issue these requests:

- Display disabled statements
- Display labels, blocks, and groups
- Load or unload tables
- Test tables
- Display the %INCLUDE structure
- Display synonyms

- With global commands, you can enable, disable, or unload automation tables. You can enable disabled statements or enable and disable blocks, groups, and labels. Global commands affect all automation tables.

  Automation statements can be enabled or disabled across all tables based on these features:

  - Label
  - Block
  - Group

  With global commands, you can also issue requests for the following tasks:

  - Locate disabled statements
  - Display labels, blocks, and groups
  - Display the %INCLUDE structure

Using Automation-Table Management

To use the AUTOMAN command, follow the steps and panel descriptions in this section.

From the command line, enter **AUTOMAN**. The panel in Figure 75 is displayed. This panel enables you to see your automation-table structure and take action as necessary.

```
EZLK8500                Automation Table Management

AUTOMATION TABLE          Enter any character in the selection fields
SEL     POS     NAME        STATUS     MARKERS     TASK     DATE     TIME
 _       1      DISTABLE    ENABLED                NETOP2   03/18/11 13:15:24
 _       2      DSITBL01    ENABLED    (AON)       NETOP2   03/18/11 13:11:09









Command ===>
F1=Help      F2=Main Menu  F3=Return     F4=Commands  F5=Refresh  F6=Roll
F7=Backward  F8=Forward    F9=Responses  F10=Global Commands      F12=Cancel
```

Figure 75. Automation-Table Structure

In the previous figure, the status of all loaded tables is displayed. The fields in this figure are described as follows:

SEL Enter any character in this field. You can select multiple tables to be acted upon. If your cursor is in a selection field shown in Figure 75 on page 249, and you press **F4** or **Enter**, the COMMANDS pop-up window is displayed. A forward slash is automatically placed in the SEL field for your reference.

POS Displays the numeric position of each table.

NAME Contains the name of loaded tables.

STATUS
One of these statuses is displayed in this field:

ENABLED
The table is loaded and active. This selection is green.

DISABLED
The table is loaded but disabled. This selection is red.

ALTERED
The table is loaded and enabled, but contains at least one disabled statement. This selection is yellow.

MARKERS
Shows the marker you designated for each table. This field includes (FIRST), (LAST), or (AON) if the table is so marked. The indicators in this field are set by AUTOMAN.

TASK The name of the task that loaded the table.

DATE The date when the table was loaded.

TIME The time when the table was loaded.

Using Commands for Selected Tables

The Commands pop-up window in Figure 76 on page 251 provides options to help you work with one or more selected automation tables. In this figure, options 1–6 apply to one or more selected tables (in contrast to global commands in Figure 79 on page 254, which apply to all tables). Options 7–8 apply to only one table.

At the Automation Table Management panel, shown in Figure 75 on page 249, pressing **F4** displays the Commands pop-up menuFigure 76 on page 251, where DSITBL01 is selected to be disabled.

Selecting option 2 displays a pop-up menu where you can confirm that you want to disable the selected table. After DSITBL01 is disabled, a message indicates if the command was successful or if failures were detected. Press **F9** in Figure 75 on page 249 to view the results of your command.

```
EZLKATBC             AUTOMATION TABLE MANAGEMENT

AUTOMATION TABLE       Enter any character in the selection fields
SEL  POS  NAME    ................................................
/     1   DSITBL0 : COMMANDS (Choose a highlighted command option)    :
                  :                                                   :
                  :  _     1 -ENABLE the selected tables              :
                  :  _     2 -DISABLE the selected tables             :
                  :  _     3 -RELOAD the selected tables              :
                  :  _     4 -RELOAD and REINSTATE disabled elements  :
                  :  _     5 -TEST the selected tables                :
                  :  _     6 -ENABLE/DISABLE parts of the selected    :
                  :           tables                                  :
                  :  _     7 -UNLOAD selected tables                  :
                  :  _     8 -INSERT a table                          :
                  :        9 -DISPLAY options                         :
                  :                                                   :
                  : Enter=Execute Command         F3 or F12=Cancel    :
                  :...................................................:

Command ===>
F1=Help      F2=Main Menu                                          F6=Roll
                                                                  F12=Cancel
```

Figure 76. Automation-Table Management Commands Popup

These commands are available in the COMMANDS pop-up menu shown in Figure 76:

1 Enables the selected tables.

2 Disables the selected tables.

3 Reloads the selected tables.

4 Reloads selected tables and reinstates all disabled elements.

5 Tests the selected tables.

6 Enables or disables parts of the selected tables.

7 Unloads or removes the selected tables.

8 Displays the panel where new tables (that are based on the currently selected table) can be inserted. See Figure 77 on page 252.

9 Displays the panel where other display options are available for automation tables. See Figure 78 on page 253.

Inserting an Automation Table: If you selected **8** in Figure 76, this panel is displayed, where you can insert a new automation table:

```
EZLKATBI             AUTOMATION TABLE MANAGEMENT

AUTOMATION TABLE INSERT PANEL (Press Enter to process INSERT request)

 Preceding Table        Focus Table AT= 1      Next Table
 ---------------------- ---------------------- ----------------------
 N/A                    DSITBL01               N/A

 SELECT INSERT OR TEST OPTION

    1 - AT(DEFAULT)       Table Name              (Required)
    2 - AFTER             Listing Name  AUTOM819  Default listing name
    3 - BEFORE                                    (names can be reused,
    4 - REPLACE                                    but cannot be in use
    5 - FIRST                                      by another table)
    6 - LAST              SELECT A MARKER OPTION
 OR
    7 - AUTOTBL TEST       _   Enter your own marker  ________
                           _   Mark as AON's table    ________


Command ===>
F1=Help   F3=Return                                          F6=Roll
                                                            F12=Cancel
```

Figure 77. Automation-Table Management Insert Option

The automation-table INSERT option is used to insert tables based on the INSERT command you chose in Figure 76 on page 251 and the `focus table` that was selected in Figure 75 on page 249.

The insert panel displays the name of the focus table and its numeric position. To the left of the focus table is the name of the `Preceding Table` and to the right is the name of the `Next Table`. If there are no tables in those positions, `N/A` is displayed. Using this information, you can specify the INSERT option as follows:

1 - AT Inserts a new table in the same position as the focus table.

The focus table is moved to the next position. You cannot insert a table using the `AT` option if the focus table is marked as `FIRST`.

2 - AFTER Inserts a new table in the position following the focus table.

If the focus table is marked as `LAST`, you cannot insert a table using the AFTER option.

3 - BEFORE Inserts a new table before the focus table. This request has the same result as the default `AT`.

4 - REPLACE Replaces the focus table with the new table.

This function has the same result as the RELOAD option in Figure 76 on page 251.

If the focus table is marked as `FIRST` or `LAST`, you cannot specify the REPLACE option unless the tables have the same name.

5 - FIRST Inserts a new table and marks it as `FIRST`.

You cannot specify this option if another table is marked as `FIRST` or if the current focus table is not the first table located at position 1.

6 - LAST Inserts a new table and marks it as `LAST`.

You cannot specify this option if another table is marked as `LAST` or the current focus table is not the last table listed.

7 - AUTOTBL TEST
Performs an AUTOTBL test on the table name specified.

The table is not loaded. The other insert fields are ignored.

In Figure 77 on page 252, the following fields must be noted:

Table Name Enter the name (1–8 characters) of the automation table to be inserted.

Listing Name The unique identifier for the listing member, which is required for automation-table management.

Enter the name of the listing member (1–8 characters) for the specified automation table. A unique listing member name is provided by default, but can be overridden. Listing names can be reused, but must not currently be in use.

To enter markers or identifiers, type any character in the **Enter your own marker** field. Enter the text of the marker in the space following the field. You can select either a custom marker or an AON marker, but not both. The table being used by AON must be marked appropriately. If AON is not present, the `Mark as AON's table` is not displayed.

Press **Enter** when all fields are complete and you are ready to insert the new table.

Using the Display Options Pop-up window: If you select **9** from the COMMANDS pop-up menu (see Figure 76 on page 251), the following DISPLAY OPTIONS pop-up window is displayed:

```
EZLKATLD             AUTOMATION TABLE MANAGEMENT

AUTOMATION TABLE       Enter any character in the selection fields
SEL  POS  NAME      .................................................
/     1   DSITBL01  : COMMANDS (Choose a highlighted command option)  :
                    :                                                 :
                    :  9     1 -ENABLE the selected tables            :
          .........................................................
          : DISPLAY OPTIONS (select option and press ENTER)           :
          :                                                           :
          :  _   1 - Browse table with includes     8 - Browse listing :
          :      2 - Browse table without includes                    :
          :      3 - Display INCLUDE structure                        :
          :      4 - Display synonyms                                 :
          :      5 - Display labels/blocks/groups                     :
          :      6 - Display disabled statements                      :
          :      7 - Display AUTOCNT statistics                       :
          :...........................................................:


Command ===>
F1=Help      F2=Main Menu    F3=Return                          F6=Roll
                                                               F12=Cancel
```

Figure 78. Automation-Table Management Display Options Pop-up Window

In Figure 78, the DISPLAY functions act only on a single automation table. The table in this panel was selected in Figure 75 on page 249. You can choose the following options on the DISPLAY OPTIONS panel:

1 Invokes the BROWSE command for the selected table with the default XINCL option.

2 Invokes the BROWSE command for the selected table with the NOINCL option.

3 The %INCLUDE structure is displayed using the WINDOW command. Each INCLUDE level is indented and color-coded. Refer to the NetView online help for more information about the WINDOW command.

4 Displays the same %INCLUDE structure as option 3 with synonyms included.

5 Displays a new panel where only label, block, and group names are displayed.

The new panel provides additional enabling and disabling functions.

6 Displays the panel in Figure 81 on page 256, but displays only the disabled statements in the selected table.

7 Displays the results of the following command in a WINDOW:

```
AUTOCNT REPORT=BOTH,STATS=DETAIL,NAME=(selected table)
```

8 Invokes the BROWSE command for the selected table listing file.

Using Global Commands

Global commands apply to all tables. To use a global command, press **F10** at the panel that is shown in Figure 75 on page 249. The following pop-up menu is displayed:

```
EZLKATBG             AUTOMATION TABLE MANAGEMENT

AUTOMATION TABLE       Enter any character in the selection fields
SEL  POS  NAME      .....................................................
_     1   DSITBL01  : GLOBAL COMMANDS (Choose a command option)           :
                    :                                                     :
                    :  _     1 -Enable all tables                         :
                    :        2 -Disable all tables (must confirm)         :
                    :        3 -Unload all tables (must confirm)          :
                    :        4 -Display options                           :
                    :                                                     :
                    : Enter=Execute Command       F3 or F12=Cancel        :
                    :.....................................................:









Command ===>
F1=Help      F2=Main Menu                                          F6=Roll
                                                                  F12=Cancel
```

Figure 79. Automation-Table Management Global Commands Popup

When the GLOBAL COMMANDS pop-up menu is displayed, you can choose from the following options:

1 Turn on all tables.

2 Turn off all tables. (See note 2.)

3 Remove all tables from memory. (See note 2.)

4 Shows the GLOBAL DISPLAY OPTIONS popup.

Any option selected from this pop-up menu applies to all tables.

Usage notes:

1. Characters that you typed in the SEL fields are ignored when the GLOBAL COMMANDS pop-up window is displayed.
2. If you select the global command to UNLOAD or DISABLE all automation tables, you receive a confirmation panel asking you to confirm whether you want to disable or remove the tables or cancel the operation. See "The Confirmation Panel" on page 258 for more information.

Using the Global Display Panel: If you select option **4** in Figure 79 on page 254, the following GLOBAL DISPLAY OPTIONS pop-up window is displayed:

```
EZLKATGD             AUTOMATION TABLE MANAGEMENT

AUTOMATION TABLE       Enter any character in the selection fields
SEL  POS  NAME      ..................................................
_     1   DSITBL01  : GLOBAL COMMANDS                                 :
                    :                                                 :
                    :        1 -ENABLE ALL TABLES                     :
          ..........................................................
          : GLOBAL DISPLAY OPTIONS  (select option and press ENTER)   :
          :                                                           :
          :  _  1 - Display Table structure (showing INCLUDES)        :
          :     2 - Display ALL synonyms                              :
          :     3 - Display ALL disabled statements                   :
          :     4 - Display ALL Labels/Blocks/Groups                  :
          :.........................................................:




Command ===>
F1=Help      F2=Main Menu    F3=Return                               F6=Roll
                                                                    F12=Cancel
```

Figure 80. Automation-Table Management Global Display Options Popup

In the previous panel, you can select the following options:

1 Displays the %INCLUDE table structure using the WINDOW command.

Each nesting level is displayed in a different color and indentation. Each table loaded by an AUTOTBL command is displayed in column one followed by the text *primary table*.

2 Displays the %INCLUDE table structure and a list of the synonyms at each level.

3 Displays the panel in Figure 81 on page 256, but shows only disabled statements.

4 Displays the label, group, and block names for all tables.

In the displayed panel, you can enable or disable these names in all automation tables. Enabling or disabling any label, group, or block from this panel results in a global action.

Enabling and Disabling Automation-Table Statements: In Figure 81 on page 256, the commands and information are gathered using the AUTOCNT STATISTICS and the listing. This panel provides global display functions that act on all loaded automation tables.

```
EZLKENDI            AUTOMATION TABLE MANAGEMENT              More:

AUTOMATION TABLE  (Place the cursor on a statement and select an action)
MEMBER    LVL  STATEMENT
FLBAUT    001  IF MSGID = 'FLB447I' THEN EXEC(CMD('PURGE TIMER=FLBTOPO'
FLBAUT    001  ROUTE(ONE AUTO1)) CONTINUE(Y);
********  001  END OF FLBAUT
DSITBL01  001  IF MSGID = 'DWO' . & TEXT=MESSAGE THEN BEGIN;
DSITBL01  002  IF MSGID = 'DWO207I' THEN HOLD(Y);
********  002  START OF EZLDWOAO
EZLDWOAO  002  IF MSGID ='DWO336I' & DOMAINID = DOMID & OPID = ORIGOP THEN
EZLDWOAO  002  EXEC(CMD('EZLE1IXL 'DOMID' 'ORIGOP' 'MESSAGE) ROUTE(ALL
EZLDWOAO  002  PPT)) DISPLAY(Y) NETLOG(Y) SYSLOG(N);
EZLDWOAO  002  IF MSGID = 'DWO575I' & DOMAINID = DOMID & OPID = ORIGOP THEN
EZLDWOAO  002  EXEC(CMD('EZLE1IXL 'DOMID' 'ORIGOP' 'MESSAGE) ROUTE(ALL
EZLDWOAO  002  PPT)) DISPLAY(Y) NETLOG(Y) SYSLOG(N);
EZLDWOAO  002  IF MSGID = 'DWO044I' & DOMID = NTV6D & TEXT = . 'LISTING


Command ===>
F1=Help                  F2=Main Menu       F3=Return          F4=Enable
F5=Display Group/Block   F6=Roll            F7=Backward        F8=Forward
F9=Disable Stmt/Member   F10=Disable Block  F11=Disable Group  F12=Results
```

Figure 81. Automation-Table Management ENABLE/DISABLE Panel

Note: When you enter this panel from the GLOBAL DISPLAY OPTIONS or DISPLAY OPTIONS (6) pop-up windows, only not allowed statements are displayed.

In Figure 81, the following data is displayed:

MEMBER The name of the automation table that contains the statements that follow.

STATEMENT The automation-table statements that were retrieved from the automation-table listing member.

The status of each statement is displayed in different colors as follows:

Green The statement is enabled.

Red The statement is disabled.

Blue The statement cannot be individually disabled because it does not contain a label or sequence number.

Pink This statement is not disabled on its own, but as the result of a disabled table, %INCLUDE, or begin block.

To activate a pink statement, you must place the cursor on the preceding red statement, which caused this statement to be disabled, and activate that statement accordingly.

To use this panel to enable or disable an automation-table statement, scroll to the statement you want to take action on and choose one of the following function keys:

F4 Enables the label, block, or group, depending on which statement is the current focus and how that statement was disabled.

F5 Displays groups and blocks.

F9 Disables a statement or member; statements displayed in green might be disabled.

F10 Disables a block. This function requires that your selection is a LABEL statement that has a corresponding ENDLABEL statement.

F11 Disables a group. This function requires that your selection is a statement that contains a group label.

F12 Displays a description of the most recent commands issued and their respective results.

You can search the ENABLE/DISABLE panel for a particular statement, group, block, or label. To search, use the following commands. An abbreviated version of the command is in parentheses.

FIND *anytext* **(F** *anytext***)** Searches for the text you specify.

NEXT TAG (NT) Searches for the next group, block, or label.

NEXT IDENTIFIER (NI) Searches for the next group, block, label, or sequence number.

NEXT GROUP (NG) Searches for the next group.

NEXT BLOCK (NB) Searches for the next block.

NEXT SEQUENCE (NS) Searches for the next sequence number.

NEXT ENABLED (NE) Searches for the next enabled statement.

NEXT DISABLED (ND) Searches for the next disabled statement.

The search begins at the position of your cursor. The cursor is placed on the line where the search target was found. If another search is specified prior to paging forward or backward, the search begins after the previous search target.

Note: A member name of ******** denotes the start or end of an included member.

Displaying the Labels/Blocks/Groups Panel: Figure 82 illustrates the pop-up window that is displayed when you choose **F5** in Figure 81 on page 256. In the following panel, you can place your cursor on a Label/Block/Group and enable or disable it directly.

```
EZLKATGB              AUTOMATION TABLE MANAGEMENT

MEMBER      TYPE    LABEL/BLOCK/GROUP NAME    STATUS    NUMBER OF STATEMENTS
------      ----    ----------------------    ------    --------------------
DISTABLE    LABEL   BOB                        ENABLED            1
DISTABLE    LABEL   BOB2                       ENABLED            1
DISTABLE    LABEL   JIM                        ENABLED            1
DISTABLE    LABEL   NITE2                      ENABLED            1
DISTABLE    LABEL   STEVE                      ENABLED            1

DISTABLE    BLOCK   NITE                       ENABLED            8

DISTABLE    GROUP   KAT                        DISABLED           3



COMMAND ===>
F1=Help             F2=Main Menu         F3=Return            F4=ENABLE
F5=DISABLE          F6=Roll              F7=Backward          F8=Forward
F9=Toggle Display                                             F12=Cancel
```

Figure 82. Automation-Table Management Label/Block/Group Panel

If you enter the previous pop-up window from Figure 78 on page 253, the display panel, your actions affect only a single automation table. If you enter this pop-up window from Figure 80 on page 255, the global display panel, your action affects similarly named labels in all automation tables.

In Figure 82 on page 257, the following information is displayed:

MEMBER Indicates the name of the automation-table member that contains the label, block, or group.

TYPE Indicates whether the type is a label, block, or group.

LABEL/BLOCK/GROUP NAME
Indicates the name of the label, block, or group.

STATUS Contains one of the following statuses:

- ENABLED (The label, block, or group is enabled.)
- DISABLED (The label, block, or group is disabled.)

NUMBER OF STATEMENTS Indicates the number of statements within a label, block, or group.

Labels are always **1**. The number varies for groups and blocks.

Note: The NUMBER OF STATEMENTS column does not include all affected statements in the case of labels or groups. The total shown is the number of statements containing that label or group. For example, if a labeled statement ends with a BEGIN, those statements within the BEGIN and END block are not counted. To view all affected automation-table statements, use the ENABLE/DISABLE panel shown in Figure 81 on page 256.

You can use the function keys in the LABEL/BLOCK/GROUP pop-up window to enable the following actions:

F4 Enables the selected label, block, or group.

F5 Disables the selected label, block, or group.

F9 Toggles the display area so that only labels, blocks, or groups are displayed at one time.

F12 Displays the commands issued by your actions and the results of the commands.

Note: If you accessed this pop-up window from Figure 80 on page 255, the functions of F4 and F5 become global enable and disable functions that affect all automation tables. For example, if group NITEOPS is defined in two tables, pressing F5 disables that group in both tables.

The Confirmation Panel

Many of the functions provided by AUTOMAN require an automation-table listing file for the table being acted upon. If the listing file is not available following a

request for action on a table, a warning message is displayed. You can press **F4** on the warning panel, to authorize AUTOMAN to create the list files that are necessary to complete your request.

When you create the listing file, the automation tables involved must be reloaded. Reloading the tables resets the following conditions:

- The operator ID for the individual who loaded the current table, date, and time
- AUTOCNT statistics
- Interval and threshold condition-item counts
- Any statements currently disabled within the table

If any of the previously listed conditions has an adverse effect on your environment, press **F12** to cancel the listing request.

Chapter 16. Policy Services Overview

NetView Policy Services is a set of functions that enable dynamic policy-based management and automation of your resources. Several NetView functions exploit the Policy Services. Before an action is taken against a resource, these functions use the policy definitions to determine what action, if any, is to be taken.

You can write your own policy-based applications using NetView Policy Services. This section provides you with information to write your own policy applications and manage NetView policy.

NetView Policy Services consist of:

Policy Repository
A persistent data store in storage for your policy

Policy APIs
A set of functions that enable access to the policy repository for querying, modifying, adding, deleting, or loading the policy definitions.

Any application using NetView Policy Services needs to provide policy definitions to be loaded into the Policy Repository as well as application code to interpret the policy and take appropriate action. NetView is shipped with two applications that use the NetView Policy Services:

Network Management Console (NMC)
Provides you with function to define time schedules (for resources in NMC views), based on NMCSTATUS policy definitions. With these schedules, policy can be applied to views to specify when:

- The displayable status of one or more resources in a view is disabled at the NMC console
- One or more resources in a view is suspended from aggregation

Automated Operations Network (AON)
Provides you with automation of your network resources based on policy definitions.

Each of these applications defines default policy in the Policy Repository and then interprets the policy in order to take appropriate action based on that policy. You can modify the default policy statements shipped by NMC and AON. All of the policy definitions used by NMC and AON are documented in *IBM Tivoli NetView for z/OS Administration Reference*. This chapter provides information about installing Policy Services, the syntax of the policy file statements, how to load the policy files into the Policy Repository, and how to manage the policy.

Using Policy Services

To use NetView policy services:

- Customize DSITBL01
- Define your Policy
- Define your Policy Files

Customizing DSITBL01 (optional)

To customize DSITBL01, perform these steps:

1. Modify the statements associated with message EZL110I.

   NetView ships with statements (for AON and NMC) that resynchronize those components whenever an EZL110I message is received.
2. Optionally, add statements as appropriate for any other policy-based application.

Note: The DSITBL01 sample automation table contains many statements that are vital to the correct operation of the NetView program. Use great care when deciding to modify or remove all or part of the DSITBL01 sample.

Defining Your Policy Files

The CNMSTYLE member contains statements that enable you to define your policy definition files. Those statements are used every time you want to load or reload the Policy Repository. Information about the CNMSTYLE member can be found in the *IBM Tivoli NetView for z/OS Installation: Getting Started*.

You can define one or more individual policy files. They are merged and loaded as one logical file name. The default logical file name is NVPOLICY. Those statements are:

POLICY.&domain = NVPOLICY
: This defines a logical file name to be used when loading the Policy Repository.

POLICY.xxx = file_name
: This defines a real file name within DSIPARM that contains policy definitions. You can have one or more of these statements, depending on your needs. The "xxx" can be any set of characters as defined by each policy application. NetView ships with 2 policy file names:

```
POLICY.AON = EZLCFG01 for AON
POLICY.GRAPHICS = DUIPOLCY for NMC status
```

Required NetView Tasks

During initialization NetView loads the Policy Repository with the defined policy files based on the POLICY statements in the CNMSTYLE member. The Policy Repository remains active and loaded in storage while the EZLTCFG task remains active.

EZLTCFG must be initialized with NetView every time (INIT=YES). When you installed NetView there was a step for you to customize active tasks. Ensure EZLTCFG is in that list.

Policy File Syntax

The Policy Repository is open and flexible so applications using it are able to do so with minimal coding effort.

The policy files must reside in DSIPARM and must be referenced in the POLICY statements in the CNMSTYLE member..

You can also use %INCLUDE within your policy files to embed other members as part of your policy files.

There are only a few syntax rules. The basic syntax within any policy construct must adhere to this convention:

```
Policy_Name  Policy_definition, keyword1=value1,
                                keyword2=value2,
                                keyword3=value3
```

Where:

Policy_Name
The name of your policy (such as RECOVERY or NMCSTATUS) This is used for the ENTRY= parm on POLICY requests. This name must start in column 1 and must contain from 1 to 32 characters without embedded blanks ( ), commas (,,,), single (' ') or double quotation marks (" "), parentheses (), or equal signs (=).

Note: Before creating your own policy refer to Automated Operations Network (AON) Definitions in *IBM Tivoli NetView for z/OS Administration Reference* for a list of policy names to avoid.

Policy_Definition
The policy you want to define (for example: HOLIDAY) . This is used for the TYPE= parm on POLICY requests. This name must have from 1 to 32 characters without embedded blanks ( ), commas (,,,), single or double quotation marks (' ') (" "), parentheses ( ), or equal sign (=).

keywordn
The *keyword=value* pairs are required for the policy definition. Keywords are 1 or more characters without embedded blanks ( ), commas (,,,), single or double quotation marks (' ') (" "), or equal signs (=)

Valuen Keyword value syntax is defined by each application. Do not use values that contain a blank or the equal sign (=) unless it is contained within single or double quotation marks (' ') (" "). For example:

```
TEXT=A=B   is invalid
TEXT='A=B' is ok
TEXT="A=B" is ok
TEXT=A B   is invalid
TEXT="A B" is ok
```

Usage Notes

- Comments start in column 1 with an asterisk (*).

  Do *not* embed comments within a policy definition. Place your comments before the start of a given policy definition.
- Policy definitions, such as RECOVERY or NMCSTATUS, start in column 1.

  Note: Policy statements *must* begin in column 1. Continuation lines for these statements must *not* begin in column 1.
- Continuation to the next line is supported.

  To continue a line, end the current line with a comma and then start the next line in column 2 or greater.
- Continuation is *not* supported for text strings or for parenthesis-delineated strings.
- Policy statements must be in columns 1–72.
- If a policy statement contains a syntax error, the load of the policy repository terminates.
- Do not use a Tivoli NetView predefined policy for your own applications.

Changing Tivoli's NetView policy might cause errors. For additional information refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Examples:

Policy information is defined by the application and stored in the Policy Repository. The following examples of AON's RECOVERY policy illustrate how you can define policy. The comments are used to explain the purpose of the policy.

```
* AON RECOVERY Policy for all Physical Units (PUs) :
*   automate recovery except from midnight to 6am every day of the week
* Policy_name=RECOVERY
* Policy_definition=PU
* keyword1=AUTO
* value1=Y
* keyword2=NOAUTO
* value1=(*,00:00,06:00)
 RECOVERY PU,AUTO=Y,NOAUTO=(*,00:00,06:00)
```

```
* AON RECOVERY Policy for all "Holidays" :
*   automate recovery except from midnight to 6am every Holiday
* Policy_name=RECOVERY
* Policy_definition=HOLIDAY
* keyword1=AUTO
* value1=Y
* keyword2=NOAUTO
* value1=(*,00:00,06:00)
RECOVERY HOLIDAY,AUTO=Y,NOAUTO=(*,00:00,06:00)
```

```
* AON RECOVERY Policy for a resource called NCP10 :
*   automate recovery except from midnight to 2am every Holiday
*   automate recovery except from midnight to 4am on weekends
*   automate recovery except from midnight to 6am on weekdays
* This example shows continuation for lines 2 through 4.
RECOVERY NCP10,AUTO=Y
  NOAUTO=(HOLIDAY,00:00,02:00),
  NOAUTO=(WEEKEND,00:00,04:00),
  NOAUTO=(WEEKDAY,00:00,06:00)
```

Remember that the keyword value syntax is defined by the application. In this case, the application is AON.

For additional information on the AON RECOVERY policy as well as other NetView policy please refer to the *Tivoli NetView for OS390 Administration Reference.*

Policy File Management

You can use the NetView POLICY command to manage which policy files are loaded into the Policy Repository and to perform actions on those policy definitions.

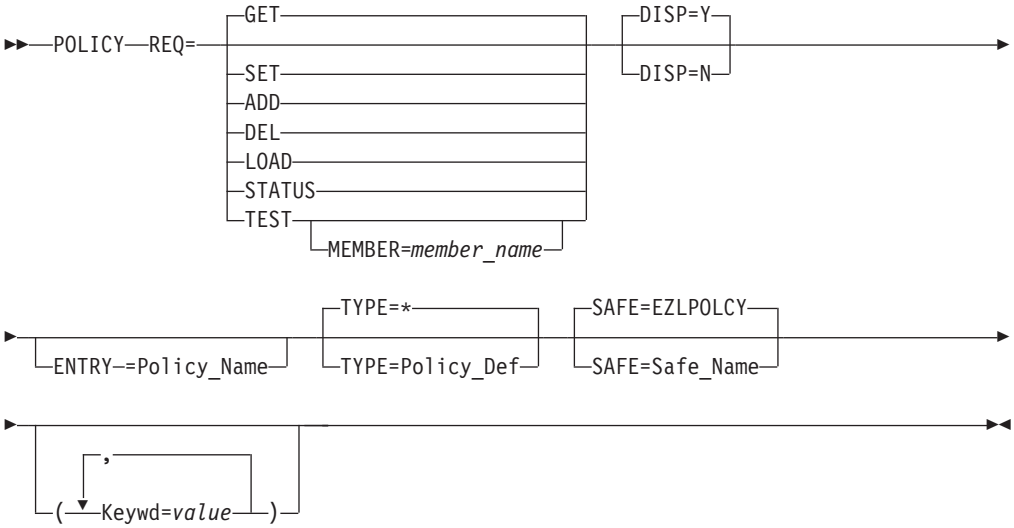
The POLICY command is a multi-purpose, generic application programming interface (API) into the Policy Repository. This command provides standardized access to all policy definitions in the Policy Repository.

Some applications ship more specific interfaces. For example, you can use the SETAUTO command to manage just the AON RECOVERY policy. When using SETAUTO, you do not see other policy definitions even if they are loaded. Application-specific interfaces are documented in the appropriate user's guide.

Using the Policy API

POLICY Syntax

POLICY



Where:

GET Retrieves the requested policy definition from the Policy Repository. This is the default.

SET Updates the requested policy definition keyword with a value.

ADD Creates a new policy definition in the Policy Repository with provided keywords and values.

DEL Deletes a policy definition from the Policy Repository.

LOAD
Loads the Policy Repository based on definitions in the CNMSTYLE member.

STATUS
Queries which policy files have been loaded in the Policy Repository.

TEST Performs a syntax check of the policy files.

MEMBER=*member_name*
The name of the actual policy file to test. If not specified on a TEST request, then all of the currently active policy files are tested.

DISP

Y: Displays pertinent messages at the user console. **Y** is the default.

N: Option that does *not* display pertinent messages at the user console.

Note: This option must be used by 3270 applications to avoid being interrupted by messages.

ENTRY=*Policy_Name*
Any valid policy name, such as RECOVERY or NMCSTATUS, as defined in the *Tivoli NetView for OS390 Administration Reference* or by other applications.

Type=*
If you enter **type=***, the POLICY command returns all policy definitions for a given Policy_Name. Type=* is the default.

TYPE=*Policy_Def*
Any valid policy definition, such as HOLIDAY, as defined in the *Tivoli NetView for OS390 Administration Reference* or by other applications.

SAFE=*Safe_Name*
The name of a safe containing the output from the request. EZLPOLCY is the default.

keyword
Any valid policy keyword allowed by the policy application, such as NOAUTO.

Value Any valid keyword value allowed by the policy application.

Return Codes:

-1 SIGNAL FAILURE

-5 SIGNAL HALT

0 Request was successful

1 Requested policy definition not found (GET/ADD/SET)

3 Missing Parameters--look for message EZL203I

4 Invalid Parameters--look for message EZL204I

7 SIGNAL NOVALUE--look for message EZL271E

8 SIGNAL SYNTAX--look for message EZL275E

9 Security Authorization Failure--look for message EZL228E

10 Request not processed--other error encountered

Usage Notes

- You can have one or more *keyword=value* pairs.
- You can specify TYPE=* to retrieve all policy definitions for a given policy grouping.

  For example, if you type **POLICY REQ=GET ENTRY=RECOVERY TYPE=***, the system returns all RECOVERY policy definitions from all polices.
- You cannot delete (REQ=DEL) keywords or keyword values, only specific policy definitions.
- You cannot query (REQ=GET) keywords or keyword values, only specific policy definitions.
- No parameters are allowed with REQ=STATUS or REQ=LOAD
- If you enter POLICY from a command line, DISP= is ignored.
- If MEMBER= is not specified for a REQ=TEST then all currently active policy files are syntax-tested, based on the current policy loaded in the Policy Repository.

Determining Which Policy Files are Loaded

To determine if the Policy Repository is loaded (and if it is loaded, with which files), issue this command:

POLICY REQ=STATUS

The response must look like:

```
EZL005I MEMBER NVPOLICY CURRENTLY BEING USED FOR THE CONTROL FILE
EZL006I NVPOLICY FILE 1 = EZLCFG01
EZL006I NVPOLICY FILE 2 = DUIPOLCY
EZL002I END
```

Where NVPOLICY is the logical file name used to load the Policy Repository. EZLCFG01 and DUIPOLCY are the real files used to create NVPOLICY when the policy definitions were loaded.

Syntax Testing the Policy Files

Before loading a policy file, you must perform a syntax test on it. To perform a syntax test, issue these commands:

POLICY REQ=TEST,MEMBER=member_name *member_name* is the name of your policy file in DSIPARM. The file can be an existing file that you just changed or it can be a new file that you want to load.

Perhaps you made changes to several policies within your policy files. You can test new versions of the currently loaded policy files by issuing a POLICY REQ=TEST without the MEMBER= parameter. For every file that is tested successfully, you see:

```
EZL023I TEST OF CONTROL FILE MEMBER "EZLCFG01" WAS SUCCESSFUL
```

For every file that is not tested successfully, you see this message, along with other more descriptive messages that document errors:

```
EZL023I TEST OF CONTROL FILE MEMBER "MYPOLICY" WAS UNSUCCESSFUL
```

When all files test accurately, you can reload the Policy Repository. See “Loading Policy Files.”

Loading Policy Files

The Policy Repository is loaded during NetView initialization based on the POLICY statements in the CNMSTYLE member. If you want to load the Policy Repository when NetView is active, issue a POLICY REQ=LOAD command. You must see:

```
EZL110I NVPOLICY BEING USED FOR THE CONFIGURATION TABLE
EZL006I NVPOLICY FILE 2 = EZLCFG01
EZL006I NVPOLICY FILE 2 = DUIPOLICY
EZL002I END
```

Reloading the policy removes temporary changes that were made to the previous policy that was loaded.

1. Use caution when loading and reloading the Policy Repository.

   Test your policy files before you load them. See “Syntax Testing the Policy Files.”

   Applications that use the Policy Repository must resynchronize whenever the policy is reloaded.

2. Authorize the tasks that load the policy.

   Applications that use the Policy Repository must resynchronize whenever the policy is reloaded.
3. If you have written your own application that uses the Policy Repository, do these steps:
   a. Review the EZLI110I automation statements that are used to resynchronize the application when the policy is reloaded.
   b. Provide similar functions for your application.

   DSITBL01 provides automation for message EZL110I.

To determine if your policy loaded successfully, you can use a sample clist (EZLECKPF) that is provided by NetView.

For example, add these lines to your EZL110I process:

```
'PIPE SAFE * | KEEP EZL110I'      /* save AIFR */
'EZLECKPF ' component              /*ie; AON    */
IF RC <> 0 THEN
    /* HANDLE POLICY NOT LOADED                */
ELSE
    /* POLICY LOADED, CONTINUE                 */
```

Querying a Policy Definition

To query a policy definition, issue this command:

POLICY REQ=GET ENTRY=Policy_Name TYPE=Policy_Definition

If the policy exists, you receive a multi-line response that includes the *keyword=value* pairs. For example, to query RECOVERY policy for NCP10, issue this command:

POLICY REQ=GET ENTRY=RECOVERY TYPE=NCP10

This is an example of what is returned:

```
EZL115I  RECOVERY NCP10 AUTO Y
EZL115I  RECOVERY NCP10 NOAUTO (HOLIDAY,00:00,02:00)
EZL115I  RECOVERY NCP10 NOAUTO (WEEKEND,00:00,04:00)
EZL115I  RECOVERY NCP10 NOAUTO (WEEKDAY,00:00,06:00)
EZL002I END
```

The Policy_Name is RECOVERY. The Policy_Definition is NCP10. There are four *keyword=value*e pairs. The first *keyword=value* pair is AUTO=Y. The second *keyword=value* pair is NOAUTO=(HOLIDAY,00:00,02:00). The third *keyword=value* pair is NOAUTO=(WEEKEND,00:00,04:00). The fourth *keyword=value* pair is NOAUTO=(WEEKDAY,00:00,06:00).

Querying a Group of Policy Definitions

To query multiple policy definitions with one command, issue this command:

"POLICY REQ=GET ENTRY=TCP390 TYPE=*".

You see:

```
EZL115I  TCP390 DEFAULTS PINGCNT 3
EZL115I  TCP390 DEFAULTS PINGRETRY 3
EZL115I  TCP390 DEFAULTS PINGLEN 64
EZL115I  TCP390 DEFAULTS PINGTIME 10
```

```
EZL115I  TCP390 DEFAULTS DEBUG 0
EZL115I  TCP390 DEFAULTS VERBOSE Y
EZL115I  TCP390 DEFAULTS SNMPRETRY 2
EZL115I  TCP390 DEFAULTS SNMPTO 3
EZL115I  TCP390 DEFAULTS MAXREP 10
EZL115I  TCP390 DEFAULTS NONREP 0
EZL115I  TCP390 DEFAULTS RPLENGTH 64
EZL115I  TCP390 DEFAULTS RPTO 5
EZL115I  TCP390 NMPIPL10 IPADDR 9.67.50.52
EZL115I  TCP390 NMPIPL10 HIER2 SP-APPL
EZL115I  TCP390 NMPIPL10 HIER3 NETSP
EZL115I  TCP390 NMPIPL10 DOMAIN LOCAL
EZL115I  TCP390 NMPIPL10 UNIXSERV YES
EZL115I  TCP390 NMPIPL10 TCPNAME TCP38
EZL115I  TCP390 NMPIPL10 FORMAT STACK
EZL115I  TCP390 NMPIPL10 SNMP MVS
EZL115I  TCP390 NMPIPL10 HOSTNAME NMPIPL10 raleigh.ibm.com
EZL115I  TCP390 NMP190 IPADDR 9.67.50.34
EZL115I  TCP390 NMP190 HIER2 SP-APPL
EZL115I  TCP390 NMP190 HIER3 NETSP
EZL115I  TCP390 NMP190 DOMAIN NTV74
EZL115I  TCP390 NMP190 UNIXSERV YES
EZL115I  TCP390 NMP190 TCPNAME TCP38
EZL115I  TCP390 NMP190 FORMAT STACK
EZL115I  TCP390 NMP190 SNMP MVS
EZL115I  TCP390 NMP190 HOSTNAME NMP190.raleigh.ibm.com
EZL002I END
```

In this case, the Policy_Name is TCP390. The query returned 3 Policy_Definition values: DEFAULTS, NMPIPL10, and NMP190.

Wild cards are supported. For example, issue **POLICY REQ=GET ENTRY=TCP390 TYPE=NMP*** to retrieve TCP390 policy definitions for only NMP*. You see a response similar to this:

```
EZL115I  TCP390 NMPIPL10 IPADDR 9.67.50.52
EZL115I  TCP390 NMPIPL10 HIER2 SP-APPL
EZL115I  TCP390 NMPIPL10 HIER3 NETSP
EZL115I  TCP390 NMPIPL10 DOMAIN LOCAL
EZL115I  TCP390 NMPIPL10 UNIXSERV YES
EZL115I  TCP390 NMPIPL10 TCPNAME TCP38
EZL115I  TCP390 NMPIPL10 FORMAT STACK
EZL115I  TCP390 NMPIPL10 SNMP MVS
EZL115I  TCP390 NMPIPL10 HOSTNAME NMPIPL10.raleigh.ibm.com
EZL115I  TCP390 NMP190 IPADDR 9.67.50.34
EZL115I  TCP390 NMP190 HIER2 SP-APPL
EZL115I  TCP390 NMP190 HIER3 NETSP
EZL115I  TCP390 NMP190 DOMAIN NTV74
EZL115I  TCP390 NMP190 UNIXSERV YES
EZL115I  TCP390 NMP190 TCPNAME TCP38
EZL115I  TCP390 NMP190 FORMAT STACK
EZL115I  TCP390 NMP190 SNMP MVS
EZL115I  TCP390 NMP190 HOSTNAME NMP190.raleigh.ibm.com
EZL002I END
```

In this case, the query returned two policy_definitions, NMPIPL10 and NMP190.

Modifying a Policy Definition

To modify a policy definition change the value of one or more keywords. Modifying a policy definition changes the keywords and values you specify and leaves all other keywords and values unchanged. To change a policy definition, issue this command:

POLICY REQ=SET ENTRY=Policy_Name TYPE=Policy_Definition *keyword=value*

To change the RECOVERY policy for NCP10 to set AUTO to N, issue this command:

POLICY REQ=SET ENTRY=RECOVERY TYPE=NCP10 AUTO=N

You see a response similar to this:

```
EZL001I REQUEST "REPL" WAS SUCCESSFUL FOR EZLEPOLY
```

To verify your changes, issue this command:

POLICY REQ=GET ENTRY=RECOVERY TYPE=NCP10

You see a response similar to this:

```
EZL115I  RECOVERY NCP10 AUTO N
```

Updates are made to the copy of the policy definition that is loaded into the Policy Repository. The original policy file in DSIPARM remains unchanged. If you want the change to be permanent, modify the original policy file in DSIPARM so that the change is not lost the next time the policy is loaded

Note: If you want to replace an existing policy definition in its entirety, delete the current policy definition and then add the new policy definition.

Deleting a Policy Definition

To delete a policy request, issue this command:

POLICY REQ=DEL ENTRY=Policy_Name TYPE=Policy_Def

To delete the RECOVERY policy for NCP10, issue this command:

POLICY REQ=DEL ENTRY=RECOVERY TYPE=NCP10

If the command works, you see a response similar to this:

```
EZL001I REQUEST "DEL " WAS SUCCESSFUL
```

Deletions are made from the policy file that is in the Policy Repository. The original policy file in DSIPARM remains unchanged. If you want the change to be permanent, modify the original policy file in DSIPARM. Then the change is not lost the next time the policy is loaded.

Adding a Policy Definition

To dynamically create or add a new policy definition issue this command:

POLICY REQ=ADD ENTRY=Policy_Name TYPE=Policy_Definition
keyword=*value1 keyword2*=*value2* **...**

To define RECOVERY policy for MYRES with AUTO set to yes and a NOAUTO window daily from midnight to 6 AM, issue this command:

POLICY REQ=ADD ENTRY=RECOVERY TYPE=MYRES AUTO=Y NOAUTO=(*,00:00,06:00).

The policy_name is RECOVERY. The policy_definition is MYRES. The first *keyword=value* pair is AUTO=Y. The second *keyword1=value1* pair is NOAUTO=(*,00:00,06:00).

This message is returned:

```
EZL001I REQUEST "ADD " WAS SUCCESSFUL
```

You can now query the policy to see what was loaded into the Policy Repository. Updates are made to the copy of the policy definition which is loaded into the Policy Repository. The original policy file in DSIPARM remains unchanged. If you want the change to be permanent, modify the original policy file in DSIPARM so that the change is not lost the next time the policy is loaded.

REXX API Usage

You can write complex routines using the POLICY API. For information refer to "Using the Policy API" on page 265.

The syntax of the POLICY API is unchanged when it is called from a REXX procedure. The output changes when you attempt to query (REQ=GET) a policy definition. The response does not contain a message id (such as EZL115I). All other data remains the same.

For example, to write a REXX routine to query the Tivoli NetView RECOVERY policy for NCP10, issue this command:

```
/* REXX */

'PIPE NETV |',

'POLICY REQ=GET ENTRY=RECOVERY TYPE=NCP10',

'| STEM POL.' \DO I=1 to POL.0

END

_

_

EXIT
```

Timer APIs

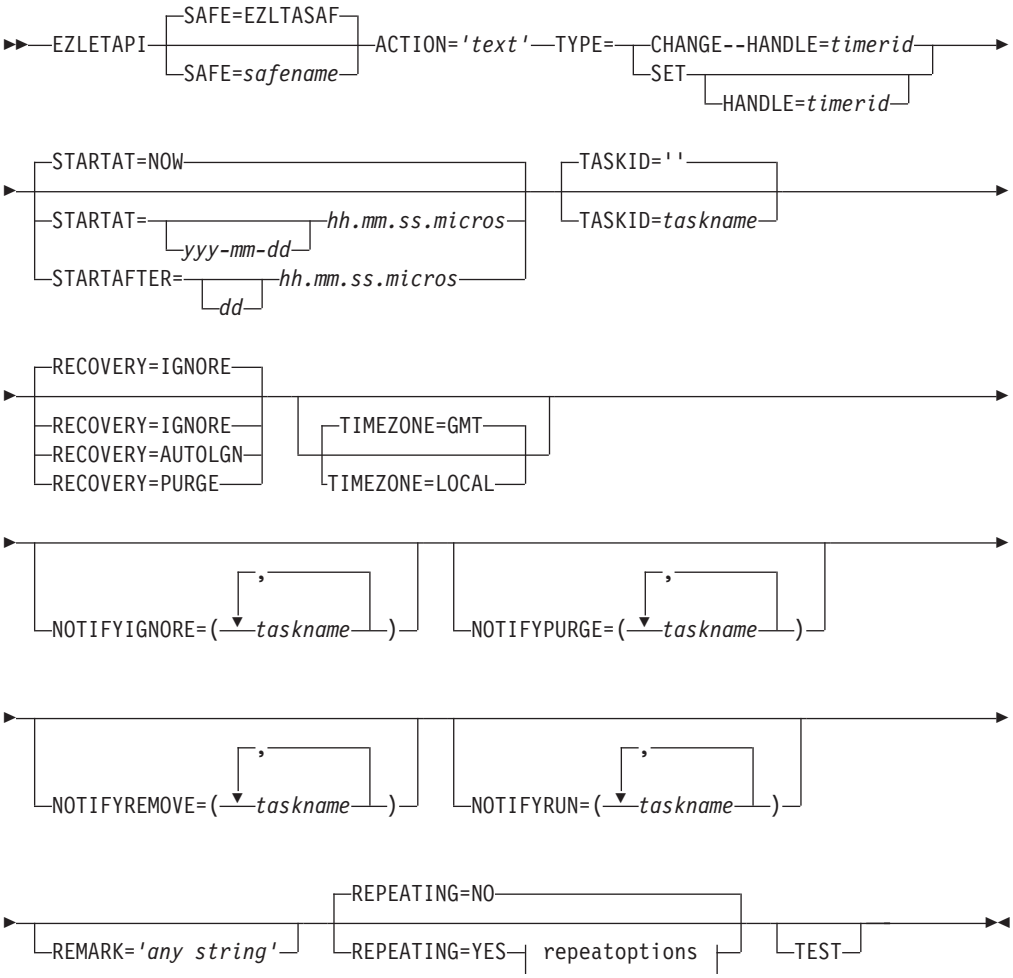
This section presents information that enables you to use NetView Timer APIs. Each section describes what the API does, the syntax for the API, and some examples on how to use the API. It also lists the return codes that you might receive when you issue the API.

EZLETAPI

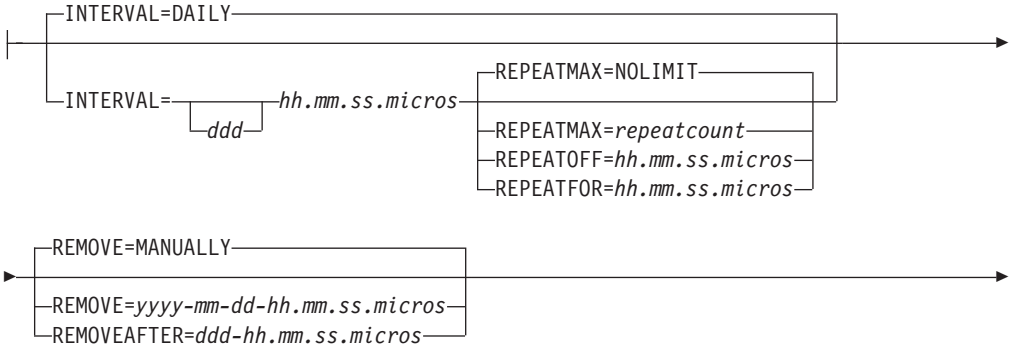
The EZLETAPI API is used to define parameters to establish or change a timer.

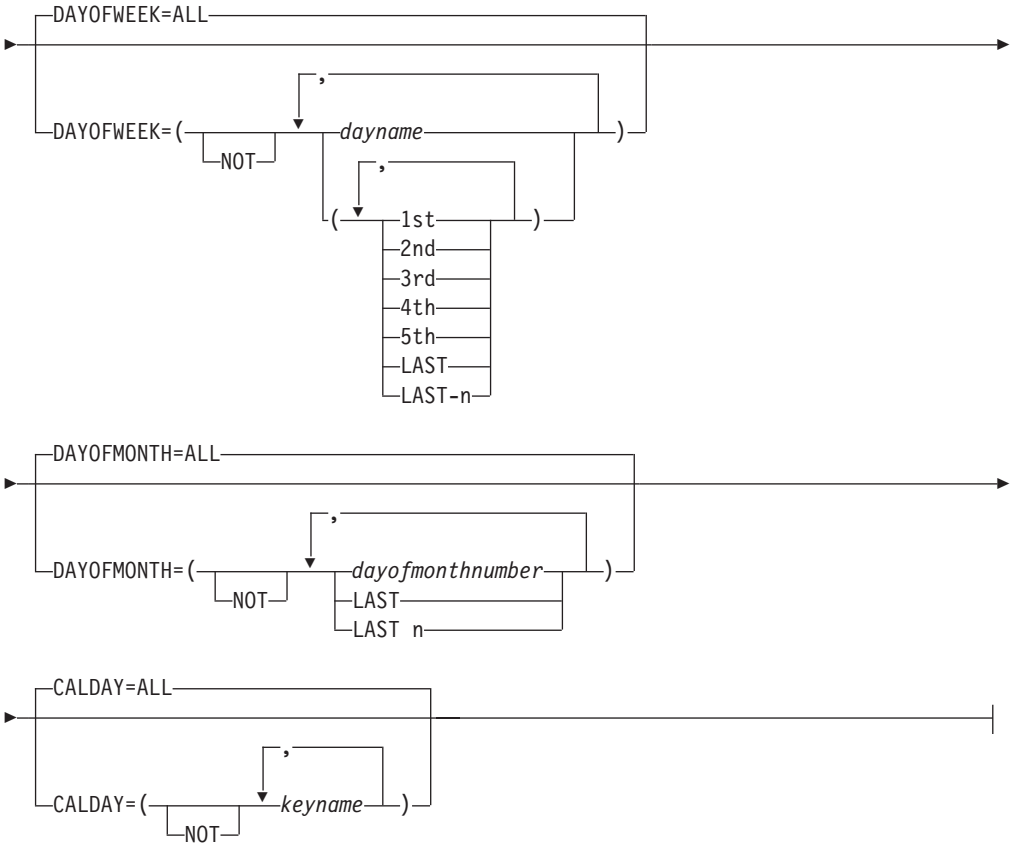
The syntax for the EZLETAPI API is:

EZLETAPI



repeatoptions:





Where:

SAFE The name of the safe where the output is placed. The default is EZLTASAF.

ACTION
The action to be taken when the timed event occurs. The string must be enclosed in single quotation marks or apostrophes. Neither the single quotation mark nor apostrophe can be contained within the string. If needed, use two apostrophes together or double quotation marks within the string.

TYPE Specifies the type of timer request:

Set Specifies that this is a new request. Passing a timer handle with this option creates a timer with that handle.

Change
Specifies that this is a change to an existing timer. The existing timer handle *must* be passed in order to delete the existing timer.

Note: If TYPE=CHANGE, all parameters that are *not* to be changed on an existing timer *must* also be specified on the invocation. These parameters can be obtained by using EZLEQAPI and changing the preferred parameter. If all parameters are not specified, the timer is set using the parameters provided and defaults for parameters not specified.

HANDLE
A unique identifier (1-8 characters) for the timer. If a handle is not specified, a unique identifier is generated. Handles cannot begin with ALL, RST, or SYS.

STARTAT
Specifies the date and time that the action starts. If not specified, the default is to start immediately. The STARTAT and STARTAFTER keywords are mutually exclusive.

STARTAFTER
Specifies an interval after which the action starts. The STARTAT and STARTAFTER keywords are mutually exclusive. If not specified, the default is to start immediately.

TASKID
The name of a task or group on which the timer is to be scheduled. The default is to schedule the timer on the task issuing EZLETAPI.

RECOVERY
Specifies how to proceed when the ACTION is scheduled to run and the specified task is not active.

AUTOLGN
Specifies that an autotask is to be started with the specific task name. AUTOLGN cannot be specified with a group of tasks.

IGNORE
Specifies that the ACTION is not to run unless the task is active. This is the default.

PURGE
Specifies that the timer is to be removed if the task is not active.

TIMEZONE
Specifies whether a time is relative to Greenwich mean time (GMT) or local system time. The default is GMT.

NOTIFYIGNORE
Specifies that the operators listed are sent a notification when an action is not run because the specified task was not active.

NOTIFYPURGE
Specifies that the operators listed are sent a notification when an action is purged because a task is not active or the timer was deleted.

NOTIFYREMOVE
Specifies that the operators listed are sent a notification when an action is removed because the REMOVE time was reached or the command was scheduled to run without an interval time.

NOTIFYRUN
Specifies that the operators listed are sent a notification when an action is scheduled to run and the specified task is active.

REMARK
Enables a remark to be specified when the timer is set. For example, you can specify what command list set the timer. The remark must be enclosed in single quotation marks or apostrophes.

REPEATING
Specifies that the action is to be repeated. The default is NO. REPEATING=YES is required to use these keywords:

INTERVAL
Specifies the time the action is to be repeated between the STARTAT and REPEATOFF times. DAILY specifies that the action is to run once each day subject to the DAYOFWEEK, DAYOFMON, and CALDAY entries. 'ddd' can be in the range of 1-365.

REPEATMAX
Specifies the number of times the command is repeated and applies during each AT time each day. The interval timespec multiplied by the repeat count must be less than 24 hours. REPEATMAX=NOLIMIT causes the timer to be scheduled at regular intervals, starting from the STARTAT or STARTAFTER time. Each new timer is set to run exactly at INTERVAL amount of time from the previous calculated run time. On subsequent days, the AT or AFTER time is not a factor. NOLIMIT is the default.

REPEATOFF
Specifies the time of day the interval is to end. The value must be less than 24 hours and can run into the next day. The timer does not run at, or after, the REPEATOFF time.

REPEATFOR
Specifies the length of time the interval is to run. The timer does not run at, or after, the STARTAT time of day with the REPEATFOR time. The interval can be anything less than 24 hours, and can run into the next day.

REMOVE
Specifies when a timed action is to be deleted. The default is MANUALLY, which means the timer is not automatically removed.

REMOVEAFTER
Specifies when a timed action is to be deleted. The dd-hh.mm.ss.micros specifies when a timer is removed following a STARTAT or STARTAFTER time.

DAYOFWEEK
The name of the weekday. DAYOFWEEK affects and is affected by DAYOFMON and CALDAY. DAYOFWEEK=ALL is the default. Valid values are:

- SUN
- MON
- TUE
- WED
- THU
- FRI
- SAT
- WEEKDAY
- WEEKEND

Specifying NOT to omit selected days eliminates a longer list of days to be included. For example, instead of specifying DAYOFWEEK=(TUE,WED,THU,FRI), you can achieve the same result by specifying DAYOFWEEK=(NOT MON,WEEKEND) and the command runs only on Tuesdays through Fridays.

You can specify that a command is to run on certain occurrences of that day within the month. For example,

DAYOFWEEK=(MON(1ST,3RD),FRI(LAST)) causes the command to run only on the first and third Monday, and the last Friday of the month. Unsigning the LAST or LAST-n prevents having to consider the number of a specific weekday within that month. Valid values are:

- In the range 1ST - 5TH
- LAST
- In the range LAST -1 through LAST -4

DAYOFMONTH
The number of the day within the month. DAYOFMONTH=ALL is the default. Valid values are:

- In the range of 1 to 31
- ALL
- LAST
- In the range of LAST -1 through LAST -30
- NOT

DAYOFMONTH=ALL is the default. DAYOFMONTH affects and is affected by DAYOFWEEK and CALDAY.

Specifying NOT to omit selected days reduces a longer list of days to be included. For example, instead of specifying DAYOFMON=(2,4,5,6,7,8,...29,30) you can achieve the same result by specifying DAYOFMON=(NOT 1,3,31) and the command is not run on the first, third, and thirty-first day of the month. Specifying LAST or LAST -n eliminates having to consider the number of days within that month.

CALDAY
Name of a key as defined in DSISCHED. The command runs on the specified days. If NOT is specified, the command does not run on the specified days. You can enter up to 1,000 unique keys in the list. If you exceed this limit, message DSI656I is issued. CALDAY=ALL is the default.

TEST Optional keyword which enables you to verify that the command you are building is syntactically correct–without actually scheduling the CRON timer.

Return Codes:

Note: For a return code of **0**, no messages are returned. For non-zero return codes, messages that CHRON generates are returned in the safe.

-8 REXX syntax failure

-5 REXX halt

-1 REXX failure

0 Successful completion

4 TYPE not specified correctly

8 Incorrect safe name

12 Error in operands passed

16 Security check failed

20 Timer identifier not unique

24 An invalid value was found by the timer command.

28 The time specified has already passed.

32 The calendar was not available.

36 No response from the timer command.

40 The existing timer deletion failed

44 Internal processing error

OUTPUT:

Safe Containing: For non-zero return codes, any message generated.

For Return Code 0:
Timer handle beginning in column 1.

Messages

The following messages and their associated return codes are generated by EZLETAPI:

EZL228
RC 16

EZL984
RC 12

DSI450
RC 12

DSI486
RC 12

DSI649
RC 4, 8, 12

DSI651
RC 12

DSI654
RC 12

DSI655
RC 12

Other messages are returned that are generated by the invoked timer command.

Example

The following test exec uses EZLETAPI, EZLEQAPI, and EZLEDAPI:

```
 /*   */
/* Valid parameters to this test exec are TIMERID and OPID       */
/* This routine changes the ACTION keyword of an existing timer  */
/* and sets the timer using EZLETAPI.                            */
trace o
parse arg argstring
parse var argstring TMRID OPID .

/* Make OPID valid for the call*/
If OPID = '' Then
```

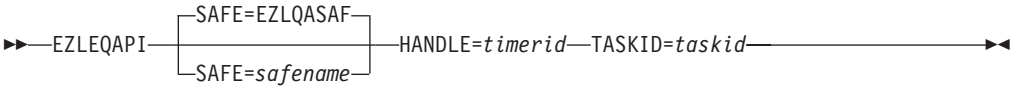
```
 OPID = ''''''
 delfailed = 0
/* Build the command to be invoked   */
testcmd = 'EZLEQAPI '
testcmd = testcmd || 'TASKID='OPID' '
testcmd = testcmd || 'HANDLE='TMRID' '
testcmd = testcmd || 'SAFE=SOMSAFE '

Address NETVASIS
/* Issue the command that was built  */
'PIPE CORRCMD (MOE) 'testcmd,
'| SEP',
'| STEM CHRON1.'

/* Check for messages returned from PIPE   */
If CHRON1.0 ¬= 0 Then
  Do I = 1 to CHRON1.0 until I = CHRON1.0
say CHRON1.I
end
Else                                    /* No PIPE errors         */
  Do
     /* Put the EZLEQAPI safe output into stem    */
    'PIPE SAFE SOMSAFE ',
    '| sep ',
    '| STEM SOUTPUT.'
   /* Line 1 contains the return code from EZLQAPI                */
   If soutput.1 = 0 Then               /* Check return code       */
     Do
       /* Line 2 contains the list of keywords that were set      */
       temp_var = soutput.2
       /* Prime the EZLETAPI command                              */
       newtimer = 'EZLETAPI SAFE=NEWSAFE '

       /* Line 3 is the first line of keywords.  Loop until the   */
       /* end of the stem to retrieve them all.                   */
       do i = 3 to soutput.0           /* Get the parms           */
         'PIPE var soutput.'i' | varload ' /* Save the keywords   */
         parse var temp_var var_name ',' temp_var
         Select
         /* NEXT_POP is not a valid keyword for EZLETAPI         */
         When var_name = 'NEXT_POP' Then
          nop

         /* Set the value of the ACTION keyword and append it to */
         /* to the command being built                           */
         When var_name = 'ACTION' Then
          newtimer = newtimer || 'ACTION=''DISCONID'''
         When var_name = 'HANDLE' Then
           Do
             If substr(HANDLE,1,3) = 'SYS' |,
                substr(HANDLE,1,3) = 'RST' Then
               Do
                 oldhandle = value(handle)
               End
             Else
               Do
                 oldhandle = ''
                 newtimer = newtimer || 'HANDLE='value(HANDLE)' '
               End
           End

         /* Append any other keyword found                       */
         Otherwise
          newtimer = newtimer || var_name ||'='||value(var_name)||' '
        End
      end                              /* Get the parms           */
    End
```

```
  Else                          /* non-zero EZLEQAPI return code  */
    Do
      do i = 1 to soutput.0
        say soutput.i
      end
    End
End

/* If there was an invalid timerid found earlier, delete that  */
/* timer before continuing.                                     */
If oldhandle ¬= '' Then
  Do
    deletecmd = 'EZLEDAPI safe=DELSAFE HANDLE='oldhandle' ' ||,
                'TASKID='value(taskid)
    newtimer = newtimer || 'TYPE=SET'
    'PIPE CORRCMD (MOE) 'deletecmd,
    '| SEP',
    '| STEM Chron1.',
    '| COUNT',
    '| VAR errorcnt',

    If Chron1.0 ¬= 0 Then
     Do
      'PIPE SAFE DELSAFE ',
      '| STEM DOUTPUT.'

    If Doutput.0 ¬= 0 Then
      If Doutput.1 ¬= '0' Then
        Do
          delfailed = 1
          say Doutput.I
        End
    End
  End
Else
 newtimer = newtimer || 'TYPE=CHANGE'
If delfailed = 0 Then
  Do
    /* Issue the EZLETAPI command just built   */
    'PIPE CORRCMD (MOE) 'newtimer,
    '| SEP',
    '| STEM Chron1.',
    '| COUNT',
    '| VAR errorcnt',
   If Chron1.0 ¬= 0 Then
     Do
       'PIPE SAFE NEWSAFE ',
       '| STEM NOUTPUT.'

       If Noutput.0 ¬= 0 Then
         If Noutput.1 = 0 Then
           Do
             say 'The handle is 'Noutput.2
           End
         Else
           say 'Failure in EZLETAPI: Return code is 'Noutput.1
       Else
         say 'Nothing in the safe'
     End
   End
 Else
   SAY 'The delete of timer 'oldhandle' was not successful'
exit
```

EZLEQAPI

EZLEQAPI is an API that enables you to easily query timers (that were set by the CHRON command) to determine if a particular timer has been set.

The syntax for the EZLEQAPI API is:



Where:

SAFE Specifies the name of the safe where the output from the EZLEQAPI command is placed. The default is EZLQASAF.

HANDLE
The timers to be queried. Valid values are:

timerid Displays the status of the named timer request. The *timerid* is the optional handle specified on the HANDLE operand of the SETTIMER command or generated by the system.

TASKID
The tasks to be queried. Valid values are:

taskid Lists only requests for the named operator and timer request. You can specify taskid even if the operator is not currently logged on.

Return Codes:

-8 REXX syntax failure

-5 REXX halt

-1 REXX failure

0 Successful completion

4 No timers found matching the criteria specified.

8 Incorrect safe name

12 Error in operands passed

16 Security check failed

20 Internal processing error

24 Requested timer was not a CHRON timer.

OUTPUT:

Safe Containing:

For non-zero return codes:
Error messages generated

Messages generated by this routine and associated return codes:

EZL228
(RC 16)

EZL253
(RC 4)

DSI450
(RC 12)

DSI486
(RC 12)

DSI649
(RC 8, 12)

DSI651
(RC 12)

Other messages are returned that are generated by the invoked timer command.

For Return Code 0

For return code zero requested timer information is returned in the form:

/* Line 1 contains the return code*/

/* Line 2 contains a comma delimited list of variables that are being returned in the safe*/

/* Line 3 to the end of the safe contains the values that can be set using the VARLOAD stage.*/

/HANDLE/*timerid*

/ACTION/*'text'*

/START/*value* | /STARTAFTER/*value*

/TASKID/*value*

/RECOVERY/*value*

/TIMEZONE/*value*

/NOTIFYIGNORE/*values*

/NOTIFYPURGE/*values*

/NOTIFYREMOVE/*values*

/NOTIFYRUN/*values*

/REMARK/*'text'*

/REPEATING/

/INTERVAL/*value*

/REPEATMAX/*value*

/REPEATOFF/*value*

/REPEATFOR/*value*

/REMOVE/*value*

/REMOVEAFTER/*value*

/DAYOFWEEK/*value*

/DAYOFMON/*value*

/CALDAY/*value*

/NEXTPOP/*yyyy-mm-dd-hh.mm.ss*

Note: NEXTPOP is the next time when this timer pops.

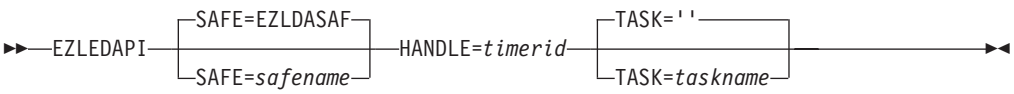
Example

For an example of a test exec using EZLEQAPI see page 277.

EZLEDAPI

EZLEDAPI

The EZLEDAPI API enables applications to delete timers they have established.



Where:

HANDLE
The timers to be deleted. Valid value is:*timerid* Deletes the specific timer request.

SAFE The name of the safe in which output from the EZLEDAPI command is placed. The default is EZLDASAF.

TASKID
The task on which the delete timer is to be performed. Valid values are:

'' Deletes timer requests for your own operator ID. If you do not specify TASKID, this is the default.

taskid Deletes only timer requests for the specified operator. You can specify taskid even if the operator is not currently logged on.

Return Codes:

-8 REXX syntax failure

-5 REXX halt

-1 REXX failure

0 Successful completion

4 No timers were deleted.

8 Incorrect safe name

12 Error in operands passed

16 Security check failed

20 Internal processing error

Safe Containing:

For Return Code 0:
Timer handle that was deleted, beginning in column 1.

For return codes above 8:
Error messages generated

Messages

The following messages and their associated return codes are generated by EZLEDAPI:

EZL228
RC 16

DSI450
RC 12

DSI486
RC 12

DSI649
RC 8, 12

DSI651
RC 12

Other messages are returned that are generated by the invoked timer command.

Note: This routine requires access to the NetView PURGE TIMER=*timerid* OP=*operid*.

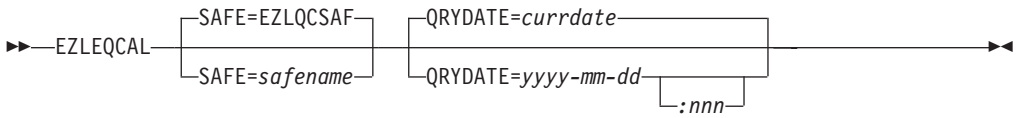
Example

For an example of a test exec using EZLEDAPI see page 277.

EZLEQCAL

The EZLEQCAL API enables you to query the calendar to determine what definitions are in place.

The syntax for the EZLEQCAL API is:



Where:

QRYDATE
Date to be queried. This can specify the number of days to be queried by specifying :nnn following the date specification. The range for *yyyy* is from 1 to 9999. The range for *nnn* is 1 to 999. The range of dates that can be queried (including the *nnn*) is 0001-01-01 through 9999-12-31.

SAFE Name of the safe in which output from the EZLEQCAL command is placed. The default is EZLQCSAF.

Return Codes:

-8 REXX syntax failure

-5 REXX halt

-1 REXX failure

0 Successful completion

4 Incorrect safe name

8 Error in operands passed

Safe Containing:

For Return Code 0:

A safe containing the data as described in this information. Data begins in column 1.

This example displays four days, beginning with the 31st of December, 2010.

```
EZLEQCAL 2010-12-31:4

DEC 31,LAST    2010 FRI 5TH,LAST   (NO SPECIAL CALENDAR DAYS)
JAN  1,LAST-30 2011 SAT 1ST,LAST-4 DAYA,HOLIDAY,NEW_YEARS_DAY
JAN  2,LAST-29 2011 SUN 1ST,LAST-4 DAYB
JAN  3,LAST-28 2011 MON 1ST,LAST-4 DAYC
```

The first column is the Month. The second column is the date specification. The third column is the year. The fourth column is the day of the week. The fifth column is the day specification. The last column is user-defined days.

Note: This routine requires READ access to the DSISCHED data set in DSIPARM.

Chapter 17. Installation Exits

This chapter provides product-sensitive programming interfaces and associated guidance information.

Installation exits available for automation are briefly described in this section.

What Are Installation Exits?

Some NetView exits enable programming access to data. Through these exits, user-written functions can obtain the text of operator commands, logons, messages, and MSUs. Different exits are driven based on the origin of the text and the stage of NetView processing. The key exits associated with automation are installation exits DSIEX02A, XITCI, DSIEX16, DSIEX16B, and DSIEX17.

For more information about the automation installation exits and about other installation exits, refer to *IBM Tivoli NetView for z/OS Programming: Assembler* and *IBM Tivoli NetView for z/OS Programming: PL/I and C*.

Installation Exit DSIEX02A

NetView calls exit DSIEX02A to process standard output to an operator's terminal. Because DSIEX02A processing occurs before the automation table is used, any changes you make to messages in exit DSIEX02A can affect message automation. Besides altering or replacing messages, you can also use exit DSIEX02A to delete messages. If a message has been deleted, the automation table is not used for that message. You can write exit DSIEX02A in assembler, PL/I, or C, but use assembler for performance reasons.

Installation Exit XITCI for BNJDSERV

The BNJDSERV task calls exit XITCI after receiving an MSU. You can modify, replace, or delete MSUs before they go to the automation table or to hardware monitor logs. An advantage of using BNJDSERV's XITCI exit instead of DSICRTR's is that BNJDSERV processes MSUs from ALERT-NETOP, in addition to MSUs from DSICRTR. You can write exit XITCI in assembler, PL/I, or C, but use assembler for performance reasons.

Installation Exits DSIEX16 and DSIEX16B

NetView calls exit DSIEX16 and exit DSIEX16B immediately after automation table processing occurs. You must write exit DSIEX16 and exit DSIEX16B in assembler. These exits are not called for automation table testing.

NetView calls exit DSIEX16 after a message has been processed by the automation table and before any commands issued from the automation table are run. With exit DSIEX16, you can modify the processing options for a message, reformat a message, or replace it. You can replace the processing options specified by the automation table, such as whether the message must be logged and displayed, and can prevent the NetView override settings from taking effect. You can specify new automation commands to be issued in response to the message. You can also use exit DSIEX16 to monitor the effectiveness of message suppression and automation.

NetView calls exit DSIEX16B after an MSU has been processed by the automation table and before any commands issued from the automation table are run. With exit DSIEX16B, you can:

- Examine or modify an MSU and its attributes
- Change the results of automation processing
- Monitor the effectiveness of your MSU automation

Depending on the functions you perform, you might be able to use the same routine for both exit DSIEX16 and exit DSIEX16B.

Installation Exit DSIEX17

Exit DSIEX17 is called after NetView converts MVS messages and delete operator messages (DOMs) into automation internal function request (AIFR) format. It enables you to modify or delete a message or a DOM. You can write exit DSIEX17 only in assembler.

Part 5. Single-System Automation

Chapter 18. Automation Setup Tasks

Before you can use automated operations, you must perform these setup tasks:

- If you want to use system automation, set up communication between the NetView system and the operating system.
- Define and activate autotasks to perform automation processing.

Establishing Communication between the NetView System and the Operating System

For system automation or for your operators to issue operating system commands from the NetView program, activate the interface between the operating system and the NetView system.

Preparing the z/OS System for System Automation

To prepare a z/OS operating system for system automation using NetView:

- Define NetView to the z/OS operating system as a subsystem.
- Choose the z/OS operating system message delivery option you want to use:
  - The subsystem interface
  - extended multiple console support (EMCS) consoles
- Forward system messages from the operating system to NetView (MPF table).
- Define subsystem allocatable consoles to the z/OS operating system.
- Review the NetView start-up procedures.
- Optionally, add CMDDEF statements for z/OS operating system commands to make it easier to issue z/OS operating system and subsystem commands from NetView (thereafter, it is not necessary to prefix your system commands with the z/OS operating system.

These steps prepare for the interaction of the z/OS operating system with both the NetView subsystem address space and the NetView application address space. The two NetView address spaces cooperate to provide automation capabilities on z/OS operating systems.

Figure 83 on page 292 shows message flow between the z/OS system and NetView when the subsystem interface is used. Figure 84 on page 293 shows the command flow.

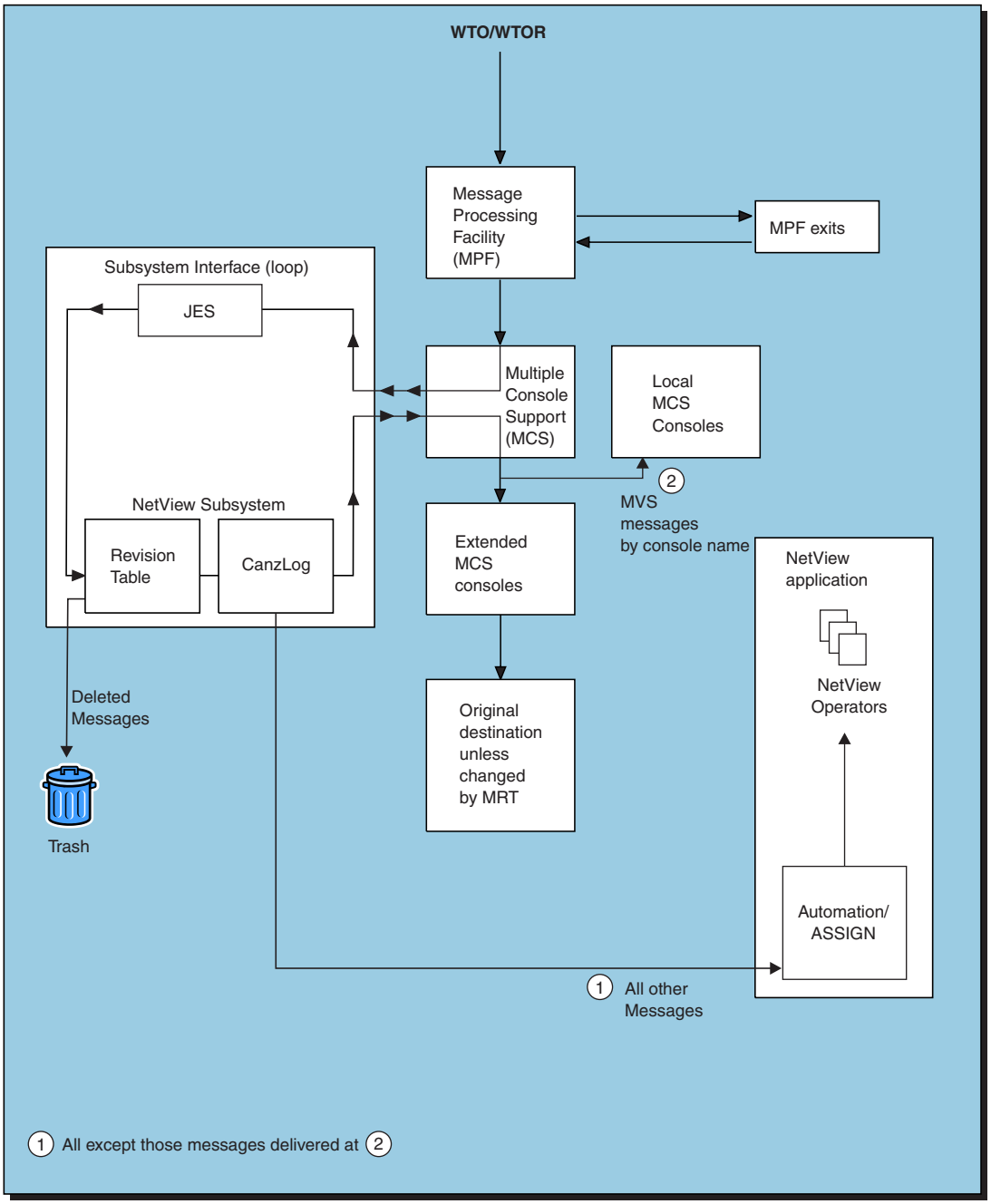


Figure 83. Message Flow between the z/OS System and the NetView Subsystem Interface

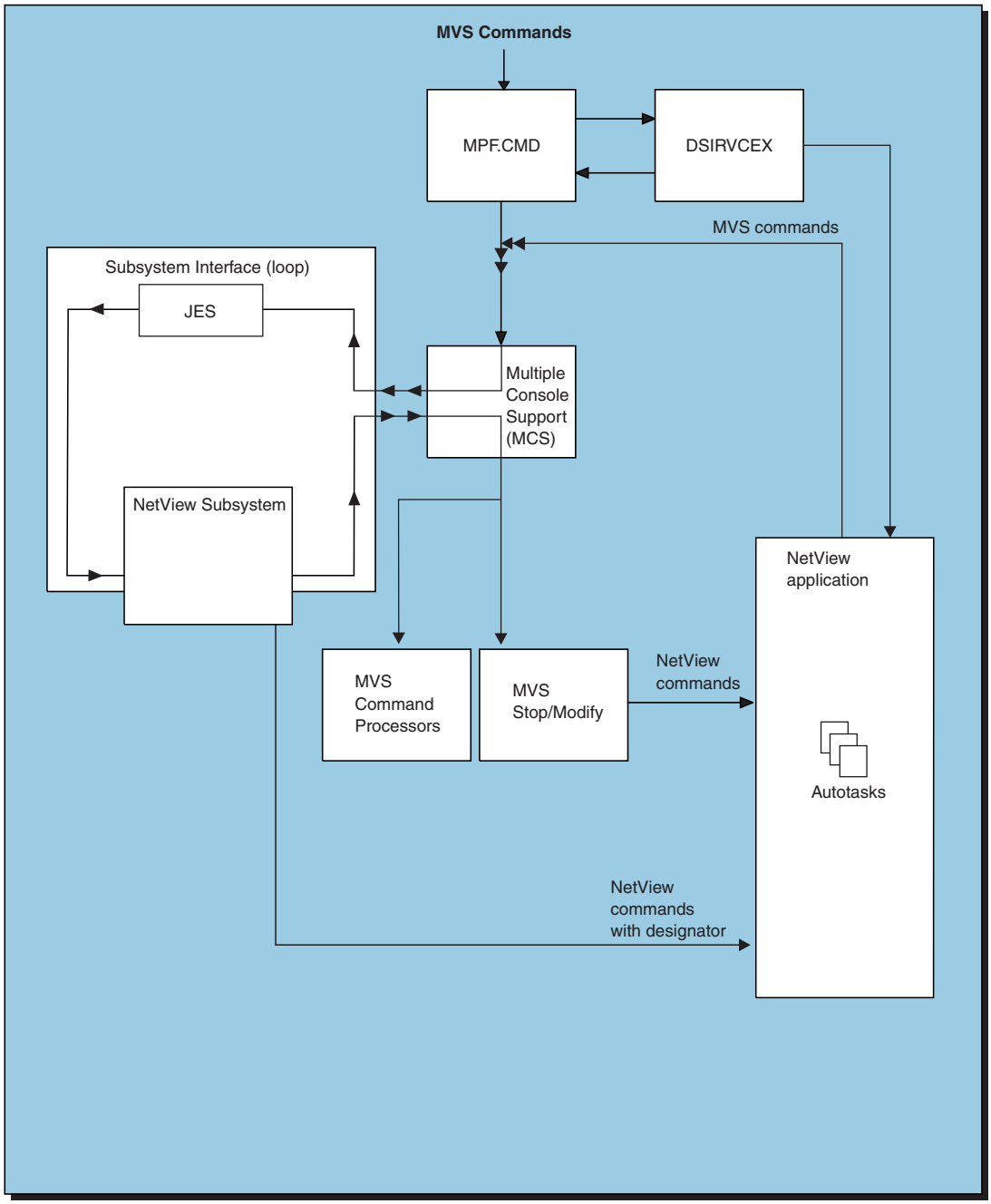


Figure 84. Command Flow between z/OS and NetView

The NetView subsystem address space acts as a z/OS subsystem. It selects messages that are broadcast on the z/OS subsystem interface and forwards copies of the selected messages to the NetView application address space for automation processing. If you are using EMCS consoles, the subsystem address space is used to receive commands, not messages. For more information about the flow of messages within the z/OS system and across the subsystem interface into the NetView system, see Appendix D, “MVS Message and Command Processing,” on page 523.

The NetView application address space performs all network management functions, system and network message processing, and presentation services for NetView. It contains the automation table and autotasks that you use for automation.

Defining the NetView program to a z/OS system as a Subsystem

Define NetView as a z/OS operating system subsystem for these reasons:

- To use the subsystem interface for system, subsystem, and application messages
- To enable you to enter NetView commands from z/OS consoles
- To enable the program-to-program interface

To define the NetView program as a subsystem, update the IEFSSNnn member of SYS1.PARMLIB. The IEFSSNnn member contains parameters that define the secondary subsystems during z/OS system initialization. Each 80-byte IEFSSNnn record contains parameters defining a single secondary subsystem.

The entry name of the NetView subsystem is the 4-character name of the NetView subsystem. The first 4 characters of the names of the start-up procedures for both the application address space and the subsystem address space must match the 4-character subsystem name you define for NetView. For example, in the samples shipped with the NetView product, the start-up procedure for the application address space and the one for the subsystem address space both begin with CNMP. If you use this procedure, include the `CNMP` entry in the IEFSSNnn member. The definition takes effect the next time you IPL the z/OS system.

Ensure that the values you specified for MAXUSER and RSVNONR in the IEASYSnn member of SYS1.PARMLIB at installation are adequate for the number of times you expect to stop and restart the NetView program. Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for more information.

Ensuring That z/OS Forwards System Messages to the NetView program

You can forward system messages from the z/OS system to the NetView program in two ways:

- Through the subsystem interface
- Through EMCS consoles

You can obtain more information about system messages if you use EMCS consoles, because z/OS sends the messages in message data blocks (MDBs) instead of write-to-operator queue elements (WQEs). MDBs include additional information, such as the color in which the message is displayed.

For the NetView program to have access to the z/OS system, subsystem, and application messages, the z/OS program must broadcast the messages on the subsystem interface. NetView examines each message on the subsystem interface unless you specified AUTO(NO) for the message in MPF. If you use EMCS consoles, the CNMCSSIR task receives by default all messages marked AUTO(YES) or AUTO(*token*) in the MPF table. Other extended consoles being used by NetView are set up by default to receive only their own command responses or WTOs directed by console ID or console name. Then NetView ASSIGN command and automation-table processing occurs, and you can route or automate the message. See “Suppressing System Messages” on page 299 for information about using MPF's AUTO function to determine which messages NetView is to examine. Be sure to use AUTO(YES) for each message that you want to forward to the NetView program for use in automation.

You can change the attributes of your EMCS consoles to enable delivery of messages with certain route codes to the consoles you specify. You can specify route codes with the RACF OPERPARM segment, or the ROUT keyword on the z/OS VARY command. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for a description of attributes for the extended console.

If you change the attributes of your EMCS consoles, ensure that you have extended consoles set up to receive all the messages that were previously received by the CNMCSSIR task.

If a z/OS command is issued from a console owned by NetView and the response is marked AUTO(YES) and SUP(YES), the message is automated under the CNMCSSIR task. The message is treated as an unsolicited z/OS system message.

Dynamically Defining EMCS Consoles

The console name is the same as the NetView operator ID. You can obtain an extended MCS console by using the GETCONID command.

Extended MCS console names must be unique across your system and across a sysplex. Any names that are defined in the CONSOL*xx* member of your SYS1.PARMLIB are not available as extended MCS console names.

The GETCONID Command: This command obtains a console for an operator, autotask, or the primary program operator interface task (PPT). A console obtained with the GETCONID command can have a different name than the default for that task. Specifying a name other than the default helps you comply with the z/OS restriction that console names must be unique within a sysplex. Refer to the NetView online help for a complete description of the GETCONID command and its parameters.

Note: Use the GETCONID command if you are sure that the operator or autotask will issue z/OS commands, or needs to receive z/OS messages directed to that console name. Use SETCONID to assign a unique name to the console without allocating it for operators or autotasks that are less likely to enter z/OS commands.

Consider using the GETCONID command in each operator or autotask initial command list so you can control the STORAGE, QLIMIT, ALERTPCT, and QRESUME parameters. You can specify these parameters with the GETCONID command, but you cannot use the z/OS VARY command to change them. These parameters are:

STORAGE
Specifies the maximum megabytes allocated to the z/OS data space for extended MCS console messages. This storage is for all the extended MCS console messages coming to NetView, not just the messages for this console. The first extended MCS console you define specifies the maximum storage. To change this storage value, you need to release all extended MCS consoles and then issue the GETCONID command again with the new storage maximum.

NetView issues message DWO201I when this storage is full.

QLIMIT
Specifies the number of messages that can be queued for this console at any one time in the extended MCS console data space for NetView.

NetView issues message DWO202I when QLIMIT is reached.

ALERTPCT
The z/OS program issues a warning message (DWO204I) to the console when a certain percentage of QLIMIT is reached. ALERTPCT specifies that percentage.

If QLIMIT is reached, the z/OS system stops queuing messages for that console. The NetView program retrieves messages from the queue until a certain level of messages (the QRESUME value) is reached, and then z/OS resumes queuing messages. However, all messages for the console from the time queuing stopped until the time queuing resumed are lost. You can trap this message and take immediate action (such as switching message traffic to another console) to prevent loss of messages.

QRESUME
Specifies the percentage of QLIMIT that must be reached before queuing resumes.

The NetView program issues message DWO608I when the value of QRESUME has been reached.

If the default values for GETCONID are sufficient, you can use command authorization to prevent operators from entering values for STORAGE, QLIMIT, QRESUME, and ALERTPCT. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for more information about restricting keywords.

If you want to use values other than the defaults, code them in each initial command list used by operators having access to an extended MCS console. With this approach, the initial command list that runs first sets the appropriate values.

The SETCONID Command: This command enables you to dynamically pick a name for the console that an operator will use. Unlike the GETCONID command, the console is not obtained when the command is issued. This command is useful during the running of a clist when the operator logs on, or when an autotask starts to reserve a unique console name in a SYSPLEX. This reduces system overhead because the console is not obtained until it is needed (a z/OS command is entered by that operator).

The RELCONID Command: Use the RELCONID command to release an extended MCS console or subsystem console for the operator, autotask, or PPT. You can also define an alternative z/OS console group and use the SWITCH parameter to route message traffic to the alternative group when you release the console.

Messages are lost when you release the console, and generally you do not need to release the console. However, the SWITCH parameter is useful during logoff if you want to transfer message traffic to an alternative console group. z/OS rules determine which console in a group receives the messages.

Refer to NetView online help for a complete description of the RELCONID command.

Note: Beginning with z/OS V1R8, the SWITCH parameter is not supported.

Reviewing the NetView Start-up Procedures

CNMPSSI (CNMSJ010) in the CNMSAMP library is the sample start-up procedure for the NetView subsystem address space. CNMPROC (CNMSJ009) is the sample start-up procedure for the NetView application address space. In the NetView samples, CNMP is the 4-character subsystem name defined to the z/OS system in IEFSSN*nn*.

Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for a description of the symbolic parameters in the sample CNMPSSI procedure. You can adjust the parameters to meet your own installation requirements.

You can start the NetView program before you start JES and VTAM and have the NetView program automate the start-up of JES, VTAM, other subsystems, and applications. The advanced automation sample set for initialization takes that approach. If you want to start the NetView program first, see "Preparing to Use the Advanced Automation Sample Set" on page 600 for information about the system definition changes required.

Adding CMDDEF Statements to Allow System Commands from the NetView Program

System automation is based on the ability to issue z/OS system, subsystem, and application commands from the NetView program. The NetView program provides a z/OS command processor that enables a NetView operator to enter a z/OS system, subsystem, or application command from the NetView by preceding the command with z/OS. Additional actions are not necessary.

As long as a z/OS system or subsystem command is not also a NetView program or VTAM command, you can set up a CMDDEF statement for it in CNMCMD. This enables you to enter specific z/OS system and subsystem commands from the NetView program without preceding them with z/OS.

The syntax of the CMDDEF statement for a z/OS system or subsystem commands is:

```
CMDDEF.commandname.MOD=CNMCMJC
CMDDEF.commandname.TYPE=R
CMDDEF.commandname.RES=Y
```

Where:

commandname
Is any z/OS system or subsystem command name.

For examples of CMDDEF statements that define z/OS, JES2, and JES3 commands in this manner, refer to members CNMS6401, CNMS6402, and CNMS6403 in the advanced automation sample set.

Note: Many common system operator command verbs are spelled like NetView commands. For example, the system commands VARY, MODIFY, DISPLAY, and REPLY have the same names and abbreviations as the ACF VTAM commands in NetView, and the z/OSabbreviation for HOLD is the same as the NetView-defined command synonym for the HELP command. You cannot change the name of a z/OS command. Avoid defining these z/OS verbs, or rename the appropriate CMDDEF statements.

Defining and Activating Autotasks

Autotasks can issue commands and respond to messages. Autotasks are vital to both system and network automation. Because autotasks are operator station tasks (OSTs), they require OST definition statements in the NetView program. Include OST definition statements in DSIOPF for all of the autotasks you need to implement your automation plan.

For a discussion of how to define autotasks, see "Defining Autotasks" on page 121.

Chapter 19. Suppressing Messages and Filtering Alerts

Message suppression and alert filtering are vital first steps toward automated operations. Suppression and filtering can relieve the operator of viewing information that does not require operator intervention. Message suppression can also relieve NetView automation facilities of the burden of handling many informational messages.

The sample set for automation provides lists of messages that are good candidates for suppression. The sample set for automation also provides a log analysis program that can help you identify messages to suppress. For more information, see "Log Analysis Program" on page 463.

Suppressing System Messages

Prior to NetView for z/OS Version 5 Release 2, the best way to suppress many system messages was through the MVS message processing facility (MPF) table or MPF exit. For more information about MPF, refer to the z/OS library.

With NetView for z/OS Version 5 Release 2, a message revision table can be coded, which has much more capability than MPF. These references provide more information:

- Description of the message revision table, found in "Message Revision Table" on page 25.
- The sample CNMSMRT1 table.
- The REVISE command in the NetView online help or *IBM Tivoli NetView for z/OS Command Reference Volume 2 (O-Z)*.

Suppressing Network Messages

Some messages, such as network messages, do not go through the operating system's message processing facilities. To suppress unnecessary messages of this sort, you can use the automation table. You can write automation-table statements that select exactly the messages you want to suppress, based on message ID, message text, or many other message attributes. Chapter 15, "The Automation Table," on page 147 explains how to code automation-table statements. For examples, see "Writing Automation Table Statements to Automate Messages" on page 318.

After identifying a message, you can use the DISPLAY(NO) action to suppress it from display. You can also use the HCYLOG, NETLOG, and SYSLOG actions to specify whether NetView must log the suppressed message. "Actions" on page 209 discusses all of these automation-table actions.

Filtering Alerts

NetView provides two sets of filters to assist you with alert management:

- Recording filters

  Determine which records NetView logs in the hardware monitor databases. You can use them to avoid accumulating unnecessary data. Recording filters also

allow you to generate messages from alerts, route alerts to a focal point, and select alert color and highlighting options.

- Viewing filters

  Limit the information displayed to individual operators. Viewing filters allow operators to display only the alerts for which they are responsible, without sorting through all the information in the hardware monitor databases.

Recording Filters

You can set recording filters with the hardware monitor's SRFILTER (SRF) command or with NetView automation-table actions.

An alert-type problem record flows first to the hardware monitor. There, any SRFILTER commands that you have issued determine the problem record's initial filter settings. Next, if the record is eligible for automation, it flows to the automation table. Statements in the automation table can use SRF, COLOR, and other actions to override the initial settings for the alert. Finally, the resulting settings take effect, and NetView processes the record according to your specifications. For complete routing information, see "NetView Program Hardware-Monitor Data and MSU Routing" on page 96.

Note: When you use the SRFILTER command to block a record, the record still goes to the automation table. The automation table has an opportunity to override the BLOCK setting.

Unsolicited records coming to the hardware monitor go into the database only if they pass recording filters. You can use several levels of filtering:

ESREC **Event and Statistics Recording Filter.** Defines whether a record must be logged as an event. Operators can view the record on event panels.

AREC **Alert Recording Filter.** Defines whether a record that passes the ESREC filter must also be logged as an alert. Operators can view the record on alert panels.

OPER **Operator Filter.** Defines whether the hardware monitor must generate BNJ146I and BNJ030I messages containing information about the alert. The messages are sent to the authorized receiver and go through normal message processing. The OPER filter applies only if a record passes the AREC filter.

ROUTE **Route Filter.** Defines whether NetView forwards the alert to the hardware monitor's alert focal point in addition to logging the alert locally. You cannot forward an alert unless it passes the AREC filter. Both LUC and LU 6.2 forwarded alerts go through the OPER and COLOR filters again at the focal point. For more information about filtering at the focal point for LUC forwarded alerts, see "Alert Forwarding with LUC" on page 390. For more information about filtering at the focal point for LU 6.2 forwarded alerts, see "Recording Filters for SNA-MDS/LU 6.2 Forwarded Alerts" on page 386.

TECROUTE **TECROUTE Filter.** Defines whether the NetView program forwards the alert to a designated event manager (in addition to logging the alert locally). You cannot forward an alert unless it passes the AREC filter.

TRAPROUT **TRAPROUT Filter.** Defines whether NetView forwards the alert to

the SNMP manager in addition to logging the alert locally. You cannot forward an alert unless it passes the AREC filter.

COLOR **Color and Highlighting Filters.** Defines how the hardware monitor must display the record. You can choose the color of the alert or specify high intensity. You can also choose extended highlighting options (underscoring, blinking, or reverse video) and specify whether an alarm must beep when the record is displayed. Color and highlighting filters, unlike other filters, do not take BLOCK or PASS values.

f

You can set the ESREC, AREC, OPER, ROUTE, TECROUTE, and TRAPROUT filters with the SRFILTER (SRF) command from the hardware monitor or with the SRF action from the automation table. You can set color and highlighting attributes with the SRFILTER command from the hardware monitor using the COLOR parameter. However, you cannot use the SRF action to set color and highlighting attributes from the automation table. Instead, use the COLOR, HIGHINT, and XHILITE actions. The automation table can override any or all of the settings specified by the SRFILTER command.

After a record receives filter settings from the SRFILTER command and the automation table, the hardware monitor examines the resulting settings for inconsistencies. Figure 85 on page 302 shows the hierarchy among the filters; any one of the filters except COLOR can block a record, stopping any of the actions below the filter from taking place.

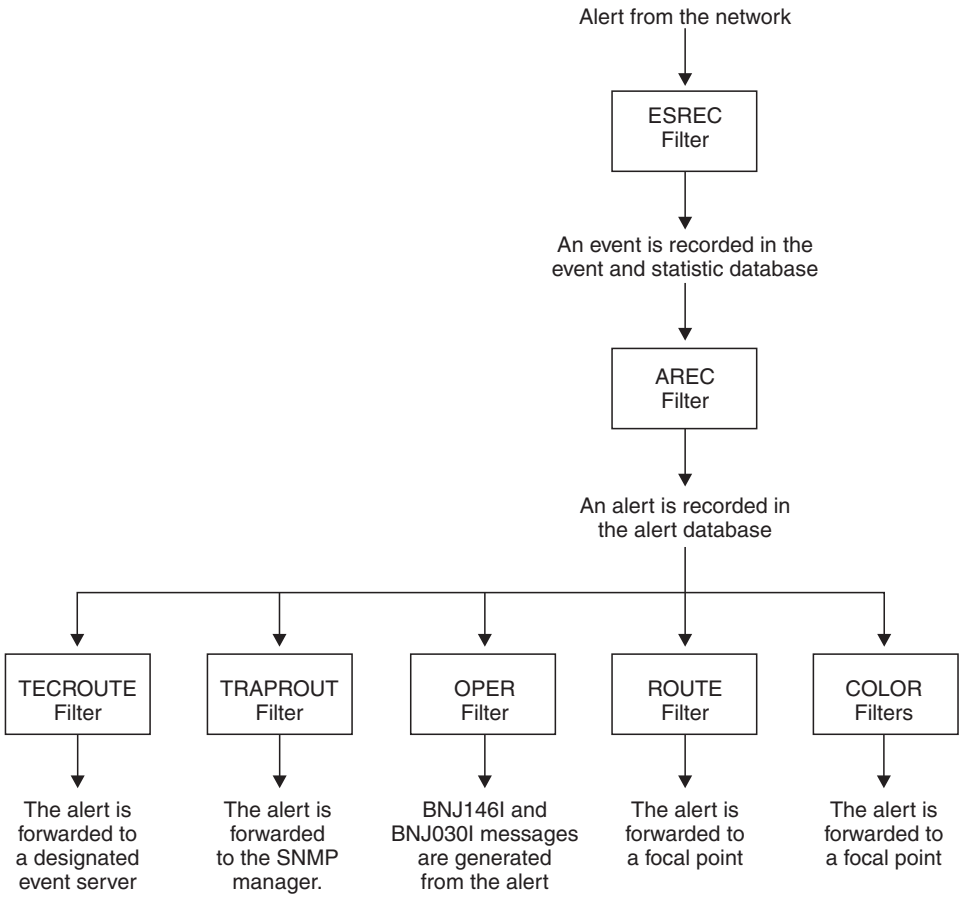


Figure 85. Filter Hierarchy

Statistics, Events, and Alerts

All unsolicited records received by the hardware monitor are classified as either events or statistics. Statistics can lead to events if they exceed established thresholds.

The default SRFILTER setting for all events is PASS. Therefore, each event is placed in the ESREC database. However, there can be times when you want to block certain records from being logged to the hardware monitor database. You can select the events to block with either an SRFILTER ESREC BLOCK command or an automation-table SRF(ESREC BLOCK) action.

If a record passes the ESREC filter, the hardware monitor records it as an event. If the event also passes the AREC filter, the hardware monitor creates an alert from the event. Operators can view the alert on the Alerts-Dynamic panel and other panels. The default AREC settings depend on the event type and the resource type. See NetView online help for information about the defaults.

COLOR and OPER Filters

COLOR filters determine how the alert appears on the Alerts-Dynamic and the Alerts-Static panels. If any filter specifies a color or highlighting value for the alert, the alert appears in that color. Otherwise, the alert is left to default handling. You can specify default handling with color maps (refer to the *IBM Tivoli NetView for z/OS Customization Guide*, SC27-2849) and the SRFILTER COLOR DEFAULT

command. With the standard color maps and COLOR DEFAULT setting, an alert initially appears in white at the top of the Alerts-Dynamic panel and sounds an alarm; otherwise, the alert is displayed in turquoise.

After an alert is recorded, the hardware monitor examines the setting of the OPER-filter attribute for the alert. The OPER filter determines whether NetView must send messages to the NetView authorized operator to describe the alert. The default for the SRFILTER OPER command is BLOCK.

For example, a record passes the ESREC and AREC filters and is displayed on the hardware monitor Alerts-Dynamic panel as shown in Figure 86.

```
DOMAIN RESNAME TYPE TIME   ALERT DESCRIPTION:PROBABLE CAUSE
CNM01  IBMRING LAN  12:30  LOBE WIRE FAULT:RING ADAPTER CABLE
```

Figure 86. Alert Received on the Alerts-Dynamic Panel

If you have set an OPER filter to PASS for resource IBMRING, the NetView program generates the messages for the alert shown in Figure 87.

```
BNJ030I 10/15 12:30 PERM ALERT RECEIVED FROM THE FOLLOWING
        RESOURCE: LAN  IBMRING
BNJ146I 10/15 12:30 N TYPE=PERM PRID=3725 MAJ=01 MIN=01 ACT=10
         HIER=A03NCP,COMC,IBMRING,LAN DOMID=CNM01
```

Figure 87. Messages Generated for Alerts by NetView

Other Recording Filter Information

You can selectively forward alerts to the hardware monitor's alert focal point by using the ROUTE filter. The default ROUTE filter is PASS, meaning that the distributed system forwards all alerts to its focal point. For information about alert forwarding, see Chapter 26, “Centralized Operations,” on page 373.

You can selectively forward alerts to a designated event manager by using the TECROUTE filter. The default TECROUTE filter is BLOCK, meaning that the alert is not forwarded.

You can selectively forward alerts to the SNMP manager by using the TRAPROUT filter. The default TRAPROUT filter is BLOCK, meaning that the alert is not forwarded to the SNMP manager.

You can use the DFILTER command to display filters that you have established with the SRFILTER command. For details about SRFILTER, DFILTER, and other hardware monitor filtering topics, refer to the NetView online help. For examples of how to code automation-table statements that select particular records, refer to “Writing Automation Table Statements to Automate MSUs” on page 322. For information about the SRF, COLOR, XHILITE, and HIGHINT actions used to control filtering from the automation table, see “Actions” on page 209.

Viewing Filters

Viewing filters enable a hardware monitor operator to concentrate on certain parts of the network or certain types of alerts by limiting the alerts that the operator sees. You can select NetView events and alerts for viewing by using the hardware monitor SVFILTER (SVF) command. You can display viewing filter settings with the DFILTER command.

The SVFILTER command affects only the display of the operator whose OST runs the command. With the SVFILTER command, the operators responsible for monitoring the system or network can use filters to exclude any extraneous alert records. This enables the operator to focus on a specified area of responsibility.

The SVFILTER command, like the SRFILTER command, can affect a particular event type, resource name, or resource type, or can affect all resources attached to a specified resource. In addition, you can base viewing filters on the time that NetView received the record. The defaults for viewing filters are PASS.

For example, suppose you have one department dedicated to ensuring that service-level agreements are met in the area of system performance. An operator in that department might set viewing filters to BLOCK for all event types other than PERF. This allows the operator to view only those alerts that affect that particular department (the performance alerts).

To implement effective viewing filters, become familiar with the syntax of the SVFILTER command and its various options. For syntax information, refer to the NetView online help.

Bypassing Filters

In unusual conditions, you might want to bypass normal filtering. You can write an XITCI installation exit routine that gives a return code of 252 or use an XLO action in the automation table to specify external logging only for a record. In this case, NetView sets all filters to BLOCK for the record. You can also give an XITCI return code of 253, in which case NetView sets the ESREC filter to PASS but all other filters to BLOCK. Another installation exit that can alter normal filtering is DSIEX16B. Refer to *IBM Tivoli NetView for z/OS Programming: Assembler* and *IBM Tivoli NetView for z/OS Programming: PL/I and C* for information about installation exits.

Chapter 20. Consolidating Consoles

Attention: Beginning with z/OS v1r8, some of the functions described in this chapter are not supported.

Console consolidation enables you to reduce the number of consoles your operators must monitor. You can combine operations for NetView, the MVS master console, and subsystem consoles on a single NetView command facility screen. Operators can issue MVS commands from the NetView console to perform master console operations.

For example, if an MVS system that is a focal point is monitoring the activities of several MVS systems, you can consolidate messages from all of the systems on one screen. The NetView command facility, at the focal point, displays messages from the target systems to operators in a consistent way, even if you have a variety of operating systems. In addition, you can reduce the number of consoles required for monitoring, possibly to one console. Chapter 26, “Centralized Operations,” on page 373 describes the operation of remote systems and networks from a centralized focal point system.

How to Consolidate Consoles

You can consolidate consoles using the message processing facility (MPF), which can route system messages to NetView. These messages flow to NetView over the subsystem interface or to extended multiple console support (EMCS) consoles being used by NetView, depending on which MVS message delivery mechanism was selected in the CNMSTYLE member.

With extended console support, MPF can route system messages to NetView by sending to the CNMCSSIR task all messages marked AUTO(YES) or AUTO(*token*) in the MPF table, or which are subject to NETVONLY or REVISE("1" AUTOMATE) revision table actions or similar. Use the NetView automation table to route messages to any NetView operator console.

The terminal access facility (TAF) enables you to intercept messages from certain applications or subsystems to their own master terminals or consoles. You can issue commands from NetView as if it were the console of the application. Therefore, you can manage many subsystems or applications from a single command facility screen. The subsystems include but are not limited to CICS and IMS. For more information about using TAF to consolidate consoles, see Table 17 on page 430.

Differences between NetView and Multiple Console Support Consoles

When displaying system and network messages through NetView in an automated environment, be aware of the differences in the way messages are displayed on the command facility screen as compared to the operating system consoles, particularly in an MVS environment. The following sections describe these differences.

Screen Handling and Message Placement

With both the command facility screen and the multiple-support-console console, new lines are written below the last message displayed. If the screen fills on a

multiple-support-console console, the complete screen is rewritten with the newest message on the lowest message line, giving the effect that the whole message area has been shifted up. The oldest (deletable) line is then lost.

If the screen fills in the command facility, the newest message is written to the first message line, which is not necessarily at the bottom of the screen. A wraparound approach is used: each message stays where it is, and a line of dashes divides the newest message from the oldest. As new messages arrive, the dividing line moves down the screen, overwriting the oldest message each time (which is similar to the technique used for JES3 consoles). If you press ENTER, the section of the panel below the line of dashes moves to the top, and the section of the panel above the line of dashes moves to the bottom.

Message Line Format

The multiple support console can optionally precede each message with a time stamp and a JOB, STC, or TSO number that indicates which job, started task, or TSO user issued the message. For NetView command facility screens, you can use a screen format (SCRNFMT) definition to customize the message prefix. Among the items that can be added to the message prefix are:

- The date, in variable format
- The domain name
- The job name and ID for MVS messages
- The TAF session name

The screen format definition can be activated using the DEFAULTS command or the OVERRIDE command. See NetView online help for a complete list of items, and information about the DEFAULTS and OVERRIDE commands.

Messages that arrive in NetView as message data blocks (MDBs) can contain additional source information. You can specify the name of this additional source information as a message prefix. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* and the *IBM Tivoli NetView for z/OS Customization Guide* for more information about screen format definitions.

Display Area Capability

A multiple-support-console console operator can use the MVS CONTROL (K) command to change characteristics of the console. NetView command facility screens do not have an out-of-line display area capability, so all command response information is displayed in-line (similar to the effect of `K A,NONE` for the multiple support console). That means that parts of a response can be overwritten if the message has many lines or occurs at a time of heavy message traffic. Operators can use the AUTOWRAP command to control the flow of messages to a NetView display.

Screen Refresh

NetView command facility screens have a timed AUTOWRAP capability, such as that offered to multiple-support-console consoles using the RTME parameter of the CONTROL command. Operators can use the AUTOWRAP NO command to stop the refresh.

Prefix Command Name

By defining each command in the CNMCMD member, operators can issue MVS commands without the prefix MVS. Sample definitions are provided in the automation samples in the CNMS6401, CNMS6402, and CNMS6403 samples.

However, MVS DISPLAY, VARY, and MODIFY can be issued without the MVS prefix and without any preparation in the CNMCMD member.

If a CNMDEF definition is used to allow JES2 commands to be entered without the MVS prefix, they must be entered in the form `$D J555` rather than `$DJ555` because the NetView program uses the first token as the command. The CNMDEF process can also be used to provide an alternative prefix other than MVS.

Message Holding

Action messages are WTORs or those marked with a Descriptor code that matches one of those specified on the MVSPARM.ActionDescCodes CNMSTYLE statement. Assuming no automation, if an MVS action message is received and displayed at an operator's screen, it is handled as if HOLD(Y) was specified in the automation table entry. Conversely, if a non-action message is automated with DOMACTION(DELMSG) or DOMACTION(AUTOMATE) rather than by HOLD(NO), the message is also held. Either of these message types can be deleted by a DOM. Action messages are highlighted as soon as they are displayed. For either of the above cases, the message does not roll off the screen when more messages arrive. If the task where one of these messages is held receives a subsequent MVS DOM request, any message highlighting is removed. Alternatively, an operator can delete any held or action messages from the NetView screen by putting the cursor on any line of the message and pressing ENTER.

For descriptions of the actions that can be performed in the automation table, see the Actions section in Chapter 13.

Delete operator message (DOM) requests are also passed to NetView-NetView tasks (NNTs). That means that the associated operator station task (OST) at a focal point system automatically removes action messages held on its screen. The OST then reroutes the DOM to all OSTs and NNTs in its own domain, allowing the message to be deleted wherever it might have been routed. If an NNT session breaks after a message has been routed on it, DOM routing does not occur.

Note: NNTs do not support all types of DOMs. Therefore, when you use EMCS consoles, use the DOM(NORMAL) EMCS console attribute.

Color and Other Highlighting Attributes

You can specify the colors to be used in the command facility in any of several ways:

Note: With the Message Revision Table capability added in NetView Version 5.2, most of the functions available to the MPF table can also be performed by using the Message Revision Table, described in “Message Revision Table” on page 25.

- You can set colors with an installation exit.
- If you are using EMCS consoles, the MVS system messages are delivered in the color specified in the MPF table or the Message Revision Table.
- You can set the color, highlighting, and intensity for messages using the COLOR, XHILITE, and HIGHINT actions in the automation table. Colors set with automation table actions override the colors specified in the MPF table or the Message Revision Table. You can change values in the automation table without stopping and restarting NetView.

- You can use a screen format definition to set colors of certain fields on the NetView command facility screen, such as the command area. In addition, the screen format definition can specify colors for action, normal, and immediate messages. The DEFAULTS or OVERRIDE command activates the screen format definition.

For a general description of customizing the NetView command facility screen, refer to the *IBM Tivoli NetView for z/OS Customization Guide*. For specific screen format definition statements, refer to the *IBM Tivoli NetView for z/OS Administration Reference*. Refer to the NetView online help for a complete list of items, and details about the DEFAULTS and OVERRIDE commands.

Notes:

1. MPF table color and highlighting for MVS system messages overrides the screen format definition for message color and highlighting.
2. Automation table specification of color and highlighting overrides the MPF color specification and the screen format definition for color and highlighting. Automation table specification of color and highlighting also overrides the color and highlighting specified with installation exit DSIEX02A or installation exit DSIEX17.
3. Installation exit specification of color and highlighting overrides the MPF color specification and the screen format definition for color. Installation exit DSIEX16 can override the color and highlighting specified in the automation table.
4. When you browse the network log, the messages do not appear in the same colors they appeared in on the NetView command facility screen.

Operators can manipulate each of the color and highlighting attributes independently. For example, an MVS system message that has a match in the automation table with a COLOR action is presented in the intensity and highlighting specified in the MPF table. The color of the message is the color specified in the automation table.

Benefits of NetView Command Facility Screens

The benefits provided by using NetView command facility screens are:

- You can customize the NetView command facility screens by changing the message colors and prefixes.
- An operator can be located some distance from the system (possibly over a communication link), in a geographically remote location.
- Using the NetView BROWSE command, an operator can directly view the network log, operational command lists, and parameter library members of the NetView application. NetView operators can view other system libraries, such as SYS1.PARMLIB, if they are included in the NetView DD statements (for example, as part of DSIPARM). They can also use TAF to browse libraries on other NetView systems.
- The NetView help facility provides information about command syntax and the use of each command, as well as VTAM return-code information. Operators have ready access to help information from any NetView terminal. You can easily add custom online help information for your own operators.
- You can use NetView command security checklists and your own command lists and command processors to provide each operator with appropriate commands and functions.

- A NetView operator can use TAF sessions to other applications such as TSO, CICS, and IMS, eliminating the need for separate application consoles.

Using Multiple-Console-Support Consoles with Autotasks

You can use the AUTOTASK command to associate a multiple-support-console console with a NetView autotask. You can then enter NetView commands from the multiple-support-console console for execution under the autotask and receive messages in response.

The multiple-support-console console also receives all messages that it would receive normally as a console. For consoles assigned with the AUTOTASK command, these considerations apply:

- Definitions for the multiple-support-console console in the CONSOL*xx* member and those of the active MPFLST*xx* apply.
- The display area can still be used only by specific MVS commands. NetView commands do not write output to multiple-support-console console status display areas. Their output appears as normal message traffic.
- NetView commands entered at a multiple-support-console console must be preceded by the designator character string for the NetView subsystem. You specify the designator character string using the MVSPARM.Cmd.Designator statement in the CNMSTYLE member.
- NetView screen control actions such as HOLD(Y) and BEEP(Y) in the automation table have no effect on multiple-support-console consoles that display messages for their associated autotask. Similarly, NetView screen control commands such as INPUT and RETRIEVE have no effect on an EMCS console.
- BROWSE and other full-screen applications are not available under autotasks and so cannot be used from multiple-support-console consoles.
- WTOR and action messages from other domains do not have console characteristics (HELD/HIGHLIGHT) so that they are not confused with the action messages from the system.

Chapter 21. Consolidating Commands

You can consolidate commands by replacing or supplementing a complex process or sequence of commands with a command procedure. Consolidating commands decreases the amount of typing needed to accomplish a process and reduces the chance of a mistake. Consolidating commands ensures that all operators use the same process to accomplish a particular action or to solve a particular problem. Consolidating commands also prepares the way for additional automation, because automation facilities, such as the automation table and timer commands, can use some of the command procedures you develop.

Writing Simple Command Procedures

You can consolidate commands most easily with command lists, which are written in REXX or the NetView command list language. For long, performance-sensitive procedures, you might want to use a command processor written in PL/I or C. You can start by writing command lists and convert some of them to command processors after they have been tested, debugged, and tuned. Chapter 10, "Command Lists and Command Processors," on page 109 discusses command lists, command processors, and the languages available on each operating system. For detailed information about command lists and command processors, see the NetView customization books.

The first step is to identify action sequences that your operators perform repeatedly. Actions in the sequence can include issuing NetView commands, VTAM commands, and system commands. Actions can also include such things as waiting for the messages that result from a command or periodically checking the status of a resource. Good sources of information about common operator actions include operator procedure books, system and network logs, and the operators themselves.

Next, create a command procedure that accomplishes the action sequence you have identified:

- Use a text editor (such as ISPF) to place your instructions in a file. See the NetView customization books for coding information.
- If you are writing a command list with a member name equal to the name of the command list, place the command list in a DSICLD data set.
- If you are writing a PL/I or C command processor, compile the command processor and link-edit it into a STEPLIB data set. Add a CMDDEF statement for the procedure to CNMCMD and stop and restart the NetView program to put the new statement into effect. See *IBM Tivoli NetView for z/OS Programming: PL/I and C* for complete information about defining a PL/I or C command processor to the NetView program.

To illustrate with a simple example, suppose that your operators activate an NCP with the command in Figure 88.

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

Figure 88. Activating an NCP with a Command

To provide them with a shorter command, you can write a command list called ACTNCP1 in the NetView command list language. The command list might look

like Figure 89.

```
ACTNCP1 CLIST
&CONTROL ERR
************************************************
*                                              *
* ACTNCP1 - Activates NCP1                     *
*                                              *
************************************************
V NET,ACT,ID=NCP1,LOAD=YES,LOADSTA=LINK1
&EXIT
```

Figure 89. Sample Command List for Activating an NCP

After you create this command list, operators can issue the command ACTNCP1 (or any command synonyms you define) instead of the whole command.

When your operators are comfortable with the change, you can enhance the command list. For example, you can make it more generic by receiving the name of the NCP to activate as a parameter. You can also have it verify that the NCP was activated successfully.

Anticipating Additional Automation

Many of the command procedures that you create for operators can later be used by automation facilities. For example, you can use an EVERY command to schedule the ACTNCP1 command list shown in Figure 89 so that it runs every day.

Therefore, consider suitability for automation when writing a command procedure. For example, any command procedure that pauses to wait for operator input or uses a full-screen panel for operator input needs to be modified before you can use the procedure from an autotask. Similarly, command procedures that wait for messages must have a time-out value specified so that an autotask does not wait indefinitely if an expected message does not arrive. A command procedure that you intend to use for automation must also provide a good audit trail, so that you can check on the automated actions taking place. For example, a command procedure can send a message to the network log every time it runs.

Modifying Command Procedures

If you want to modify a command procedure after it is in production, make a copy of the procedure under another name or in another data set. Test and tune the procedure under the test name or in the test data set before placing it into production. You can substitute the new procedure for the one in production after you have tested and tuned the new procedure. Waiting until then reduces the chance that an error in an untested procedure affects your production environment.

You can control the order in which the NetView program searches your command-procedure data sets for a command list. You can do so by concatenating the data sets in your NetView start procedure in the order you want them searched. Consider using this order:

1. Production data set
2. Test data set
3. Original command-list data set (CNMCLST)

Documenting Command Procedures

In an automated environment, command procedures take the place of many critical operator activities. Therefore, it is important to create command procedures that are easily understood and maintained.

You must include the information in Table 11 in a consistent format in all command procedure prologs.

Table 11. Documenting Command Procedures

Information	Reason
Creation date	For tracking purposes
Software level of command procedure	For reference, if your system software is changed
Author	For tracking purposes
Function	To provide a quick reference of the procedure's function
Expected input and output	For reference when updating this and other procedures
Variables used	For reference when updating this and other procedures
Procedures that call or are called by this procedure	To enable quick reference to other procedures in the chain
Change activity	To maintain control of change activity

For example, Figure 90 on page 314 shows the documentation within the AOPIGUPD (CNME6401) command list. AOPIGUPD is a bilingual command list in the advanced automation sample set. An asterisk at the beginning of a line indicates a comment in the NetView command list language portion. Comments in the REXX portion begin with /* and end with */ delimiters.

```
/*AOPIGUPD CLIST
&CONTROL ERR
*********************************************************************
*  (C) COPYRIGHT IBM CORP. 2010                                     *
*     IEBCOPY    SELECT MEMBER=((CNME6401,AOPIGUPD,R))             *
*  LAST CHANGE:  12/18/10                                           *
*                                                                   *
*  DESCRIPTION:  SEE COMMENTS AT END OF SAMPLE                      *
*                                                                   *
*  CNME6401 CHANGED ACTIVITY:                                       *
*  CHANGE CODE  DATE      DESCRIPTION                               *
*  -----------  --------  ----------------------------------------*
*********************************************************************
* BUILD THE COMPLEX COMMON GLOBAL VARIABLE                         *
* &COMPLEXVAR IS SET TO THE CONCATENATION OF THE FIRST THREE       *
*   INPUT PARAMETERS                                               *
*********************************************************************
&COMPLEXVAR = &CONCAT &1 &2
&COMPLEXVAR = &CONCAT &COMPLEXVAR &3
*********************************************************************
* SET THE VALUE OF THE CONSTRUCTED GLOBAL VARIABLE                 *
*********************************************************************
&CGLOBAL &COMPLEXVAR
&&COMPLEXVAR = &4
&EXIT
*********************************************************************
* CLIST NAME     :  AOPIGUPD                                       *
* CATEGORY       :  INITIALIZATION UTILITY                         *
* DESCRIPTION    :  SET COMMON GLOBAL VARIABLE VALUE FOR AN        *
*                :  AUTOMATED APPLICATION.  THE DESIRED COMPLEX *
*                :  GLOBAL VARIABLE IS CONSTRUCTED USING INPUT  *
*                :  PARAMETERS &1, &2, AND &3  THE CONSTRUCTED *
*                :  GLOBAL VARIABLE IS THEN SET TO THE VALUE    *
*                :  PASSED IN INPUT PARAMETER &4               *
* INPUT PARMS    :  &1 - RESOURCE COMMON PREFIX (MAX. 5 CHARS)  *
*                :  &2 - FUNCTION IDENTIFIER    (MAX. 3 CHARS)  *
*                :  &3 - APPLICATION IDENTIFIER (MAX. 3 CHARS)  *
*                :  &4 - DESIRED VARIABLE VALUE                 *
* VARIABLES      :  &COMPLEXVAR  IS A TEMPORARY VARIABLE TO HOLD *
*                :    THE NAME OF THE CONSTRUCTED COMPLEX       *
*                :    GLOBAL VARIABLE.                          *
* ACTION         :  VALUE SET FOR COMMON GLOBAL VARIABLE        *
* CALLING CLISTS :  AOPIVARS                                     *
* CALLED CLISTS  :  NONE                                         *
*                                                                *
*********************************************************************
&EXIT
       END OF CLIST */
```

Figure 90. Sample Command Procedure (Part 1 of 2)

```
/* REXX CONVERSIONS */

/* AOPIGUPD Command List */

TRACE E
/*********************************************************************/
/*  (C) COPYRIGHT IBM Corp. 2010                                     */
/*********************************************************************/
/*********************************************************************/
/*  Build the complex common global variable COMPLEXVAR and set      */
/*  it to the concatenation of the first three input parameters      */
/*********************************************************************/
complexvar = MSGVAR(1)||MSGVAR(2)||MSGVAR(3)
/*********************************************************************/
/*  Set the value of the constructed global variable                 */
/*********************************************************************/
INTERPRET complexvar '= MSGVAR(4)'
'GLOBALV PUTC 'complexvar
EXIT
/*********************************************************************/
/*  Command List Name :  AOPIGUPD                                    */
/*  Category          :  Initialization utility                      */
/*  Description       :  Set common global variable value for an     */
/*                    :  automated application.  The desired complex */
/*                    :  global variable is constructed using input  */
/*                    :  parameters 1, 2, and 3.  The constructed    */
/*                    :  global variable is then set to the value    */
/*                    :  passed in input parameter 4.                */
/*  Input Parms       :  MSGVAR(1) - resource common prefix          */
/*                                   (max. 5 chars)                  */
/*                    :  MSGVAR(2) - function identifier             */
/*                                   (max. 3 chars)                  */
/*                    :  MSGVAR(3) - application identifier          */
/*                                   (max. 3 chars)                  */
/*                    :  MSGVAR(4) - desired variable value          */
/*  Variables         :  COMPLEXVAR is a temporary variable to hold  */
/*                    :    the name of the constructed complex       */
/*                    :    global variable.                          */
/*  Action            :  Value set for common global variable        */
/*  Called By         :  AOPIVARS                                    */
/*  Calls             :  None                                        */
/*                                                                   */
/*********************************************************************/
```

Figure 90. Sample Command Procedure (Part 2 of 2)

Chapter 22. Automating Messages and Management Services Units (MSUs)

This chapter describes how you can use the NetView automation table to automate the handling of common messages and management services units (MSUs). It includes information about how you can automate other information by first converting it to messages or MSUs. For example, you can generate messages from hardware monitor records other than MSUs and from status changes detected by the status monitor and the NetView management console.

While reading this chapter, you might want detailed syntactical information about how to code certain automation-table statements. For this type of information, see Chapter 15, "The Automation Table," on page 147.

You can also use the Automated Operations Network (AON) component of NetView to automate handling of common messages and MSUs. See Chapter 31, "Using Automated Operations Network," on page 441 for more information.

Deciding Which Messages and MSUs to Automate

A good way to identify which messages and MSUs to automate is to review with your operators the system and network logs that record activity during a typical shift.

Operators can help identify messages and MSUs that always lead to a predictable sequence of commands at the operator console. For example, a message might require a REPLY command, might indicate that it is time to cancel a job, or might indicate a recoverable device failure. Look for and automate the most obvious candidates first. Later, you can review your environment with your operators to select another set of messages and MSUs to automate.

A log analysis program for MVS can help you analyze the messages in your message logs. See "Log Analysis Program" on page 463. When you are identifying messages to automate, try to handle system messages with the operating system's message processing facility. Use NetView to suppress messages that the operating system cannot suppress, such as network messages or messages on an MVS system that you want to identify by both job name and message ID.

Operators and logs are just two of many sources that can help you find messages and MSUs to automate. See Chapter 3, "Defining an Automation Project," on page 41 for a discussion of several other sources that you might find useful, such as procedure books, problem management reports, help desk logs, service level agreements, and users.

Some messages cannot be automated. For example, messages issued by the DSIPSS macro with TYPE=FLASH are not exposed and cannot be automated.

After you decide on the messages and MSUs you want to automate, you can code automation-table statements to select those messages and MSUs.

Writing Automation Table Statements to Automate Messages

After you decide to automate a message, the next step is determining how to select it, or describe it to the automation table so that it cannot be mistaken for another, similar message. Depending on the message or class of messages you want to automate, you can use the message ID, other specific criteria, or general criteria that select a large class of messages. Many message attributes are available for automation, such as whether a message is solicited or unsolicited and whether a message was issued by an authorized or unauthorized program.

Checking by Message ID

In many cases, you can select a message by using its message ID. The message ID is a key that distinguishes the message from all others and typically appears at the beginning of the message. You can use the MSGID keyword to automate a message based on its ID.

For example, suppose you want to select a NetView message `DSI001I MESSAGE SENT TO taskid`. To select this message and suppress it, you can use the IF-THEN statement in Figure 91.

```
IF MSGID = 'DSI001I' THEN
      DISPLAY(N);
```

Figure 91. Example of Checking a Message by Message ID

The message ID is not always the first word, or token, in a string. In MVS WTORs, the ID is the second token, because the message begins with a reply ID. In the WTOR message `01 AHL125A RESPECIFY TRACE OPTION OR REPLY U`, AHL125A is the message ID. In these cases, you can still use the message ID to select a message. For example, you might select and suppress the preceding WTOR message with the statement in Figure 92.

```
IF MSGID = 'AHL125A' THEN
      DISPLAY(N);
```

Figure 92. Example of Checking an MVS WTOR Message Using the Message ID

Automating Action Messages

Messages that NetView marks as action messages may not have associated DOMs. Therefore, ensure that messages that do not have associated DOMs are accounted for with appropriate automation statements. The DOMACTION(NODELMSG) automation action prevents storage usage associated with waiting for the DOM. See *IBM Tivoli NetView for z/OS Administration Reference* for more information about the MVSPARM.ActionDescCodes CNMSTYLE statement. See "DOMACTION" on page 212 for more information about what action to take regarding a DOM.

A backup approach is to specify a threshold on the MAXCSSIR keyword of the DEFAULTS command. This uses a REXX procedure to remove the oldest, most duplicated messages from the address spaces having the most held messages. Refer to sample CNME1103 for more information.

Checking Other Specific Criteria

Sometimes the message ID is not specific enough to identify a message you want to automate. The message text might contain additional information that you can use to select the message for a specific action. These sections provide examples of

how you can select a subset of the messages with a certain message ID, such as DSI039I MSG FROM AUTOMGR : CHECKING AUTOTASK - AUTOJES.

DSI039I, in the previous example, is a message that one operator or autotask can send to another by issuing the MSG command. In this case AUTOMGR, one of the autotasks in the advanced automation sample set, is issuing the MSG command. If you were to select the message based on the ID alone, you would be automating all messages generated by MSG. But you can use additional criteria to select just those DSI039I messages that you want to automate.

Checking Messages by Domain ID

One criterion you can use is the domain ID. For example, you can select all DSI039I messages from domain CNM01 and place copies in the MVS system log, as shown in Figure 93.

```
IF MSGID = 'DSI039I' &
   DOMAINID = 'CNM01' THEN
      SYSLOG(Y);
```

Figure 93. Example of Checking a Message by Domain ID

Checking Messages with Tokens

You can use tokens to help specify a message for automation. NetView divides the message text into tokens wherever blanks, or spaces, appear. The message DSI039I MSG FROM AUTOMGR : CHECKING AUTOTASK - AUTOJES. has nine tokens:

TOKEN(1) DSI039I
TOKEN(2) MSG
TOKEN(3) FROM
TOKEN(4) AUTOMGR
TOKEN(5) :
TOKEN(6) CHECKING
TOKEN(7) AUTOTASK
TOKEN(8) -
TOKEN(9) AUTOJES

In the DSI039I message, TOKEN(4) is the task that issued the message command. The statement in Figure 94 selects all DSI039I messages from AUTOMGR.

```
IF MSGID='DSI039I' &
   TOKEN(4)='AUTOMGR' THEN
      SYSLOG(Y);
```

Figure 94. Example of Logging A Message Using a Token

Checking Messages by Position

Message DSI039I is arranged so that the content of the message begins in position 29. Therefore, you might use TEXT(29) to obtain the contents of the message. You fail to select the message if you specify only part of the contents, as in the statement in Figure 95.

```
IF MSGID = 'DSI039I' &
   TEXT(29) = 'CHECKING' THEN
      SYSLOG(Y);
```

Figure 95. Example of Logging a Message Using a Text Position

The statement in Figure 95 does not select the message, because TEXT(29) equals CHECKING AUTOTASK - AUTOJES, rather than just CHECKING.

Checking Messages by a Placeholder

In contrast, the statement in Figure 96 uses a placeholder (.) and does select the sample message.

```
IF MSGID = 'DSI039I' &
   TEXT(29) = 'CHECKING' . THEN
      SYSLOG(Y);
```

Figure 96. Example of Logging a Message Using a Placeholder

This statement compares the text beginning in position 29 to the string `CHECKING` followed by anything else, which is indicated by the placeholder.

Checking General Criteria

By using general criteria, you can make your automation statements select a large group of messages instead of a single message ID or domain ID. Again, the message used for demonstration purposes is `DSI039I MSG FROM AUTOMGR : CHECKING AUTOTASK - AUTOJES`.

Checking Criteria with Logical-AND Logic

The statement in Figure 97 selects DSI039I messages from any domain whose name starts with the string CNM. The statement sends these messages to OPER1, if OPER1 is logged on; otherwise, the statement does not affect the routing. The statement compares the domain ID to the string CNM followed by anything, as indicated by the placeholder (.).

```
IF MSGID = 'DSI039I' &
   DOMAINID = 'CNM' . THEN
      EXEC(ROUTE(ONE OPER1 *));
```

Figure 97. Example of Routing Messages Using Logical-AND Logic

Checking Criteria with Logical-OR Logic

You can also make IF conditions more general by using logical-OR logic. You can select a statement if either of two (or more) conditions applies. The statement in Figure 98 selects the DSI039I message if it comes from either AUTOMGR or SYSOP.

```
IF MSGID = 'DSI039I' &
   (TOKEN(4) = 'AUTOMGR' | TOKEN(4) = 'SYSOP') THEN
      EXEC(ROUTE(ONE OPER1 *));
```

Figure 98. Example of Routing Messages Using Logical-OR Logic

Checking Criteria Using Placeholders

You can use placeholders both before and after a string value for comparison. The statement in Figure 99 selects any DSI039I message that has the string CHECKING in it anywhere starting with position 29.

```
IF MSGID = 'DSI039I' &
   TEXT(29) = . 'CHECKING' . THEN
      EXEC(ROUTE(ONE AUTO1));
```

Figure 99. Example of Routing Messages Using a Placeholder

Use caution when using such general comparisons. Too general a comparison can lead to automation of messages that you did not intend to process. The IF condition in Figure 99 matches the example message `DSI039I MSG FROM AUTOMGR : CHECKING AUTOTASK - AUTOJES` but also matches the message `DSI039I MSG FROM OPER1 : I AM CHECKING ON THE STATUS OF THE SPOOL UTILIZATION`.

Do not use a general comparison for a string anywhere in the text of a message unless you use a more specific condition in conjunction with it. In Figure 99 on page 320, the MSGID='DSI039I' condition prevents the statement from matching any other messages that contain the string CHECKING after the 28th position.

Comparing Text with Parse Templates

When you are describing the messages you want to automate, you can perform flexible text comparisons with parse templates.

For example, you might want to automate a message whose contents are not always the same. The data in the message can be different each time the message is displayed, but the location of the data in the message helps identify the message as the one you want to automate.

Using Placeholders in a Parse Template

For example, suppose you want certain actions to occur whenever the message `IEE362A  SMF ENTER DUMP FOR SYS1.MANx ON volser` appears.

The IF-THEN statement in Figure 100 selects that message for automation. This statement uses placeholders (periods) to skip unpredictable text.

```
IF MSGID='IEE362A' & TEXT= . 'FOR SYS1.MAN' . 'ON' . THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

Figure 100. Example of Using a Placeholder in a Parse Template

The statement checks for a message ID of IEE362A and the string FOR SYS1.MAN anywhere in the message text, followed by anything, followed by the string ON, followed again by anything. With a statement like this, you can check a long text string without knowing exactly what data appears in all parts of the string. You do not have to know the sizes or contents of the fields indicated by the placeholders.

Using Variables in a Parse Template

By using variables instead of placeholders, you can extract data from a message. You can then use the variable in the action portion of the statement to represent the extracted data. For example, you can code an IF condition using the variable names LIBIND and VOLSER instead of the second and third placeholders in Figure 100. The statement appears as shown in Figure 101.

```
IF MSGID='IEE362A' & TEXT= . 'FOR SYS1.MAN' LIBIND 'ON' VOLSER THEN
      EXEC(CMD('CLISTA ' LIBIND ',' VOLSER) ROUTE(ONE AUTO1));
```

Figure 101. Example of Using Variables in a Parse Template

The variable LIBIND stores whatever data is in the message between the strings FOR SYS1.MAN and ON. The variable VOLSER applies to whatever data follows the string ON, to the end of the message.

If the message `IEE362A  SMF ENTER DUMP FOR SYS1.MANX ON CPDLIB` occurs, the value of the variable LIBIND becomes X, and the value of VOLSER becomes CPDLIB.

Using Parse Templates with Multiline Messages

You can automate single-line and multiline messages. For multiline messages, you can use a parse template to extract information only from the first non-blank line of the message. You must use a command procedure if you want to extract information from the other lines of the message. Examples of multiline messages

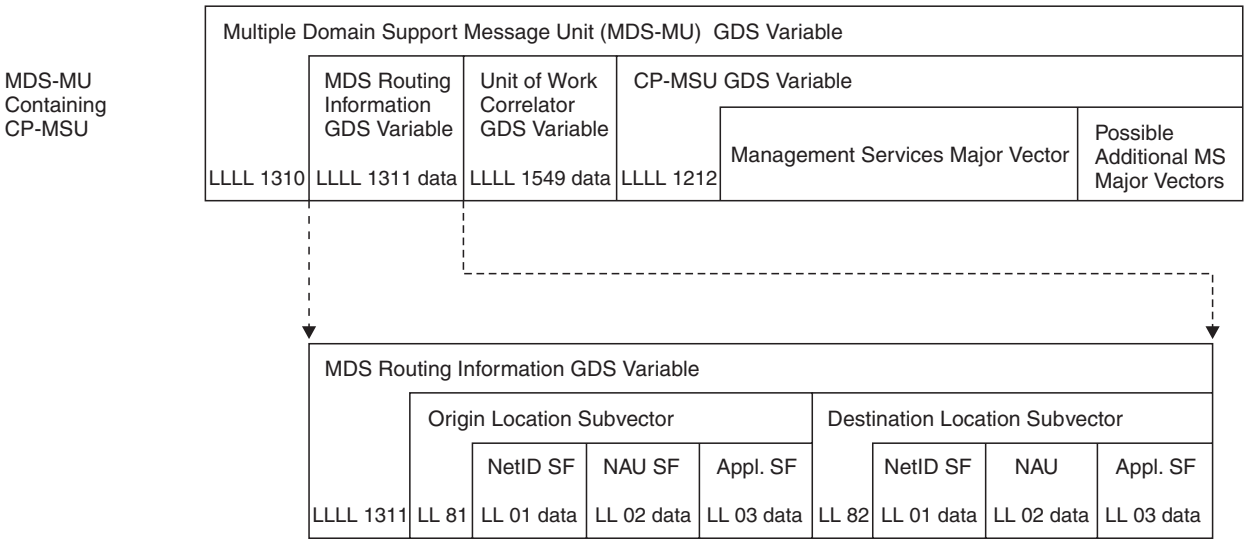
are those that are issued in response to the NetView MAPCL command, the VTAM DISPLAY command, the MVS DISPLAY command, and several forms of the JES2 $D command. Actions that you specify for these messages apply to the entire message, including all the individual lines of the message.

Writing Automation Table Statements to Automate MSUs

The NetView automation table enables you to automate handling of management services units (MSUs). You can automate five types of MSUs:

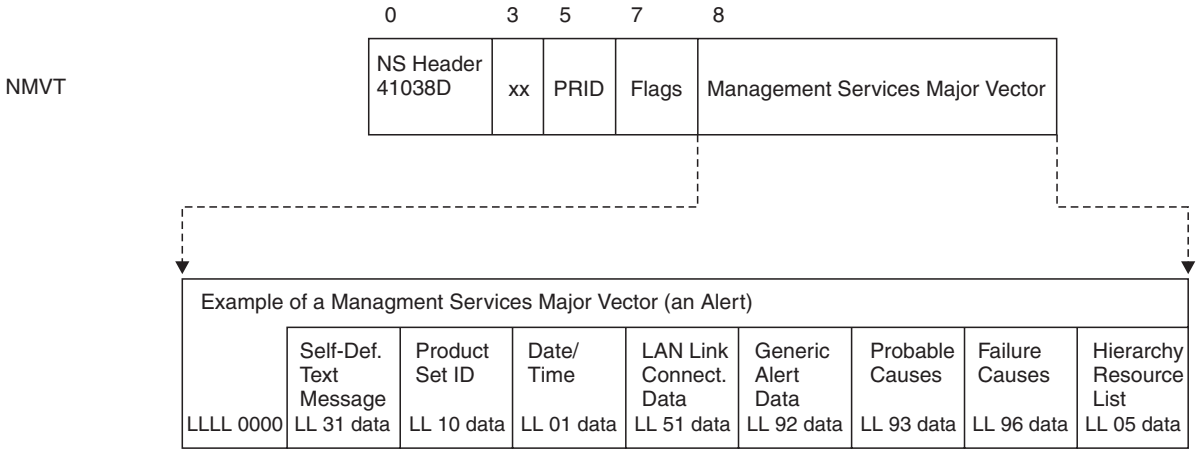
- NMVTs
- CP-MSUs
- MDS-MUs
- RECMSs
- RECFMSs

Figure 102 shows the structure of a multiple domain support message unit (MDS-MU) that contains a control point management services unit (CP-MSU). Figure 103 on page 323 shows the structure of a network management vector transport (NMVT). For more information, refer to the Systems Network Architecture library.



All numbers are in hexadecimal format.
LL or LLLL refers to a 1- or 2-byte field specifying a structure's length.

Figure 102. Conceptual View of a CP-MSU



All numbers are in hexadecimal format.
LL or LLLL refers to a 1- or 2-byte field specifying a structure's length.

Figure 103. Conceptual View of an NMVT

Notice that the data being conveyed, in this case an alert, lies in a structure called a ***management services major vector***. A management services major vector includes these major vectors:

- X'0000'
- X'0001'
- X'0002'
- X'0025'
- X'1332'
- X'1044' (encapsulated RECMS)
- X'1045' (encapsulated RECFMS)

A management services major vector looks the same in a CP-MSU as in an NMVT.

You can use the RATE statement to suppress repetitive MSUs from resources. MSUs that are blocked by a filter generated as a result of the RATE function are not passed to automation. If you want these MSUs to be automated, add an AUTORATE statement to the BNJDSERV DST initialization member. Refer to the RATE and AUTORATE statements in the *IBM Tivoli NetView for z/OS Customization Guide*.

For these examples, suppose you want to automate an alert that you are receiving from a local area network. The alert comes to NetView in an NMVT, and you decide to select the NMVT for automation. You might start by observing an instance of the alert on the hardware monitor's Alerts-Static panel. From there, you can type `DM` to get the detail menu and choose `1` to get a hexadecimal display of the NMVT. Paging through two or three panels, you can view the entire contents of the alert, as shown in Figure 104 on page 324.

```
TIVOLI NetView         SESSION DOMAIN: CNM99    R350581   12/02/10 15:11:18

NPDA-44C           * HEXADECIMAL DISPLAY OF DATA RECORD *          PAGE 1 OF 3


 CNM01       DEVLAN3      LANMGR      DEVLAN3

            +--------+   +--------+     ----
 DOMAIN     |  SP    |---|  TP    |---( LAN  )
            +--------+   +--------+     ----
DATE/TIME: 05/02 15:10

   NMVT MAJOR VECTOR 0000 - 41038D 0000000000 0157 0000
   SUBVECTOR 31
B1310602 02B90025 07110349 00140003 21119F30 E3888540 D9899587 40C59999
969940D4 969589A3 9699404D D9C5D45D 408881A2 408485A3 8583A385 8440A388
81A34081 95408184 8197A385 994089A2 4085A797 85998985 95838995 874085A7
8385A2A2 89A58540 83969587 85A2A389 96954081 95844089 A2408489 A2838199
84899587 408140A2 89879589 86898381 95A34095 A4948285 99409686 40869981
9485A24B 40404DC4 C6C9D7C4 F1F0F7C5 5D
   SUBVECTOR 10
2E10002B 11040908 F5F6F0F1 F2F2F708 04F0F2F0 F0F0F017 06C9C2D4 40D3C1D5
40D4C1D5 C1C7C5D9 40404040 4040
   SUBVECTOR 01
0A010810 5B05020F 0B39
   SUBVECTOR 51
20510402 057C0808 10005A74 355C1228 40404040 40404040 40404040 40404040
   SUBVECTOR 92
0B920000 03501157 D16A21
   SUBVECTOR 93
04933223
   SUBVECTOR 96
0E960601 10223324 06812010 3101
   SUBVECTOR 05

2D052B10 00098395 94F0F140 4040003E 08938195 D3C1D5F3 008107D3 C1D5D4C7
D9001808 938195D3 C1D5F300 39


 ???
CMD==>
```

Figure 104. Hardware Monitor's Hexadecimal Display of Data Record

Checking for Field Existence

To select an MSU for automation, you can use the MSUSEG compare item (see "MSUSEG" on page 195 for syntax rules). Begin at the major-vector level and work your way down. For example, you might select all MSUs that pass through the hardware monitor and contain alert major vectors, which have a key of X'0000', and color them green, as shown in Figure 105.

```
IF HMONMSU = '1' & MSUSEG(0000) ¬= '' THEN
      COLOR(GRE);
```

Figure 105. Example of Selecting an MSU

In the example, MSUSEG(0000) returns the entire contents of the alert major vector, if one exists. By comparing the result to the null ('') keyword, you can ignore the contents of the alert major vector and merely check whether such a major vector exists.

Checking Subvectors

To move down in levels, use the period (.) character. The level below the major vector is the subvector. If you want to select only those X'0000' major vectors that

contain X'31' subvectors (self-defining text messages), you can use the statement in Figure 106.

```
IF MSUSEG(0000.31) ¬= '' THEN
      COLOR(GRE);
```

Figure 106. Example of Selecting a Subvector

Checking Subfields

You can also go to the subfield or sub-subfield level as shown in Figure 107.

```
IF HMONMSU = '1' & MSUSEG(0000.51.02) ¬= '' THEN
      COLOR(GRE);
```

Figure 107. Example of Selecting a Subfield

The example statement selects any MSU that passes through the hardware monitor and contains a X'0000' major vector (alert) with a X'51' subvector (LAN link connection subsystem data) that in turn contains an X'02' subfield (ring or bus identifier).

Checking Field Contents

To be more specific, you might want to test the contents of a particular field, rather than just testing for existence. For example, you can test for an alert that passes through the hardware monitor and whose probable cause, given in subvector X'93', is X'3223'. As explained in *Systems Network Architecture Formats* a probable cause of X'3223' means a token-ring adapter interface. To select alerts with this probable cause, you might code the statement shown in Figure 108.

```
IF HMONMSU = '1' & MSUSEG(0000.93) = HEX('04933223') THEN
      COLOR(GRE);
```

Figure 108. Example of Checking the Contents of an MSU Subvector

Here, MSUSEG returns the entire contents of the subvector X'93', including the length byte (X'04' in this case) and the key (X'93'). However, you can skip the length and the key by specifying a byte position. A position of 1 is the default and starts the comparison at the first byte, which is a length byte. This is different from the notation described in *Systems Network Architecture Formats*, where 0 designates the first byte. The statement in Figure 109, using a position of 3, skips the length byte and the key byte, giving you the remainder of the data.

```
IF MSUSEG(0000.93 3) = HEX('3223') THEN
      COLOR(GRE);
```

Figure 109. Example of Checking the Contents of a Position in an MSU Subvector

In an automation table statement, you can also use a placeholder (.) or assign a value to a variable. Placeholders and variables work the same with MSUs as they do with messages. For instance, the statement in Figure 109 checks whether subvector X'93' contains exactly the data X'3223'. But you can check whether the subvector merely *begins* with the data X'3223' by adding a placeholder at the end, as in Figure 110.

```
IF MSUSEG(0000.93 3) = HEX('3223') . THEN
      COLOR(GRE);
```

Figure 110. Example of Using a Placeholder to Check the Contents of a Position in an MSU Subvector

Checking for RECMSs and RECFMSs

When the hardware monitor submits RECMSs and RECFMSs to automation, it encapsulates them within a designed major vector. The X'1044' major vector is used for RECMSs. The X'1045' major vector is used for RECFMSs.

RECMS 82

Figure 111 is an example of a RECMS 82 (recording mode) received by the hardware monitor.

```
01038103C4000182F04000zzzz
```

Figure 111. RECMS 82

Where:

X'010381' Is the RECMS header.

X'82' Indicates RECMS 82 (see byte 8). RECMS 82 starts with offset 1 instead of offset 0.

F04000zzzz Is the remainder of RECMS 82.

Refer to the Network Control Program library for information about RECMS and RECFMS record formats.

Encapsulated RECMS

When the hardware monitor receives this RECMS 82, it encapsulates it in a CP-MSU that contains major vector X'1044'. An example is shown in Figure 112 on page 326.

```
LLLL1212nnnn104401038103C4000182F04000zzzz
```

Figure 112. RECMS Encapsulated in X'1044'

Where:

LLLL Is the 2-byte length. The length equals the sum of:

- **X'D'** Length of the RECMS. Your RECMSs can be longer.
- **X'2'** Length of X'1044'.
- **X'2'** Length of `nnnn`.
- **X'2'** Length of 1212.
- **X'2'** Length of `LLLL`. In Figure 112, the 2-byte length of `LLLL` is X'0015'.

1212 Indicates a CP-MSU.

nnnn Is the 2-byte length. The length equals the sum of:

- **X'D'** Length of the RECMS. Your RECMSs can be longer.
- **X'2'** Length of X'1044'.
- **X'2'** Length of `nnnn`. In Figure 112, the 2-byte length of `nnnn` is X'0011'.

X'1044'
Is the major vector key indicating a RECMS.

X'010381'
Is the RECMS header.

X'82' Indicates RECMS 82 (see byte 16). RECMS 82 starts with offset 1 instead of offset 0.

F04000zzzz
Is the remainder of RECMS 82.

Example: Checking for a RECMS with a Recording Mode of X'82'

```
IF MSUSEG (1044 12) = HEX('82') . THEN
    COLOR(GRE);
```

This example checks the 12th byte for a X'82', indicating a RECMS 82, and if found, colors the MSU green. You not include the 2-byte length and the X'1212' of the CP-MSU.

Note: RECMSs do not support subvector, subfield, and sub-subfield keys, and RECFMSs support only a limited number of subvectors. You cannot use MSUSEG to access any subvectors, subfields, or sub-subfield keys in RECMSs and RECFMSs.

MSU Actions

The actions you can specify for an MSU include issuing a command, command list, or command processor with the EXEC action. EXEC is available for any MSU.

The other actions control how the hardware monitor processes alerts. These actions have meaning only for MSUs containing alert major vectors and passing through the hardware monitor.

Some actions set highlighting attributes for the alert:

XHILITE Sets a foreground highlighting option, such as blinking text, underscoring, or reverse video

COLOR Lets you choose a color for color monitors

HIGHINT Determines whether the high-intensity 3270 setting is used for monochrome monitors

BEEP Determines whether an audible alarm sounds

The remaining actions control recording:

SRF Sets recording-filter attributes and determines whether the MSU passes ESREC, AREC, OPER, ROUTE, TECROUTE, and TRAPROUT filters.

- Set the ESREC filter to pass for the AREC filter to function. These are methods of setting the ESREC filter:
  - The automation table
  - The SRFILTER command
  - The DSIEXI6B installation exit
- Set the ESREC and AREC filters to pass for the OPER, ROUTE, TECROUTE, and TRAPROUT filters to function as follows:
  - The OPER filter controls message generation.
  - The ROUTE filter controls alert forwarding.
  - The TECROUTE filter controls alert forwarding to the designated event server.
  - The TRAPROUT filter controls alert forwarding to the SNMP manager.

XLO Specifies that none of the recording filters take effect, and the MSU goes to external logging only

Highlighting and recording attributes that you set in the automation table override those specified by the hardware monitor. For example, the SRF action overrides the hardware monitor SRFILTER command. However, installation exit DSIEX16B can override even the automation table.

Hexadecimal, Character, and Bit Notations

It is often convenient to use hexadecimal notation when working with MSUs. However, you might prefer character notation in some statements. Character notation is helpful when the MSU contains an EBCDIC representation of character data.

The sample alert contains EBCDIC characters in subvector X'05' (the hierarchy/resource list) in subfield X'10' (hierarchy name list). You can use either hexadecimal or character notation to test the hierarchy name list.

Using Hexadecimal Notation

For example, suppose you want to block the sample alert from being recorded in the alert database, based on the first resource in the list. As explained in *SNA Formats*, the first resource begins in position 5 of subfield X'10'; therefore, you can code the MSUSEG statement with hexadecimal notation, as shown in Figure 113.

```
IF MSUSEG(0000.05.10 5) = HEX('C3D5D4F0F1') . THEN
      SRF(ESREC PASS)
      SRF(AREC BLOCK);
```

Figure 113. Example of Using Hexadecimal Notation

Using Character Notation

You can also use the equivalent character notation, as shown in Figure 114.

```
IF MSUSEG(0000.05.10 5) = 'CNM01' . THEN
      SRF(ESREC PASS)
      SRF(AREC BLOCK);
```

Figure 114. Example of Using Character Notation

Using Bit Notation

Another option is to specify a bit position. With a bit position, the rules of the comparison change, and the item you specify on the right side of the expression must be a bit string. Like byte positions, bit positions begin at one (1) rather than zero (0). Figure 115 uses a bit position and a bit string to test for the hierarchy name list (subfield X'10').

```
IF MSUSEG (0000.05.10 5 1) = '10000011100101011001010011110000111110001' THEN
      SRF(ESREC PASS)
      SRF(AREC BLOCK);
```

Figure 115. Example of Using Bit Notation

A placeholder is not used in Figure 115, because bit-string comparisons test only as many bits as you provide. You can also use Xs in the bit string if you want the comparison to skip specified bits.

The location specification is in hexadecimal, while the byte and bit positions are in decimal numbers. In Figure 115, for example, the X'0000', X'05', and X'10' are in hexadecimal, while the 5 and the 1 are decimal numbers.

When a Field Occurs More than Once

Sometimes, an MSU contains more than one instance of a particular major vector, subvector, or other field. To check an instance other than the first, use an occurrence number.

For example, the statement in Figure 116 highlights an alert if its second subvector X'10' (product-set ID) contains the string IBM LAN MANAGER.

```
IF MSUSEG(0000.10(2)) = . 'IBM LAN MANAGER' . THEN
      XHILITE(REV) COLOR(BLU) BEEP(YES);
```

Figure 116. Example of Checking Multiple Occurrences of a Field

The sample alert in Figure 104 on page 324 fails the test, because it only has one X'10' subvector. However, the sample alert passes the test if you check all X'10' subvectors at once. You can do this by using an asterisk (*) for the occurrence number as shown in Figure 117.

```
IF MSUSEG (0000.10(*)) = . 'IBM LAN MANAGER' . THEN
      XHILITE(REV) COLOR(BLU) BEEP(YES);
```

Figure 117. Example of Checking All Occurrences of a Field

In Figure 117, the asterisk results in a match if the comparison evaluates as true for any subvector X'10' in the first major vector X'0000'. You can also use occurrence numbers or asterisks at other levels such as the major-vector and subfield levels. For an MSU that comes through the hardware monitor, NetView separates extra major vectors into individual MSUs prior to automation.

The default at each level is to check only the first occurrence of a specified field. The statement in Figure 118 determines whether any X'0000' major vectors contain X'10' subvectors, the first of which contains any X'11' subfields, the second of which contains any X'00' sub-subfields. If so, the statement checks the first X'00' sub-subfield to see whether the third byte beginning with the fourth bit contains a 1 followed by a zero (0).

```
IF MSUSEG(0000(*).10.11(2).00 3 4) = '10' THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

Figure 118. Example of Detailed Checking of an MSU Field

Because the sample alert in Figure 104 on page 324 has only one X'11' subfield in its X'10' subvector, it does not satisfy the condition in the statement in Figure 118.

Using Header Information

Figure 102 on page 322 shows that MDS-MUs contain a substantial amount of header information outside of the major vector. In some cases, you might want to automate MDS-MUs based on their header information.

To automate MDS-MUs based on their header information, add an H to the beginning of the MSUSEG. When you use the H, the syntax rules for MSUSEG remain the same. However, the first level of field you specify is the level of a GDS variable within the MDS-MU, rather than a major vector. Therefore, you can obtain information from outside the major vector.

For example, you can examine the data in the MDS routing information GDS variable (X'1311'), destination-location subvector (X'82'), destination-application subfield (X'03'). The statement in Figure 119 on page 330 skips the length byte and

the key byte and obtains the data, which begins in position 3.

```
IF MSUSEG(H1311.82.03 3) = 'APPLA' THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

Figure 119. Example of Checking an MDS Header

You can use the H parameter only for MDS-MUs. NMVTs processed with MSUSEG(H) return a value of null, as do any CP-MSUs that are not within MDS-MUs, such as those from the program-to-program interface. Therefore, you can check for alert major vectors carried in MDS-MUs by entering this statement:

```
IF MSUSEG(H1212.0000) ¬= '' THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

Figure 120. Example of Checking for Alert Major Vectors in an MDS-MU

In Figure 120, H1212 selects a CP-MSU within an MDS-MU, and 0000 checks for an alert major vector.

Using Major Vectors Other than Alerts

Alerts are the most commonly automated major vectors, but you can automate other major vectors (such as X'0001', X'0002', X'0025', X'1332', RECMSs, and RECFMSs).

Checking Resolution Major Vectors

For example, resolution major vectors, which have a key of X'0002', inform you that a problem identified by an alert has now been resolved. Resolution major vectors can be accessed from the hardware monitor BNJDSERV XITCI exit and are forwarded to the automation table for automation processing. Just as with alerts, the hardware monitor displays resolution major vectors, logs them to a VSAM database, and makes them available for hardware monitor filters (set by the SRFILTER command).

Suppose you want to trap each resolution major vector and deliver it along with its entire MSU to CLISTA. CLISTA might be a command list you have written to track the resolution data by sending it over the NetView Bridge to Information/Management. You can enter this statement:

```
IF MSUSEG(0002) ¬= '' THEN
      EXEC(CMD('CLISTA') ROUTE(ONE AUTO1));
```

Figure 121. Example of Automating a Resolution Major Vector

Checking R&TI GDS Variables

If you are working with operations management served applications, an MDS-MU sent from a served application to the hardware monitor can contain a routing and targeting instruction (R&TI) generalized data stream (GDS) variable (X'154D'). The hardware monitor places the routing and targeting information after the alert or resolution major vector, still in the same CP-MSU. Suppose you want to check for a CP-MSU containing an alert major vector (X'0000') and a routing and targeting instruction GDS variable (X'154D') with an origin application name subfield (X'60'). To extract the origin application name and pass it to CLISTA, you can enter this statement:

```
IF MSUSEG(0000) ¬= '' & MSUSEG(154D.60) ¬= '' &
   MSUSEG(154D.60) = ORIGIN THEN
      EXEC(CMD('CLISTA 'ORIGIN) ROUTE(ONE AUTO1));
```

Figure 122. Example of Automating a Routing and Targeting Instruction GDS

The hardware monitor sends X'0000', X'0001', X'0002', X'0025', X'1332', RECMSs (encapsulated in a X'1044'), and RECFMSs (encapsulated in a X'1045') major vectors to the automation table, along with any X'154D' GDS variables that might be appended. However, you can send any major vector to the automation table through the NVAUTO MS application, the CNMAUTO service routine, or the DSIAUTO macro. MSUSEG can process any major vector you send and can accept more than one major vector per CP-MSU. The major vectors must be in valid MSUs.

Using the Resource Hierarchy

When an alert comes through the hardware monitor, NetView builds a resource hierarchy for the alert. The hierarchy can contain up to five resources. As Figure 104 on page 324 shows, the hierarchy for the sample alert has three resources:

DEVLAN3 The service point (SP)
LANMGR The transaction program (TP)
DEVLAN3 The local area network (LAN)

To test the resource hierarchy, use the HIER keyword. If you specify the number of the resource in the list, as shown in Figure 123, HIER returns the 8-character name followed by the 4-character type.

```
IF HIER(2) = 'LANMGR  TP  ' THEN
      COLOR(GRE);
```

Figure 123. Example of Checking a Resource in the Resource Hierarchy

The spacing is important in Figure 123. You need the two spaces after LANMGR to make TP start in the ninth column. The statement in Figure 124 also matches the sample alert.

```
IF HIER(2) = 'LANMGR' . &
   HIER(3) = 'DEVLAN3' . &
   HIER(4) = '' THEN
      COLOR(GRE);
```

Figure 124. Example of Checking Multiple Resources in the Resource Hierarchy

If you omit the resource number, as shown in Figure 125, you get a concatenated string of all the resource names and resource types.

```
IF HIER = 'DEVLAN3' . 'LANMGR' . 'DEVLAN3' . THEN
      COLOR(GRE);
```

Figure 125. Example of Checking All Resources in the Resource Hierarchy

For the sample alert, the resource hierarchy is based on the information in subvector X'05', the hierarchy/resource list. Therefore, you can also obtain resource hierarchy information from MSUSEG(0000.05). However, this is not true for all alerts. The most reliable way to test the hardware monitor resource hierarchy is to use the HIER keyword.

Using the Domain ID

The DOMAINID keyword indicates which NetView domain first received the MSU. Checking DOMAINID is a general test. Use it with other conditions, as shown in Figure 126.

```
IF MSUSEG(0000) ¬= '' &
   HIER(1) = 'DEVLAN3 SP  ' &
   DOMAINID = 'CNM01' THEN
      COLOR(GRE);
```

Figure 126. Example of Using the DOMAINID Keyword

Automating Other Data by Generating Messages

The automation table processes messages and MSUs. There are other types of data that the NetView automation table does not process. If you want to automate responses to these types of data, you must first convert them to messages or MSUs. Two important examples that illustrate this process are hardware monitor data records and status information.

Automating Hardware Monitor Records

You can automate problem notifications sent to the hardware monitor by generating messages from them and sending the messages to automation.

Many problem records sent to the hardware monitor are MSUs. For these records, you have the option of generating messages to automate, or automating the MSUs directly. Direct automation, which is more efficient, is described in "Writing Automation Table Statements to Automate MSUs" on page 322. However, there are several other types of problem records, such as OBR and MDR records, that do not go to the automation table. You can automate these problem records by generating messages.

The hardware monitor can produce two messages for each record that the alert database receives:

BNJ030I States that the database has received an alert
BNJ146I Contains information about the alert

Automation usually uses the BNJ146I message because it contains more information.

The OPER filter determines which alerts generate messages. However, an alert must pass the ESREC and AREC filters before it can pass the OPER filter and generate the messages.

To automate an alert, you can use the MSGID keyword to select message BNJ146I. You can use several of the fields in BNJ146I as a basis for automation, or you can automate a small subset of the fields sufficient to uniquely identify the alert.

In addition to routing the message for display, you can use the NetView automation table to schedule one or more command procedures to run under one or more NetView tasks when a BNJ146I message arrives. For example, suppose a command procedure is scheduled to run under the task of a monitor operator. That command procedure can receive the BNJ146I message and process it so that a more meaningful message is written to the operator. Another command procedure can automate the recommended actions of the alert.

Automating Status Changes

Status changes tracked by the status monitor or the NMC can trigger automation. By coding SENDMSG statements in the status monitor initialization member DSICNM (CNMS5001), you can cause the NetView program to issue the message CNM094I when specified types of resources change status. For more information about the SENDMSG statement, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

CNM094I indicates a change as shown in Figure 127.

```
CNM094I STATUS UPDATE FOR RESOURCE = resourcename IN NETWORK = netname
FROM DOMAIN = domainname STATUS = status
```

Figure 127. Format for a CNM094I Message

You can automate status changes by using the MSGID keyword to select the CNM094I message.

Putting Your Automation Statements into Effect

To enable automation statements, place all your automation statements into members of the DSIPARM library. The member name can be from 1 to 8 characters long.

If you are making changes to an existing automation table used in production, consider copying the table into a new file or member before making the changes. You can leave the existing automation table in production while you are creating and testing the new one in a separate file.

Before activating an automation table, you can verify that your statements are syntactically correct by issuing the AUTOTBL command with the TEST keyword. You can also use the LISTING keyword to obtain detailed debugging information. For example, if your main automation table is in ATABLE1, you can issue the command shown in Figure 128.

```
AUTOTBL MEMBER=ATABLE1,TEST,LISTING=LIST1
```

Figure 128. Example of Verifying an Automation Table

If there are syntax errors, messages are sent indicating the records in which errors occur and describing the kinds of errors. With this information, you can correct the syntax of your table.

You can test the logic of an automation table using the AUTOTEST command. For testing information, see Chapter 34, “Automation Table Testing,” on page 471.

You can activate the table by entering `AUTOTBL MEMBER=ATABLE1`. To avoid unintended actions caused by a syntax error in the automation table, NetView does not activate a table unless all of the syntax is correct.

To add another DSIPARM member to the list of active automation tables, use the AUTOTBL command and specify where in the list the new member is to be inserted. For example, to insert member DSITBL99 as the second member in the list of active automation table members, enter this command:

```
AUTOTBL MEMBER=DSITBL99 AT=2
```

To verify what automation tables are still active, use the AUTOTBL command with the STATUS keyword as follows:

```
AUTOTBL STATUS
```

The AUTOMAN command provides a full-screen panel interface to enable you to:

- View and manage single or multiple automation tables
- Enable or disable individual automation tables or statements
- View existing tables and their status

For more information, see "Managing Multiple Automation Tables" on page 248.

Correlating Messages and MSUs Using the Correlation Engine

You can convert copies of messages and MSUs to an event format and route them to an automation correlation engine running outside the NetView address space. This correlation engine allows different messages and MSUs to be correlated according to criteria you specify. For example, messages with different identifiers or a message and MSU can be related for automation purposes.

Use the correlation engine simplifies timer and state variable management. You can correlate multiple events over time. For example, automation might be used to check that three different messages are issued within a 10-minute interval. State variables can be used to record that each message was issued and a timer set to verify that all three state variables are set at the end of 10 minutes. By using the correlation engine, you can simplify this coding by using a single correlation rule.

The correlation process is linked to the automation table. It is triggered as a command on an automation table action. The output from the correlation process is one or more of the messages or MSUs that have been correlated being run through the automation table again. You can then use automation table conditions that allow correlated output to be identified and final processing performed using automation table actions.

For information about installing the correlation engine, see *IBM Tivoli NetView for z/OS Installation: Configuring Additional Components*.

Correlation Overview

This is an overview of the correlation process:

- A message or MSU passes through the automation table. You can use an automation table condition (CORRELATED='0') to test whether this message or MSU has gone through correlation.
- If the message or MSU is not already correlated, a copy of the message or MSU is made and an event is constructed from the copy and sent to the correlation engine using the COREVENT pipe stage or the CNMCRMSG command list. A predefined subset of information from the message or MSU is contained in the event sent to the correlation engine. You can add additional information using the PIPE EDIT stage or the CNMCRMSG command list. The original message or MSU can be discarded or continue through the automation table for further processing.
- The event is then processed by the correlation engine. The event is checked against a rules base constructed from an XML document. If the event meets the rule criteria, one or more events (depending on the type of rule and the options specified) are returned to the NetView address space. The rules base must be

coordinated with the automation table to successfully correlate messages and events. The correlation rules determine what information from the message or MSU is contained in the event.

- When the event is returned from the correlation engine, the copy of the original message or alert is retrieved and resubmitted to the automation table. The automation table can detect this resubmission by testing for CORRELATED='1'. Automation table actions can then be driven to handle the correlation detection.
- When correlated output is run through the automation table, you can access the event data that is returned using the COREVTDA PIPE stage. You can check the contents of the event, which might be modified by user-written code during correlation processing. The COREVTDA stage constructs an MLWTO on the secondary output stream with each slot and value representing one line of the MLWTO.

This is a sample automation table entry that detects a command message that is not valid. In this example, the original message is suppressed when its copy is sent to the correlation process.

```
IF MSGID='DSI002I' THEN
  BEGIN;
  IF CORRELATED='0' THEN
    EXEC(CMD('CNMCRMSG INVALID'))
    DISPLAY(N)
    NETLOG(N)
    SYSLOG(N)
    CONTINUE(N);
  IF CORRELATED='1' THEN
    EXEC(ROUTE(NETOP))
    CONTINUE(N);
  END;
```

You can use the CORRFAIL condition to determine if there is a problem with the correlation process. Notice in the example that follows that you test the CORRFAIL condition first. The CORRELATED condition always returns '0' if there is a failure in the correlation process.

```
IF MSGID='DSI002I' THEN
  BEGIN;
  IF CORRFAIL='1' THEN
    EXEC(CMD('MSG NETOP CORRELATION FAILED'))
    CONTINUE(N);
  IF CORRELATED='0' THEN
    EXEC(CMD('CNMCRMSG INVALID'))
    DISPLAY(N)
    NETLOG(N)
    SYSLOG(N)
    CONTINUE(N);
  IF CORRELATED='1' THEN
    EXEC(ROUTE(NETOP))
    CONTINUE(N);
  END;
```

Storage Considerations

Messages and MSUs are processed as follows:

- The NetView program translates the messages and MSUs that the automation table processes into events for the correlation engine. When this translation occurs, the NetView program keeps the original messages or MSUs on a special queue waiting to be matched with the corresponding event when a correlation rule fires.

- Statements in the CNMSTYLE member control how long the NetView program keeps the original message or MSU on the queue. If a message or MSU is discarded because of exceeding the time limit, correlation events can be received by the NetView program without finding a match on the queue. In this case, the NetView program attempts to construct a new message or MSU.

  Only a subset of the content of the original message or MSU is converted into the correlation event format. Because of this, when the event is sent back to the NetView program, the reconstructed message or MSU is not an exact copy of the original. MSUs can lose some information about the forwarding mechanisms that were used to send the MSU to the NetView program, and messages lose **AIFR** and message data block information. A reconstructed message contains the **AIFR** time field and the job name and job number fields. If the restored message or MSU is sufficient for your needs, you can supply a timer value of 0 on the COREVENT pipe stage indicating that the original message or MSU can be discarded. This can be useful in reducing the amount of storage used for correlation.

  You might also specify a value of 0 for other reasons. For example, using event correlation rules you can also modify the event. If you modified the message text in the event and returned the event to the NetView program, the message modifications are not used if the original message is retrieved from the queue.

Correlation Processing

The NetView program takes data from a message or MSU and constructs an event (name/value pair) for the correlation engine. The correlation engine does not work directly with messages or MSUs.

The correlation logic is presented as a series of rules in an XML file. Information from the message or MSU becomes an attribute of the event and can be tested according to rules in the XML file.

Creating Correlation Events Using COREVENT and CNMCRMSG

Messages and alerts are sent to the correlation engine by using the COREVENT pipe stage.

You can use the COREVENT stage to set the eventType attribute for the constructed event and to optionally specify the amount of time to keep the original message or alert on the queue to wait for a matching returned correlation event. For syntax of the COREVENT stage, refer to *IBM Tivoli NetView for z/OS Programming: Pipes* or the online help.

You can add slot and value pairs to the outgoing event by using the NAMEBIND edit order. As an example, this pipe takes the current message and sends it to the correlation engine with a user-defined slot called USERDATA. The eventType attribute is SAMPEVENT.

```
'PIPE (NAME CORSAMP)',
'| SAFE *',           /* copy complete message into pipeline */
'| EDIT',            /* begin edit */
'COPY *',            /* copy complete message to EDIT output*/
'/SAMPLE DATA/' 1,   /* start value one: variable value */
'NAMEBIND /USERDATA/', /* create output line for the new slot */
'| COREVENT SAMPEVENT'/* transfer event to correlation */
```

Before using the COREVENT stage, verify that the DSICORSV task is active. Error messages are written to the secondary stream if communications with DSICORSV fail.

You can also use the CNMCRMSG command list to construct a COREVENT pipe stage. You can use the CNMCRMSG command list to add data. The first parameter identifies the eventType. Subsequent parameters must be in pairs. The first is the name and the second is the value. This command is equivalent to the pipe shown in the example on page 336:

```
CNMCRMSG SAMPEVENT USERDATA SAMPLE DATA
```

Notice that SAMPLE DATA is expressed as multiple tokens. Name/value pairs must be separated by commas.

This example adds an additional slot of USER1 to the event:

```
CNMCRMSG SAMPEVENT USERDATA SAMPLE DATA,USER1 SOMEDATA
```

This example specifies a time-out value of 0 seconds:

```
CNMCRMSG 'SAMPEVENT 0' USERDATA SAMPLE DATA,USER1 SOMEDATA
```

Notice that the optional time interval with the event type parameter must be enclosed in quotation marks.

This example uses mixed case values:

```
CNMCRMSG  SAMPEVENT 'UserData' 'A Mixed Case Value'
```

You can use REXX functions as parameters. For example, to include a user slot of EPNET that contains the entry point network for an MSU, enter:

```
CNMCRMSG SAMPEVENT EPNET HMEPNET()
```

Message and MSU to Event Mapping

Messages and MSUs are converted to correlation engine events prior to being sent to the correlation engine. Most of the event name and value pairs are the same as the default mappings used for converting messages and alerts into Event Integration Facility (EIF) events. For event mappings, see the *IBM Tivoli NetView for z/OS Customization Guide*.

Table 12 lists the default set of attributes for messages.

Table 12. Default message attributes

Attribute Name	Description
HOSTNAME	TCP name of the host where the correlation engine is running
ADAPTER_HOST_SNANODE	NetView NETID and NAU name
ORIGIN	NetView domain that originated the message
DATE	Date of the message
MSGID	Message ID
MSG	First text line of the message
JOBNAME	Job name of the message, if available
SUB_ORIGIN	Job number of the message, if available
SEVERITY	Maps the message to a severity of CRITICAL, FATAL or WARNING. The same mapping is used as when a message is converted to an EIF event.

Table 12. Default message attributes (continued)

Attribute Name	Description
TEXT_LINES	Number of text lines in the message. For MLWTO messages, this field contains the total number of lines in the message. For non-MLWTO messages, this attribute has a value of 1.
MLWTO2..MLWTOn	In the case of an MLWTO, the second and succeeding lines of the message. The second line is in MLWTO2, the third in MLWTO3, and so on. The total number of lines in the message can be found in the TEXT_LINES attribute.
NV_OBJECT	The value "MSG".
EVENTTYPE	User supplied name for the eventType attribute

Table 13 lists the default set of attributes for MSUs.

Table 13. Default MSU attributes

Attribute Name	Description
HOSTNAME	TCP name of the host where the correlation engine is running

ORIGIN	A character string with the name/type hierarchy pairs from the Hierarchy Name List or Hierarchy/Resource List subvectors. The character string contains the hierarchy in this format:
	`resnam1/typ1,resnam2/typ2,resnam/typ3,`
	`resnam4/typ4,resnam5/typ5`
	
	Only the number of pairs in the subvector are used.

ADAPTER_HOST_SNANODE	NetView NETID and NAU name
DATE	Date when the alert was received by the NetView alert adapter in the format: `MMM HH:MM:SS` For example, OCT 10 12:08:30.
MSG	Long error description: long probable cause message that describes the problem. This message is similar to the ALERT DESCRIPTION:PROBABLE CAUSE message displayed on the hardware monitor ALERTS-DYNAMIC panel.
EVENT_TYPE	The event type displayed on the hardware monitor EVENT DETAIL panel (for example, PERMANENT or TEMPORARY). For generic alerts, this is the alert type byte of the generic alert data subvector.
SUB_ORIGIN	A character string with the last pair in the name/type hierarchy pair list from the Hierarchy Name List or Hierarchy/Resource List subvectors. The string is in the form: `resnam_x/typ_x` where *x* is the number of the last pair in the list.

Table 13. Default MSU attributes (continued)

Attribute Name	Description	
ARCH_TYPE	GENERIC_ALERT	NMVT alert major vectors containing a generic alert data subvector
	GENERIC_RESOLUTION	NMVT resolution major vector
	NONGENERIC_ALERT	All other alerts
SEVERITY	Alert type field from the Generic Alert Data subvector or the event type that is used to determine the severity (for example FATAL or CRITICAL). For more information, refer to the *IBM Tivoli NetView for z/OS Customization Guide*.	
EVENT_CORREL	Correlators extracted from MSU correlation subvector 47. These correlators correlate alerts to other alerts. For example, you can have two or more alerts that pertain to the same underlying problem and those alerts are correlated by subvector 47.	
INCIDENT_CORREL	Correlators extracted from Incident Identification subvectors. These correlators correlate alerts to resolutions.	
SELF_DEF_MSG	Text extracted from self-defining text message subvector 31	
BLOCK_ID	For non-generic alerts, the code used to identify the IBM hardware or software associated with the alert.	
ACTION_CODE	For non-generic alerts, code that provides an index to predefined screens. The combination of the block identifier and action code that uniquely identifies the sending product.	
PRODUCT_ID	The hardware or software product set identifier (PSID) of the alert or event sender. This can be 4, 5, 7, or 9 characters. This pertains to all generic alerts and some non-generic alerts.	
ALERT_CDPT	A 2-byte hexadecimal value from the alert description code field of the generic alert data subvector, or the resolution description code field of the resolution data subvector.	
ALERT_ID	For non-generic alerts (including resolutions), an 8-character hexadecimal value assigned by the sender to designate an individual alert condition. The value is 00000000 for resolution alerts.	
MSU	The alert converted into character format. (This attribute is intended for internal use by NetView.)	
DOMID	The NetView domain that first processed the alert. (This attribute is intended for internal use by NetView.)	

Filtering with State Correlation

State correlation monitors information from incoming events and associates this information with user-defined patterns. State correlation analyzes the incoming events for user-defined states to suppress duplicate events, identify event thresholds, and collect or group similar events.

State correlation helps minimize event traffic by identifying similar events and consolidating their information into *summary events* where possible. Those events that cannot be included in a summary are *single events*. The event server then receives these summary events, as well as the single events. The event server does not receive the redundant, individual events for each summary.

Creating Rules

Correlation is achieved with state-based and stateless rules. You specify these rules by using XML syntax, defined by the supplied DTD file, rule.dtd. The default XML file is located in:

```
/var/netview/v6r1/rulefiles/znvrules.xml
```

You can use the CORRSERV REFRESH to override the default file.

Table 14 contains a summary of the XML statements.

Table 14. XML Statements

Type or Element	Attribute	Function
<rule>... </rule>		Specifies the start and end of a rule
	id="identifier"	Associates an identifier for this rule. The identifier can be used to delete, deactivate, or activate this rule when the correlation engine is active
<eventType>... </eventType>		Controls the events that are processed by the rule
	event_identifier	Specifies an event type or class of events
<*correlation_type*>... </*correlation_type*>		Specifies one of these correlation types: **collector** Collects events that meet the specified rule criteria within a specified time interval. All events are then sent off for processing. **duplicate** Suppresses duplicate events that meet the criteria specified in the rule. **match** Checks that an event meets the criteria specified in the rule. **passthrough** Checks for multiple events meeting specified criteria. **ResetOnMatch** Checks for multiple events that do not meet specified criteria. **threshold** Checks for the specified number of occurrences of events meeting rule criteria within a specified time interval .

Table 14. XML Statements (continued)

Type or Element	Attribute	Function
<*correlation_type*> (continued)	randomOrder = "true" | "false"	Used for correlation types passthru and ResetOnMatch: specifies whether ordering of events is significant. If this is set to false, events in the rule have to be received in the same order as expected by the predicates to start the correlation process.
	timeInterval = *miliseconds*	Used for correlation types passthru and ResetOnMatch: specifies the time interval in milliseconds.
	triggerMode = *which_events*	Specifies which events are processed by the action tag: **allEvents** All the events processed by the rule. **firstEvent** The first event to start correlation processing for the rule. **lastEvent** The last event to start correlation processing for the rule. **forwardEvents** For Threshold processing only: Causes the Threshold rule processing to return all events and continue to forward events until the time limit expires. **Note:** How this option is coded can affect how you code the automation table. A single rule firing can result in multiple messages or MSUs being sent through the automation table with the CORRELATED attribute set to '1'.
<cloneable.../>		Causes the correlation engine to separate events according to the specified event attributes and maintain separate correlation processes for those events. This statement cannot be used with match correlation.
	attributeSet = "*event_attributes*"	Specifies the event attributes.
<predicate>... </predicate>		Specifies a Boolean test that evaluates to TRUE or FALSE. Within the <predicate> tag, comparison operations are available to test event attributes. Different types of correlation require different numbers of predicates. Match, Duplicate, Threshold, and Collector correlation require a single predicate. Passthrough and ResetOnMatch correlation require multiple predicates.

Table 14. XML Statements (continued)

Type or Element	Attribute	Function
<action.../>		Specifies what action to take when all the predicates in a rule are TRUE.
	function = *function_type*	Identifies the name of a Java class that is driven to handle the correlation function. The NetView program provides a default action of ReturnToNV. The ReturnToNV action returns the event to the NetView address space, where the original message or MSU is retrieved from a queue and sent to the automation table. If the original message or MSU is not found on the queue (because of a timeout condition), the NetView program uses the returned event to build a new message or MSU and sends it to the automation table. The NetView program provides a superclass FLBCorAction that can be subclassed to allow you to modify or replace this default action. Writing your own action is useful for adding or modifying event attributes.

You define each rule in a state machine. The *state machine* gathers and summarizes information about a particular set of related knowledge. It is composed of states, transitions, summaries, and other characteristics, such as expiration timers and control flags.

These are the state-based rules: collector, duplicate, and threshold. These are all based on state machines. Each state machine looks for a trigger event to start it. Additionally, there is the matching rule, which is a stateless rule.

State-based rules rely on a history of events, whereas the stateless rules operate on a single, current event. These specify the rules:

- Predicates for matching events relevant to that rule
- Actions that run after the rule triggers
- Attributes, such as a threshold limit

Predicates: A predicate in the predicate library consists of a Boolean operator and zero or more arguments. Each argument can be a predicate returning these types:

Table 15. Predicate types and examples

Predicate Type	Example
Boolean value	Equality
Function returning a value	Addition
Event attribute	&hostname
Constant	The string `const01`

Actions: The default action for state correlation is the ReturnToNV action. This action supports a common, optional Boolean attribute named singleInstance. If this attribute is false, the action is not shared among different rules. One instance of the

action is created for every rule that triggers it. This is the default behavior. If the attribute is true, a single instance of the action is created and shared with all rules that trigger it.

Attributes common to all rules: These attributes are common to all rules:

id Specifies the identifier for each rule. It must be unique within the correlation engine where it is registered. Periods are treated as directories. For example, if you have the `id test.threshold`, you cannot have another rule with `test.threshold.1` as the identifier.

eventType
Specifies the set of event classes this rule applies to and optimizes performance. When you omit this parameter, state correlation applies the rule to all event classes.

Matching rules: Matching rules are stateless. They perform passive filtering on the attribute values of an incoming event. A matching rule consists of a single predicate; if the predicate evaluates to true, the trigger actions, which are specified in the rule, run.

In the example that follows, the SAMPLE rule processes an event type of INVALID that was sent by the automation table (see the example shown on page 335). The SAMPLE rule checks the message identifier of the event submitted to correlation and if the message identifier is DSI002I, the event is returned to the NetView program.

```
<rule id="SAMPLE">
 <eventType>INVALID</eventType>   1
 <match>   2
  <predicate>   3
   <![CDATA[
     &MSGID == "DSI002I"
     ]]>
  </predicate>
 </match>
 <action function="ReturnToNV"/>
</rule>
```

The statement explanations for the XML example follow:

1 The eventType is specified as a parameter on the sample CNMCRMSG command list.

2 Defines the type of correlation that the rule represents. In this example, the rule is looking for events that match the specified criteria.

3 The &MSGID attribute is checked for equality with the string DSI002I. When all the predicates in a rule are TRUE, the rule performs the action defined in the rule.

Duplicates rules: The duplicates rule blocks the forwarding of duplicate events within a time interval. It requires these arguments:

- A time interval during which state correlation blocks duplicates of the trigger event. You control the interval with the timeInterval attribute, specified in milliseconds. The trigger event is the first event detected by the duplicates rule and is the only one that is not actually discarded.
- A predicate that is used in detecting the trigger event.

Figure 129 on page 344 shows the state transitions for the duplicate rule:

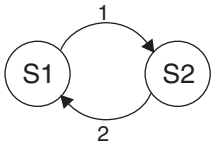


Figure 129. State transitions for the duplicate rule

In Figure 129, state one is the initial state. Transition 1 occurs when there is a match on an incoming event. At that time, state correlation forwards the matching event, and the timer starts. Transition 2 occurs when the time interval expires, and the state machine resets. This is an example of the rule:

```
<!-- Show me only the first error number 10
for my hostname that happens each 10
seconds.
-->
<rule id="test.duplicate" >
    <eventType>NV_Error</eventType>
    <duplicate timeInterval="10000">
      <predicate>
        <![CDATA[
           &msg == "internal error " &&
           &hostname == "hostname1" &&
           &errno = 10
          ]]
      </predicate>
    </duplicate>
    <action function="ReturnToNV"/>
</rule>
```

Threshold rules: The threshold rule looks for *n* occurrences of an event within a time interval. When the threshold is reached, it sends events to the defined actions. The threshold rule requires these parameters:

- One of the sending modes specified by the triggerMode attribute:

  firstEvent
  Sends the first event.

  lastEvent
  Sends the last (*n*th) event.

  allEvents
  Sends all events 1 through *n*, the default mode.

  forwardEvents
  Sends all events after the *n*th until it resets.
- A time interval during which the threshold has to be reached. You control the interval with the timeInterval attribute, specified in milliseconds.
- The time interval mode that indicates if the time interval is fixed. The attribute timeIntervalMode=fixedWindow | slidingWindowinterval controls the mode. The default value is fixedWindow.
- The number of events to match, specified by the thresholdCount attribute.
- A trigger predicate that is used in detecting 1 through *n* events.

Figure 130 on page 345 and Figure 131 on page 345 show the operation of the threshold rule with timeIntervalMode=fixedWindow specified.

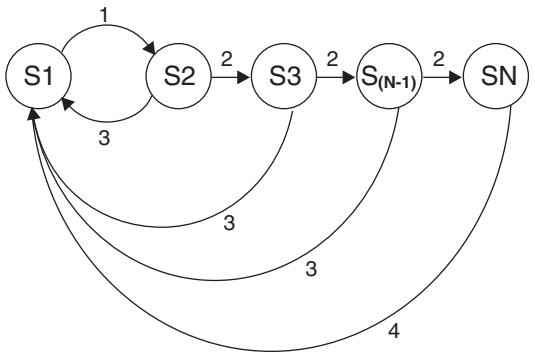


Figure 130. State transitions for the basic threshold rule

Figure 130 shows the state machine for the modes firstEvent, lastEvent, and allEvents. Transition 1 occurs when state correlation detects the trigger event (trigger predicate matches). Transition 2 takes place when an incoming event matches the second predicate. When the time interval expires, transition 3 occurs and the state machine resets. Transition 4 resets the state machine after the threshold is reached. When the state SN is reached, either the first event, the last event, or all *n* events are sent before resetting.

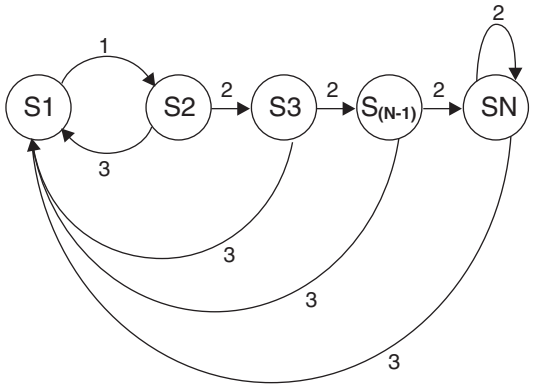


Figure 131. State transitions for the threshold rule using forwardEvents

In forwardEvents mode (Figure 131), the threshold rule operates as in the previous case. Except, it sends all events matching the second predicate after the threshold is reached and until the time interval expires.

When the state machine has timeIntervalMode=slidingWindow specified, the operation of the threshold rule is the same as the fixedWindow time interval. Except that from each node K, there is a transition of 1, 2, .., K-1. This transition accounts for events that are not in the sliding time window. This is an example of the rule:

```
<!--
I'm only interested when at least 5 Node_Down
events for hostnames in my local subnet happen
within 1 minute.
-->
<rule id="test.threshold">
    <eventType>Node_Down</eventType>
    <threshold thresholdCount="5" timeInterval="60000"
    timeIntervalMode="slidingWindow" triggerMode="allEvents">
      <predicate>
```

```
        <![CDATA[
           (&msg == "node down") &&
           (isMemberOf(&hostname, [ 192.168./16 ]))
           ]]
      </predicate>
    </threshold>
    <action function="ReturnToNV"/>
</rule>
```

Threshold rules can also define complex aggregate values, instead of a simple count of events. Use the `aggregate` configuration tag to define this rule. You can construct an aggregate value similar to the definition of a predicate. Threshold rules with aggregate values trigger only when the aggregate value is equal or greater than the thresholdCount value. This is an example of the rule:

```
<!--
If I receive a slot value with a relative percentage between
0 and 1, but I want to check my threshold using the normal
percentage value of 100%, I can define an aggregate of the
slot relative_percentage, by multiplying it by 100 and counting
all percentages until it reaches 100%.
-->
<rule id="test.aggregate_threshold">
    <eventType>Temperature_Variation</eventType>
    <threshold
        thresholdCount="100"
        timeInterval="2000"
        triggerMode="allEvents"
        timeIntervalMode="fixedWindow" >
      <aggregate>
         <![CDATA[
           &relative_percentage * 100
           ]]
      </aggregate>
      <predicate>true</predicate>
    </threshold>
    <action function="ReturnToNV"/>
  </rule>
```

Collector rules: The collector rule gathers events that match the given predicate for a specified period of time. The rule triggers when the timer expires and sends all collected events to the defined actions. The collector rule requires these arguments:

- A time interval during which matching events are collected. You control the interval with the timeInterval attribute, specified in milliseconds.
- A predicate, which is part of filtering the relevant events to add to the collection.

Figure 132 shows the state transitions for the collector rule:

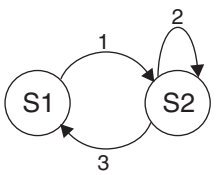


Figure 132. State transitions for the collector rule

In Figure 132, S1 is the initial state. Transition 1 occurs when there is a match on an incoming event; the initial event is not sent but collected. A timer is set to the specified interval. Before the timer expires, all incoming and matching events are

collected (transition 2). Transition 3 occurs when the time interval expires, and the state machine resets. At this time, all collected events are sent. This is an example of the rule:

```
<!--
Collects 10 seconds of Server_Down
events for my database.
-->
<rule id="test.collector">
    <eventType>Server_Down</eventType>
    <collector timeInterval="10000" >
      <predicate>
        <![CDATA[
            &servername == "my_database"
            ]]
      </predicate>
    </collector>
    <action function="ReturnToNV"/>
</rule>
```

Passthru rules: The passthrough rule forwards the trigger event only if a specific set of events arrives within a specified time interval. If the required events arrive before the timer expires (optionally is a specific sequence), the trigger event is forwarded; if they do not arrive, the timer resets and the trigger event is not forwarded.

The passthrough rule requires these arguments:

- A Boolean value (randomOrder) indicating whether the required events can arrive in any order or must arrive in the order specified. If randomOrder is equal to yes, the events can arrive in any order.
- A time interval, after which the state machine resets.
- A trigger predicate, defining the trigger event. This is the event that initializes the state machine and is forwarded if the required subsequent events arrive within the time interval.
- One or more predicates specifying the required subsequent events.

Figure 133 shows the state transitions for the passthrough rule when the required events must arrive in sequence (randomOrder=no).

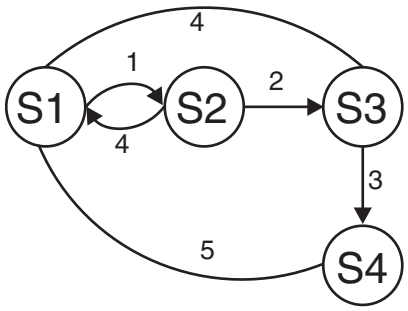


Figure 133. State transitions for the passthrough rule (randomOrder=no)

In Figure 133, S1 is the initial state. Transition 1 occurs when the trigger event is detected; the transition stores the event and starts the timer. Transition 2 occurs when an incoming event matches the first predicate in the required sequence; similarly, transition 3 takes place when an incoming event matches the second predicate in the sequence. When state S4 is reached, the rule forwards the trigger

event and resets to the initial state S1 (transition 5). Transition 4 occurs when the time interval expires, resetting the rule to the initial state without forwarding the trigger event.

Figure 134 shows the state transitions for the passthrough rule when the required events can arrive in any order (randomOrder=yes).

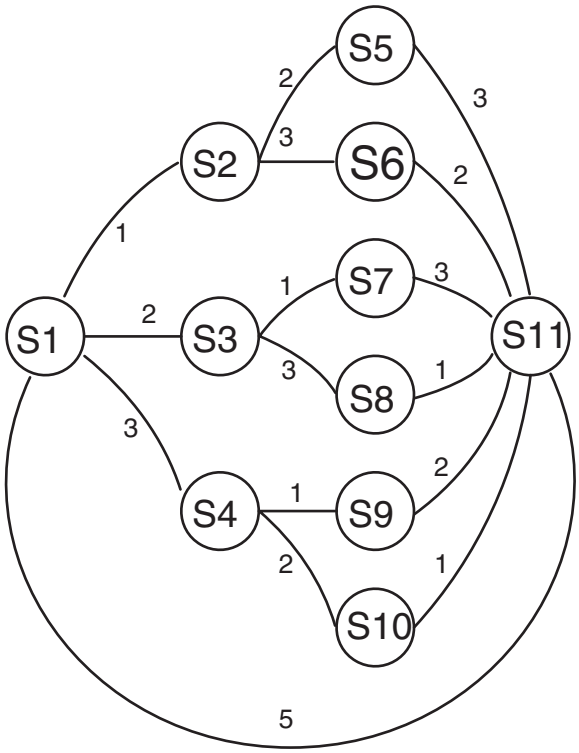


Figure 134. State transitions for the passthrough rule (randomOrder=yes)

In Figure 134, the transitions are the same as in Figure 133 on page 347. In this case, however, the final state is S5, after which the state machine resets.

The Passthrough correlation function ties different events together. In the example that follows, an automation action occurs when two different messages (MSG001 and MSG002) are received within a 30-second timeframe:

```
<rule id="SAMPLE001">
 <eventType>DUMMYMSGS</eventType>
 <passthrough randomOrder="true"  1  timeInterval="30000"  2
                                        triggerMode="allEvents"  3 >
  <predicate>
   <![CDATA[&MSGID == "MSG001" ]]>
  </predicate>
  <predicate>
   <![CDATA[&MSGID == "MSG002" ]]>
  </predicate>
 </passthrough>
 <action function="ReturnToNV"/>  4
</rule>
```

The statement explanations for the XML example follow:

1 The rule is looking for eventTypes of DUMMYMSGS with message identifiers MSG001 and MSG002. The correlation engine starts keeping track of the 30-second time limit when one of these messages are received.

If randomOrder was set to false, an event sequence of MSG002 then MSG001 would not cause the rule to fire.

2 30000 milliseconds represents 30 seconds.

3 When the rule triggers, all events are returned to the NetView program.

You can use triggerMode to specify that only the first or last event is returned to the NetView program. If you specify triggerMode, also select which automation table entries checks for the CORRELATED='1' condition.

4 If the correlation type in this example was ResetOnMatch instead of Passthrough, the ReturnToNV action is invoked if either MSG001 or MSG002 is received, but not both.

Reset on match rules: The reset on match rule forwards the trigger event only if a specific set of events does not arrive within a specified time interval. If the required events arrive before the timer expires (optionally is a specific sequence), the trigger event is not forwarded; if they do not arrive, the timer resets and the trigger event is forwarded.

The reset on match rule requires these arguments:

- A Boolean value (randomOrder) indicating whether the required events can arrive in any order or must arrive in the order specified. If randomOrder is equal to yes, the events can arrive in any order.
- A time interval, after which the state machine resets.
- A trigger predicate, defining the trigger event. This is the event that initializes the state machine and is forwarded if the required subsequent events arrive within the time interval.
- One or more predicates specifying the subsequent events required to prevent forwarding of the trigger event.

Figure 135 shows the state transitions for the reset on match rule when the required events must arrive in sequence (randomOrder=no).

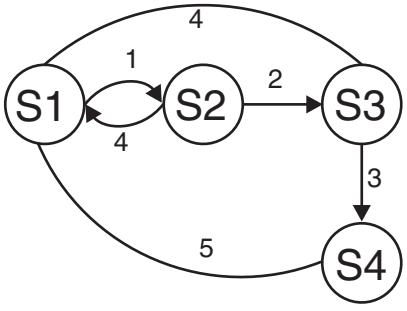


Figure 135. State transitions for the reset on match rule (randomOrder=no)

In Figure 135, S1 is the initial state. Transition 1 occurs when the trigger event is detected; the transition stores the event and starts the timer. Transition 2 occurs when an incoming event matches the first predicate in the required sequence; similarly, transition 3 takes place when an incoming event matches the second predicate in the sequence. When state S4 is reached, the rule resets to the initial state S1 (transition 5). Transition 4 occurs when the time interval expires, causing the rule to forward the trigger event and then reset to the initial state.

Figure 136 shows the state transitions for the match rule when the required events can arrive in any order (randomOrder=yes).

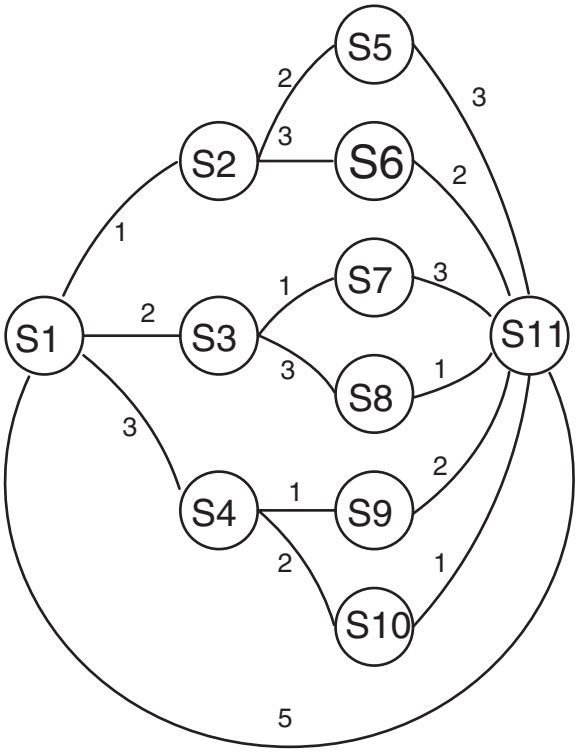


Figure 136. State transitions for the reset on match rule (randomOrder=yes)

In Figure 136, the transitions are the same as in Figure 135 on page 349. In this case, however, the final state is S5, after which the state machine resets without forwarding the trigger event.

Cloning state machines

You can clone any state-based rule by using the **clonable** tag. If state correlation clones a rule when the trigger event occurs, state correlation creates another instance of the rule. This rule is useful for handling multiple event sequences without the need to write many rules.

The cloneable tag causes the correlation engine to separate events according to the specified event attributes and maintain separate correlation processes for those events. For example, to verify that the NetView task initialization message DSI166I is followed by the initialization complete message DSI530I within 10 seconds, you might code a ResetOnMatch rule that fires if message DSI530I is not received. The rule might be coded as follows:

```
<rule id="testcases.taskinitfailure">
 <eventType>INITMSGS</eventType>
 <resetOnMatch randomOrder="false" timeInterval="10000" triggerMode="firstEvent">
  <cloneable attributeSet="TASKN"/>  1
  <predicate>
   <![CDATA[&MSGID == "DSI166I" ]]> 2
  </predicate>
  <predicate>
   <![CDATA[&MSGID == "DSI530I" ]]>
```

```
  </predicate>
 </resetOnMatch>
 <action function="ReturnToNV"/> 3
</rule>
```

The statement explanations for the XML example follow:

1 This rule assumes that all events of eventType INITMSGS have an attribute named TASKN that contains the task name.

2 As each DSI166I is received, a separate correlation process starts a 10-second timer to wait for the receipt of an INITMSGS event with the same TASKN value and a MSGID value of DSI530I. An event with a MSGID of DSI530I but a different TASKN value does not satisfy the predicate.

3 If both messages are not received within 10 seconds, the first event from message DSI166I is returned to the NetView program.

The events can be built out of the automation table with this coding:

```
IF MSGID='DSI166I' & TEXT= 'DSI166I' TASKNAME 'IS ACTIVATED' . THEN
  BEGIN;
   IF CORRELATED='0' THEN
    EXEC(CMD('CNMCRMSG INITMSGS TASKN ' TASKNAME))
    DISPLAY(Y) SYSLOG(Y) NETLOG(Y) CONTINUE(N);
   IF CORRELATED='1' THEN
     EXEC(CMD('PIPE LIT /SAMP003 ' TASKNAME 'FAILED TO COMPLETE ITS
INITIALIZATION WITHIN SPECIFIED TIME PERIOD /|COLOR RED|CONS'))
     DISPLAY(N) SYSLOG(N) NETLOG(N) CONTINUE(N);
  END;

IF MSGID='DSI530I' & TEXT=. '''' TASKNAME ''' :' .  THEN
  BEGIN;
   IF CORRELATED='0' THEN
      EXEC(CMD('CNMCRMSG INITMSGS TASKN ' TASKNAME))
      DISPLAY(Y) SYSLOG(Y) NETLOG(Y) CONTINUE(N);
  END;
```

Writing custom actions

In addition to the standard actions, your rules can also use custom actions you write using Java code. By writing custom actions, you can perform more sophisticated event processing, including modification of event attributes.

Each action is implemented as a Java class. When the state correlation engine starts, it creates instances of all of the action classes required by the rules, using any parameters specified by the rules. If an action is declared as shared, only a single instance of each action is created, and this same instance is used by all rules that call that action. If an action is not shared, a separate instance is created for each rule that uses a particular action class. (If you need to call an action using different parameters in different rules, the action cannot be shared.)

Event objects: The state correlation engine works with Java objects that represent events. When an event arrives, a Java object is created containing all of the attribute data from the event. This object, an instance of class com.tivoli.zce.engine.Event, is sent to the state correlation engine. If persistence is enabled, the state correlation engine then writes a record of the event to a persistent store. The persistent store is a recovery mechanism used to ensure that no events are lost if the gateway shuts down while events are processed by the state correlation engine. When the gateway is restarted, any unsent events recorded in the persistent store are immediately sent to the gateway.

As initially created, the Java Event object contains two copies of the event:

- A working copy, which can be directly accessed using the methods of the event object. Actions can use this working copy to make changes to event attributes during processing.
- An internal snapshot of the state of the event as it was received by the state correlation engine. This field is initially equivalent to the record written to the persistent store, and it is not dynamically updated to match changes to the working copy.

The event object is then processed by the state correlation rules and any actions called by those rules. This processing might include changing the event attribute values or creating new events.

Action structure: An action is implemented as a Java class and must be included in the com.tivoli.zce.actions.libs package to be found by the state correlation engine. This class must implement the necessary interfaces to support event handling within state correlation rules. An action class must extend the com.tivoli.zce.actions.DefaultActionHandler class, which provides the necessary interfaces and some default behavior. The FLBCorAction superclass that is supplied with the NetView program does this and can be used as the superclass for user-written actions. An action must implement these three methods:

- `processEvent()`. This method is called by a rule, or by the preceding action within a rule, and takes as its parameter a single Event object. This method is used in cases where the action is processing a single event.
- `processEvents()`. This method is similar to the `processEvent()` method, but takes as its parameter a single EventList object, which contains an array of multiple events. Both methods must be implemented, because an action can be called with either a single event or an event list. The `processEvents()` method can parse the list and then call the `processEvent()` method for each one.
- `doParse()`. This method is called by the state correlation engine during initialization, after the action class is instantiated. The `doParse()` method parses the parameters specified in the XML rule that calls the action; these parameters govern the behavior of the action instance for all rules that use it. (Note that if an action is not shared, multiple instances might be created with different parameters.)

When an event is received by the action (through either the `processEvent()` method or the `processEvents()` method), the action can call the methods of the event object to retrieve or change the event attribute data. (See "Working with events" on page 353 for more information.) Finally, assuming the event is not discarded, the action must send the event to the next step in processing. This can either be to another action, or if the current action is the last one within a rule, back to the state correlation engine. The basic method for doing this is the `forward()` method, but under some circumstances you might need to use a different method.

Figure 137 on page 353 summarizes the general structure of an action class.

```
package com.tivoli.zce.action.libs;

import com.tivoli.zce.IRule;
import com.tivoli.zce.ParserException;
import com.tivoli.zce.CorrelatorException;
import com.tivoli.zce.engine.EventList;
import com.tivoli.zce.engine.Event;
import com.tivoli.zce.actions.DefaultActionHandler;

public class MyAction extends DefaultActionHandler implements ITecEventAttributes
{

  public void processEvents(EventList eventList) throws Exception
  {
    // code to process multiple events
    forward(eventList);
  }

  public void processEvent(Event event) throws Exception
  {
    // code to process single event
    forward(event);
  }

  public Object doParse(IRule rule, String params) throws ParserException
  {
    // method to parse parameters after instantiation
  }
}
```

Figure 137. Structure of an action

Working with events: An event object (an instance of com.tivoli.zce.engine.Event) contains the event attribute information as a set of name-value pairs stored in a hash table. The Event class provides methods you can use to work with these attributes. The methods include:

hasAttribute()
: The `hasAttribute()` method takes a single string as a parameter and returns a Boolean value indicating whether the event contains an attribute with the specified name. For example, `event.hasAttribute("HOSTNAME")` returns `true` if *event* has an attribute called HOSTNAME. (Note that attribute names are case-sensitive.)

getString()
: The `getString()` method takes a single string as a parameter and returns a string containing the value of the attribute with the specified name. For example, `event.getString("SEVERITY")` returns the current value of the SEVERITY attribute of *event*.

putItem()
: The `putItem()` method takes as its parameters a string key and a value. This method sets the value of the attribute whose name is equal to the specified key string. For example, `event.putItem("ORIGIN","SCE")` sets the value of the ORIGIN attribute to the string "SCE". If the specified key does not match an existing attribute, a new attribute is added. If the attribute value contains spaces or special characters, enclose it within nested single quotes to ensure correct parsing by the event server.

Chapter 23. Establishing Coordinated Automation

You can automate many operations that are more complex than scheduling commands or responding to messages and MSUs. For example, you can automate these operator tasks:

- Initializing the products in your system or network
- Monitoring the products
- Initiating recovery actions when necessary
- Shutting down products in an orderly way when you want them deactivated.

Advanced automation requires you to coordinate actions among many command procedures and other automation facilities. For example, the automation table can receive information in the form of messages and MSUs and pass the information to monitoring command procedures. The monitoring procedures in turn can initiate recovery whenever necessary. In addition, to ensure the availability of the automation, have your automation applications monitor each other.

Because of the coordination required among your automation applications for advanced automation, you must thoroughly design your automation project before you begin implementation. See Chapter 4, “Designing an Automation Project,” on page 51 for automation design guidelines.

You can achieve coordinated automation by using NetView global variables, the Resource Object Data Manager (RODM), or both. This chapter explains establishing coordinated automation with NetView global variables. See Chapter 28, “Automation Using the Resource Object Data Manager,” on page 407 for a discussion about establishing coordinated automation with RODM.

Before establishing coordinated automation using NetView global variables, examine the advanced automation sample set that NetView provides. The sample set automates initialization, monitoring, recovery, and shutdown for several MVS products and components. The sample set also uses internal monitoring to ensure that its own autotasks remain active and functioning. By examining the sample set, you can see how global-variable naming conventions and other common protocols ensure effective communication among command procedures.

The State-Variable Technique

One way to structure your coordinated automation is to build it on a system of state variables. You can view the operation of a system or network as a process of monitoring the state of each system or network resource. Resources change state when a problem occurs or when you take action to resolve a problem. The shutdown of an application program, the activation of a network resource, or the logon of an operator all represent transitions between states.

To monitor the system or network, you watch for messages and MSUs that indicate the state of each element. You also keep track of the desired state of the element and attempt corrective action if the state does not match the desired state.

In the automated environment, your automation applications can keep track of current and target states. For example, you can assign two global variables for each component or resource that you want to automate. One can hold the current state, and the other can hold the target state. When the automation table receives a

message or an MSU that indicates a change of state, you can update the current-state variable accordingly. Target states can be based on conditions or policy statements that you establish beforehand. An example is a policy that the VTAM program must be active between 6:00 a.m. and midnight. You can also provide operator interfaces that allow you to update the target-state variables directly.

You can also track information. For example, you can use a variable to indicate the automated action being taken for each resource. By keeping track of the action being taken, you can avoid attempting corrective action a second time for a problem before the first attempt is completed. The advanced automation sample set records the action being taken in the action-state variable. If the sample set attempts to restart TSO after a failure, for example, it updates the status of TSO from DOWN to STARTING.

You might also use variables that store the number of users logged on to an application or that store policy information, such as these:

- Product dependencies (for example, do not attempt to start TSO unless VTAM is active)
- Timing information, such as when a product must be activated and when it must be shut down again
- Whether automation is responsible for keeping a resource in its desired state or just for monitoring the resource; you can then turn off automation if you want to operate a resource manually

Figure 138 on page 357 illustrates a possible structure for coordinated automation using state variables.

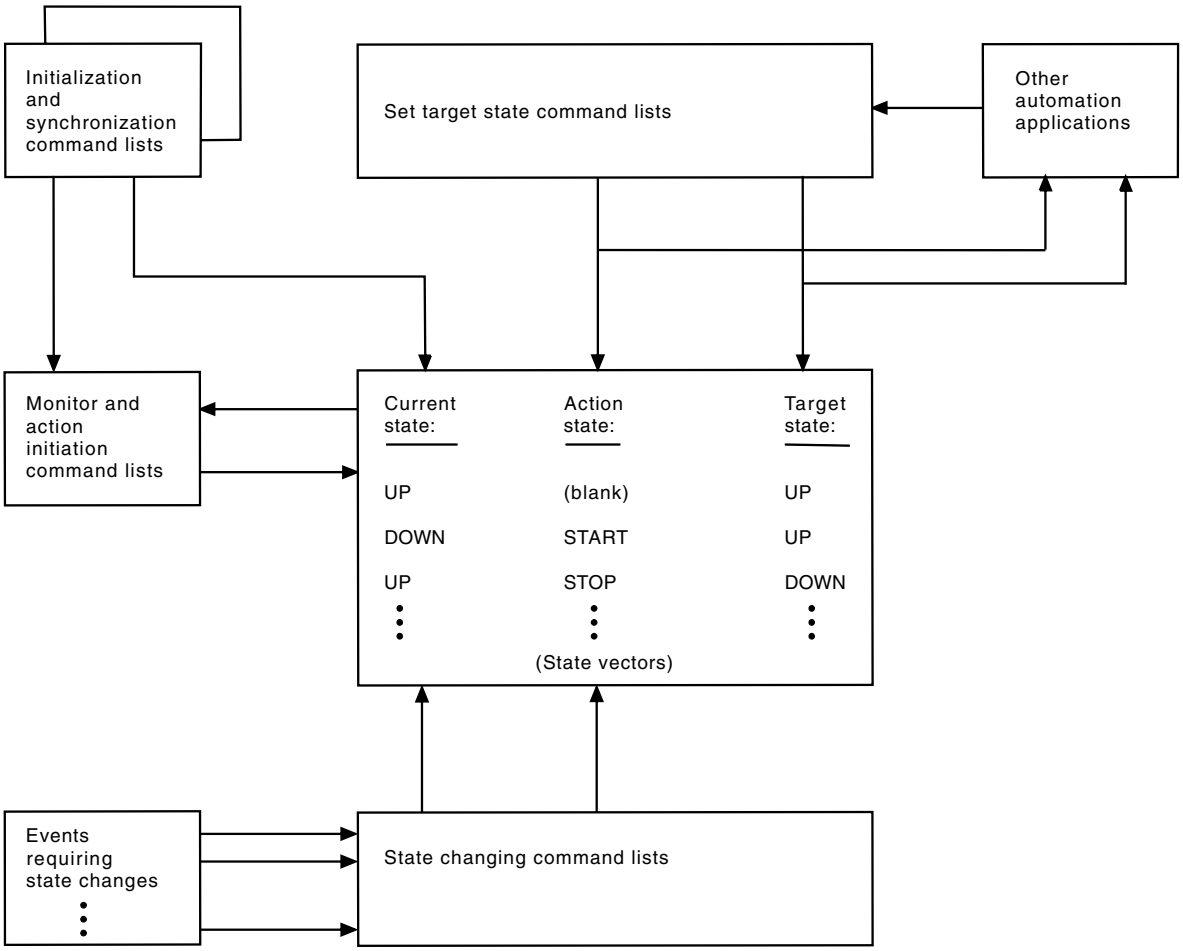


Figure 138. Coordinated Automation Using State Variables

Automating Initialization, Monitoring, Recovery, and Shutdown

Important operating tasks that you can perform with coordinated automation include the initialization, monitoring, recovery, and shutdown of system and network resources and components.

Initialization Starting or activating a product or component

Monitoring Watching the system to keep track of the state of each product or component

Recovery Taking corrective action when monitoring reveals a problem or a discrepancy between the actual and desired states

Shutdown Stopping a product or component in an orderly fashion

These sections describe how you can automate initialization, monitoring, shutdown, and recovery. For an illustration of how you can automate these four tasks, see the advanced automation sample set, which is described in "Using the Advanced Automation Sample Set" on page 585.

Automating Initialization

To accomplish automated initialization of a product or component, you can usually use a process modeled on the manual process that operators use. Operators issue certain commands and await certain messages that indicate successful initialization. If the messages are not received within the expected time, a problem is indicated and operators can take recovery action.

To accomplish automated initialization of an entire system, you can begin by activating the operating system manually. Alternatively, you can activate the operating system remotely; see "Establishing Remote Operation" on page 17 for information. The operating system can automatically activate NetView, and the NetView program can automate the initialization of other products and components. You can start NetView automatically by placing the start command for NetView in the COMMND*xx* member of SYS1.PARMLIB.

Use the NetView initial command list to call your automation procedures and begin initializing all remaining products and components. Wait for the successful initialization of one product before initializing another because of product dependencies. For example, VTAM must be active before you can initialize TSO.

Automating Monitoring

Automation can use both passive monitoring and proactive monitoring, described as follows:

Passive monitoring
: Watching for certain messages and MSUs and acting when they are received

Proactive monitoring
: Issuing query commands to determine status

Passive Monitoring

In an automated environment, operators no longer need to monitor all messages and MSUs that indicate the status of system and network components. The automation table performs passive monitoring. Automated actions that can be taken upon receipt of a message or an MSU include updating state variables so that the NetView program has an accurate record of the state of each component.

Proactive Monitoring

Proactive monitoring involves issuing commands that query the system or network to obtain status information. You can issue query commands at regular, timed intervals so that you obtain updated information. NetView timer commands, such as the EVERY command, can schedule your query commands for you.

The shorter the interval you use, the more up-to-date your status information is and the faster you can respond to failures. However, you can place an unnecessary burden on the system by issuing queries too frequently.

Also, use proactive monitoring to monitor the status of your automation application. For example, you can monitor autotasks by sending test commands or test messages to them at regular intervals. If the autotasks are set up to issue a specific response to a test message with the help of the automation table, failure to send the correct response can indicate an autotask failure. The advanced automation sample set illustrates this technique.

You can monitor the automation table by having an autotask periodically issue the AUTOTBL STATUS command and wait for the results. In this way, you can ensure that the correct automation table is running at all times.

As with passive monitoring, you can place the information you gather with proactive monitoring in state variables. You can provide this information to your entire automation application.

Combining Active and Passive Monitoring

Passive monitoring usually provides a speed advantage, because automation does not wait until the next scheduled query command to detect a problem. However, proactive monitoring might provide a reliability advantage, because a component that changes state without issuing the messages you expect can still be accurately observed with proactive monitoring. By combining active and passive monitoring, you can gain the advantages of both methods.

Automating Recovery

When passive or proactive monitoring detects a problem, such as a mismatch between an actual state and a desired state, you can initiate recovery. Automated recovery is similar to an operator's attempting to restart a product after receiving a console abend message.

The recovery process for a failing component can be the same as the initialization process for that component and can use the same command procedure. However, automation might need to first answer a failure message, investigate the cause of a problem, or ensure that a failing component is ready for reactivation before restarting the component.

Automating Shutdown

As with automatic initialization, automatic shutdown typically follows a process similar to the manual process. To shut down a specific product, issue the shutdown command for the product and await messages indicating successful completion.

When shutting down an entire system, you can generally shut down the products in the reverse of the order in which you initialized them. Do not shut down a product until all other products that depend on it have first completed their shutdowns.

Chapter 24. Enhancing the Operator Interface

An important part of implementing your automation plan is to create an operator interface that is appropriate to your evolving environment. A good operator interface presents operators with the information they need to monitor the environment, examine the state of each resource, and verify that automation is functioning correctly.

In addition, you must provide for ***exception notification***, which is the process of informing operators when automation routines encounter problems or events that you have not yet automated. With exception notification, you focus operator attention on any problems that still require manual intervention.

In an automated environment, you can present information to operators in these forms:

- As messages with the command facility
- As status information with the status monitor and, the NetView management console (NMC)
- As alerts with the hardware monitor or NMC
- As full-screen displays with VIEW and help panels
- As e-mail or alphanumeric pages

Displaying Messages

You can display information to operators in the form of messages on the NetView command facility. NetView messages and network messages can continue to be displayed on the command facility, just as in an unautomated environment. In addition, if you have consolidated your consoles, you can display system messages from the operating system, subsystems, and applications.

Automation must reduce the number of messages displayed. Chapter 19, "Suppressing Messages and Filtering Alerts," on page 299 and Chapter 22, "Automating Messages and Management Services Units (MSUs)," on page 317 describe ways to reduce the flow of messages.

However, messages are still useful in the automated environment, and you can have your automation procedures issue messages to the command facility. When testing automation, for example, you can have automation procedures issue messages that inform operators of the actions being taken. After testing is complete and your automation is working smoothly, you can reduce your use of this type of message. Another use of messages is to inform an operator when automation procedures encounter problems that require manual intervention.

Command lists can issue messages with the MSG command. Refer to the NetView online help for a description of the MSG command.

Displaying Status Information

NetView provides two ways to display the status information of a network:

- The status monitor (text form)
- The NetView management console (graphical display)

These facilities track the states of network resources and display them to your operators in an organized, hierarchical fashion. They use color changes to draw attention to network problems.

You can use status displays to complement your automation. While automation is handling individual messages, alerts, and MSUs, an operator can quickly view the status of the network and confirm that automation is keeping each resource in its correct state.

Tracking Status with the Status Monitor

The status monitor displays status information in text format.

Besides displaying information to operators, the status monitor can automatically reactivate failing resources, except for applications and cross-domain resources. The status monitor intercepts status information from the VTAM program that indicates an inactive resource and starts attempting reactivation at 1-minute intervals until the resource returns to active status. To enable this function, you must have an O MONIT statement in DSICNM. You can turn automatic reactivation on and off for a specific resource or for all resources with the MONON and MONOFF commands. You can also use STATOPT statements in VTAMLST members to choose which resources the status monitor attempts to reactivate.

For information about O MONIT and STATOPT statements, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Tracking Status with the NetView Management Console Display

The NetView management console displays status information in graphical format, drawing pictures of your network on the screen. You can customize the pictures to display the information your operators require. Several operators can monitor parts of the network, each from a different workstation, or a single operator can monitor your entire enterprise.

NetView graphical displays show information to operators through a workstation connected to an MVS system. You can graphically monitor status information about the operating systems by first forwarding the information to an MVS system. See Chapter 26, “Centralized Operations,” on page 373 for a discussion of forwarding.

For more information about graphical status displays, see the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

Monitoring Alerts with the Hardware Monitor

You can also use the hardware monitor to monitor your system and network. In an automated environment, you can use the hardware monitor for both hardware and software. The hardware monitor can perform exception notification for you by displaying alerts to operators when a problem occurs that automation alone cannot handle.

The hardware monitor allows you to display more information about a problem than a message gives. This additional information can include a problem description, a list of probable causes of the problem, and a list of recommended actions. The hardware monitor also provides:

- A history of reported problems
- Filtering capabilities
- A problem management interface to the Information/Management program
- Recording capabilities for the system management facilities (SMF) or another external log

Therefore, you might want to convert messages that require operator intervention into alerts and display them on the hardware monitor. But do not convert a message into an alert if you can suppress or automatically respond to that message instead. The aim of automation is to reduce the number of event notifications that operators must view.

You can use these facilities to send alerts to the hardware monitor:
- The program-to-program interface
- The GENALERT command
- The MS transport

To avoid issuing alerts too quickly and depleting storage, ensure that any automation that creates alerts does not run at a higher priority than the DSICRTR, BNJDSERV, and LUC (domain ID followed by LUC) tasks.

Note: Alerts sent to the hardware monitor through the program-to-program interface or over the MS transport go through the NetView automation table, as do alerts created by GENALERT. If you use an alert to initiate automation and automation can create another alert, be careful to avoid an endless loop. See "NetView Program Hardware-Monitor Data and MSU Routing" on page 96 for complete routing information.

Sending Alerts with the Program-to-Program Interface

To send alerts to NetView from another application program running in the same system, use the program-to-program interface. The program-to-program interface is also an option for generating alerts from within NetView.

With the program-to-program interface, application programs can send generic alerts to each other and to the hardware monitor in NMVT or CP-MSU format. When an application program detects a problem, it can send an alert to the NetView program. You create the alert by calling the CNMCNETV module in the NetView subsystem and passing the alert information to NetView. NetView treats the alert as an unsolicited record. If the alert passes the appropriate hardware monitor filters, it becomes a hardware monitor alert and can be displayed to operators.

You can also create an alert in a REXX command and use the PPI PIPE stage to send it to the hardware monitor.

For information about creating software alerts with the program-to-program interface, refer to the *IBM Tivoli NetView for z/OS Application Programmer's Guide*.

Sending Alerts with the GENALERT Command

You can use the GENALERT command if you want to create alerts from within NetView. For example, the automation table can issue the GENALERT command when it receives a message that requires operator attention. Also, a command procedure can issue the GENALERT command if it encounters a problem that requires operator attention.

See the NetView online help for more information about the GENALERT command. For more information about the code points and code point formats that can be used by the GENALERT command, see the generic alert code points appendix in the *IBM Tivoli NetView for z/OS Messages and Codes Volume 2 (DUI-IHS)*.

Sending Alerts with the MS Transport

You can use the MS transport to send alerts to the hardware monitor:

- From within NetView
- From another application on the system
- From another system

Send your alerts to the NetView program's ALERT-NETOP MS application in MDS-MUs. Each MDS-MU must contain a CP-MSU with one or more alert major vectors.

Monitoring Alerts with the NMC

Operators using the NMC can request alert history to view alerts generated by AON. AON sets the `Automation in Progress` status so that operators can see that automation is attempting to recover the failed resource. Failed resources that cannot be recovered appear in the Operator Intervention view (OIV).

For more information, refer to the *IBM Tivoli NetView for z/OS User's Guide: NetView Management Console*.

Creating Full-Screen Panels

NetView lets you create your own full-screen panels with extensive color and highlighting options. You can create full-screen panels to complement other operator interfaces or to replace them, both for displaying the states of network resources and for exception notification.

You can create full-screen panels with a standard editor such as ISPF. You display the panels from a command procedure by issuing the NetView VIEW command.

You can update panels dynamically, so that operators can monitor changing information. You can specify locations on the panel for accepting operator input. The input is sent back to the calling command procedure, enabling the automation command procedure to interact with the operator through a full-screen interface. Panels can display the values of NetView global variables and can enable an operator to change the values of the variables. A calling command procedure can also be informed if an operator presses a special key, such as ENTER or a PF key. You can establish chains of panels, enabling operators to press a key to move from one panel to another.

The advanced automation sample set demonstrates how you can use the VIEW command with automation. The sample set uses full-screen panels to display the current status of each program or component that NetView is automating. The sample set stores each status in a global variable and displays it to operators in an appropriate color. For example, the line on the panel for CICS turns bright red if CICS fails. When the status changes, the sample set automatically updates the screen of any operator who is viewing the status panel, showing the latest status and the time of the change.

Operators can access additional panels for more information about a specific program or component that you are automating. For more information about the operator interface in the sample set, refer to "Automation Display Panels" on page 598. For additional examples of the use of the VIEW command, see the NetView command lists BROWSE, TUTOR, and DISG in the NetView online help.

The HELP command also uses the VIEW command, enabling you to create help panels of your own or to modify existing help panels. NetView offers an extensive set of online help panels and online message help for network management. By modifying these panels for your automated environment, you can give operators the help they need to solve problems and to perform standardized procedures. You can also introduce new help panels to assist operators in using your automation command lists, command processors, and operator interfaces.

For more information about using full-screen panels, including help panels, see the *IBM Tivoli NetView for z/OS Customization Guide*.

Sending Email or Alphanumeric Pages

You can define which personnel must be contacted for a problem, when they must be contacted, and how they must be contacted by using the INFORM policy member. By default the INFORM policy provides support for mail and alphanumeric pagers. For more information about the inform policy, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Part 6. MultiSystem Automation

Chapter 25. Propagating Automation to Other NetView Systems

The first step toward automating your entire data-processing enterprise is to ensure that you are doing as much local automation as possible on each NetView system. Therefore, if you have begun with single-system automation on one system or on a few test systems, propagate that automation onto all of your NetView systems. Copy your automation routines and tailor the routines to the new systems. In this process, it is important to automate all of your systems consistently to keep maintenance as simple as possible.

Propagation also involves preparing for exception forwarding and the use of focal points. When you connect your systems and forward exceptions, the automation of one system can affect the automation of others. Therefore, it is important to synchronize your automation and to determine the relationship that each system has with its focal point. This chapter describes guidelines for effectively propagating automation.

Automating Close to the Source

In a multisystem environment, automate as many tasks as possible on the distributed systems and forward only those things that cannot be handled at the distributed systems to the focal point. At the distributed system, if the function of the operating system facility (the message processing facility (MPF)) enables you to accomplish what you want without using the automation table, use the appropriate operating system function; otherwise, use NetView.

Distinguishing between Automation Procedures

Categorize automation procedures into focal point control procedures and single system automation procedures. Focal point control procedures are those performed by the focal point system, or those performed by distributed systems on behalf of the focal point, such as those that periodically send updated information to the focal point. Single-system automation procedures are the intrasystem automation procedures used on the individual systems that do not require communication with the system designated as the focal point.

Defining Responsibilities

Establish clear boundaries between responsibilities of the focal point and those of the distributed systems to avoid duplication of work. For example, if each distributed system has an autotask that periodically checks the status of the automation table with AUTOTBL STATUS, it is not necessary for the focal point to monitor the automation tables of the distributed systems.

Defining Autotasks Consistently

Use consistent operator definitions for autotasks, profiles, and operator passwords or password phrases across systems. Consistent definitions reduce the effort required to make changes among systems. For more information, refer to *IBM Tivoli NetView for z/OS Security Reference*.

If you have multiple NetView programs in a single MVS system, or if you are using a sysplex configuration, the extended multiple console support (EMCS) consoles obtained for operators (and autotasks) must have unique console names. Use the NetView GETCONID or SETCONID command in the initial command list for each operator to resolve any conflicts. Refer to the NetView online help for a description of the GETCONID or SETCONID command.

Developing Generic Automation Command Procedures

To minimize the work required for development, maintenance, and synchronization of automation procedures for multiple systems, write generic procedures to function equivalently in all applicable systems. These procedures should also function on the focal point system when the system is not performing focal point functions.

To simplify migration, use global variables for system and resource names rather than hard-coding them into command procedures. By keeping the definition setup for the global variable in only a few procedures, you can migrate the same set of automated procedures to multiple systems and customize only a few procedures on each system. This technique is used in the advanced automation sample set (see "Using the Advanced Automation Sample Set" on page 585).

Developing a Portable Automation Table

From a maintenance perspective, it is best to have one automation table common to several systems. However, an automation table must be tailored to different needs; therefore, the table might be large. Also, a certain message can be used in different ways in different environments. Thus, you might need a separate automation table for each system. If you have multiple automation tables, ensure that updates in the systems are coordinated.

The %INCLUDE statement enables you to keep sections of an automation table in separate data set members. For example, you could keep the portion of your automation table that is common to all systems in one data set member and the portion that is specific to each system in a second.

You can also use multiple automation table members and only enable the appropriate members on a system by system basis. This has the advantage of allowing you to enable and disable portions of your automation logic to reflect workload movement from one system to another.

The SYN statement also facilitates maintenance by enabling you to define synonyms for those parts of the automation table that must vary from system to system. You can then adapt the table to a new system by changing the values of your synonyms.

Including Forwarding

Message, alert, and command routing are key to managing the delegation of automation responsibilities across multiple focal point and distributed systems. Use routing to direct where messages and alerts are to be processed and where commands are to be run. As with autotask IDs and command procedures, consistency in your approach simplifies automation maintenance. See "NetView Program Message Routing" on page 84 and Chapter 26, "Centralized Operations," on page 373 for more information.

Installing and Testing Before Distribution

When you are developing generic intrasystem automation procedures, it is a good idea to install the procedures and test them on one system before distributing the function to all systems. In this way, you can work out any generic problems in an isolated environment. When you are satisfied that the procedures work in one environment, you can distribute the procedures, customizing global variables or control files on each system as appropriate, and test again throughout the enterprise. By testing first on one isolated system, you can reduce the number of corrections that must be made when testing throughout the enterprise.

Logging Intrasystem Automation

Intrasystem automation that occurs in each system should be logged in the local network log. Messages and alerts that are forwarded to a focal point should be logged at both the distributed system and the focal point for two reasons:

- When an alert or message is forwarded to a focal point, all of the pertinent information might not get forwarded. An operator at the focal point might have to go to the distributed system for additional information.
- If a line failure occurs while a message or alert is in transit, the information is lost. In that case, the focal point operator must browse the distributed system's log to gather information.

It is a good idea to have all procedures driven by automation identified in some way within the log. The sample set for automation has each command list write a message to the log that is preceded by a less-than sign (<). A quick glance at the log lets you know whether automation has played an active role in activities occurring within NetView. These indicators provide an audit trail for automation, which provides a basis for measurement against the quantified objectives that you developed as part of your automation plan (see Chapter 3, “Defining an Automation Project,” on page 41). It also can assist in problem determination in the event of an automation failure.

See Chapter 35, “Logging,” on page 487 for more information about logging.

Chapter 26. Centralized Operations

With automation, you can centralize operations so that you manage all systems, networks, and data centers from a single system or a few centralized systems. Often, you can run many of your systems unattended and consolidate your operation staff at a single location. This process has some of the same objectives as single-system console consolidation and can further reduce the number of consoles you monitor.

Before centralizing operations, use local automation on each system to perform as many operation tasks as possible. Part 5, "Single-System Automation," on page 287 describes the techniques for local automation on each system. Do not forward problem notifications for a problem that you can solve locally. However, you must forward the following types of information to the central system:

- Forward information about the *state* of each local system, including the system portion of the network, so that operators and automation on the central system have an accurate, up-to-date description of every resource and application.
- Forward notifications about *exceptions* or problems that local automation alone cannot solve. These problems can be solved by automation on the central system or by operators logged on to that system.

A central system that receives information from distributed systems is called a *focal point*. For design guidelines about choosing a suitable focal point, refer to Chapter 4, "Designing an Automation Project," on page 51.

This chapter describes how to transmit information between a distributed NetView system and a focal-point NetView system.

Data Transports

To help you centralize operations, NetView provides different data transport methods. The transport methods are: LU 6.2, LUC, and OST-NNT. These transports are used to transfer data between NetView programs that reside in different nodes. LU 6.2 transports are also used to transfer data between NetView and non-NetView products, such as the AS/400®. When you centralize operations between NetView nodes, one or more of these transports are used to move data between the nodes.

LU 6.2 Transports

NetView supports two LU 6.2 transports, which use different versions of the SNA LU 6.2 protocols:

- The management services (MS) transport is for low-volume transmissions that require high reliability, such as sending alerts.
- The high-performance option of the MS transport is for large-volume transmissions that require better performance.

The NetView LU 6.2 transports are based on the MULTIPLE_DOMAIN_SUPPORT Function Set described in *SNA Management Services Reference*. The LU 6.2 transports are used by Management Services (MS) applications to send and receive data.

MS applications can be architectural applications, such as the applications that are provided with the NetView program or user-defined applications. An application is created when it *registers* with the LU 6.2 transports. After registering, an application can send data to other registered applications and receive data from them. For example, you can create a user-defined application that can send alerts to the NetView ALERT-NETOP application. You can display the applications known to the NetView program (both the applications that are provided with the NetView program and user-defined applications) by using the REGISTER QUERY command, shown in Figure 139:

```
* CNM02    REGISTER QUERY
' CNM02
DWO468I TYPE APPL        COMMAND  TASK     FPCAT       FOCALPT LOGMODE  NOTIFY
DWO469I MS   HMON_DST    BNJ62DST BNJDSERV --NONE--    NO      SNASVCMG NONE
DWO469I MS   HMON_OST    BNJ62OST BNJDSERV --NONE--    NO      SNASVCMG NONE
DWO469I MS   LINKSERV    BNJNETOP BNJDSERV --NONE--    YES     SNASVCMG NONE
DWO469I MS   ALERT       BNJNETOP BNJDSERV ALERT       YES     SNASVCMG NONE
DWO469I MS   OPS_MGMT    DSIOURCP DSI6DST  --NONE--    YES     SNASVCMG NONE
DWO469I MS   EP_OPS      DSIOURCP DSI6DST  OPS_MGMT    NO      SNASVCMG ALL
DWO469I MS   SPCS        DSIYPIF  DSIGDS   --NONE--    YES     SNASVCMG NONE
DWO469I MS   MS_CAPS     DSIFPRCV DSI6DST  --NONE--    NO      SNASVCMG ERROR
DWO469I HP   RMTCMD_S    DSIUDST  DSIUDST  N/A         N/A     PARALLEL NONE
DWO469I HP   RMTCMD_R    DSIUDST  DSIUDST  N/A         N/A     PARALLEL NONE
DSI633I REGISTER COMMAND SUCCESSFULLY COMPLETED
```

Figure 139. Sample Output From the REGISTER QUERY Command

Many of the applications shown in Figure 139 are described in later sections. You can think of these applications as *sitting on top* of the LU 6.2 transports. The DSI6DST task must be active to use the LU 6.2 transports.

Ensure that the names in the DSI6SCF sphere of control member and the DSI6INIT LU 6.2 transport initialization member match the node configuration. For example, if the VTAMCP USE option of a focal point is set to yes (`VTAMCP USE=YES`), then the focal point is referenced in DSIDMN and must be referenced by the VTAM CPName using the DSI6INIT member of the entry point. If the USE option of a focal point is set to no (USE=NO), then the focal point must be referenced by the domain name of the NetView program on which it is running.

The VTAMCP statement specifies if the NetView MS transport running on that NetView program can receive MDS-MUs with the VTAM CPName as the destination. Therefore, when coding DSI6SCF and DSI6INIT statements, be aware of the VTAMCP statements in your DSIDMN members.

The SNA protocols used for the MS transport are not limited to NetView. You can use the MS transport to communicate with MS applications on any system or device that supports these protocols, for example, an AS/400.

To use the LU 6.2 transports, first define NetView to VTAM as an LU 6.2 application. The NetView domain ID serves as the VTAM LU 6.2 application name.

Figure 140 shows how to define NetView as a VTAM LU 6.2 application:

```
CNM01    APPL  AUTH=(NVPACE,ACQ,PASS),PRTCT=CNM01,EAS=6,              X
               MODETAB=AMODETAB,DLOGMOD=DSILGMOD,APPC=YES
```

Figure 140. VTAM APPL Statement

For detailed information about writing applications that use the LU 6.2 transports, refer to the *IBM Tivoli NetView for z/OS Application Programmer's Guide*.

LUC

Unlike the LU 6.2 transports, the LUC transport supports communication only between NetView programs. The LUC tasks (for example, CNM01LUC) must be active to use the LUC transport. For NV-UNIQ/LUC alert forwarding, the DSICRTR task must also be active.

OST-NNT

Like the LUC transport, the OST-NNT transport supports communication only between NetView programs. To establish an OST-NNT session, issue a START DOMAIN command from a central system to start a session with a target system NNT. For additional information on OST-NNT sessions, see "Forwarding with OST-NNT Sessions" on page 391.

NetView Architected Focal Point Support

A *focal point application* resides in a central network node and receives information from *entry point applications* that reside in distributed network nodes. The information, which is sent from the entry point applications to the focal point application, belongs to a specific *category* of data, for example, the *alert* category. This section describes how NetView uses methods defined by SNA to keep track of focal points for operations management, alert, and user-defined categories of MS data.

NetView uses a subset of the SNA management services focal point architecture described in the *SNA Management Services Reference*. The way in which NetView handles LU 6.2 focal point support is based on the architecture described in this book.

In contrast to forwarding alerts and status information over LUC sessions, which are unique to NetView, the architectural method can encompass non-NetView entry points and non-NetView focal points. NetView can be an architectural focal point or entry point for any application or device that implements the required subset of functions in the SNA management services focal point architecture. NetView can act as an architectural entry point and as an architectural focal point for alert, operations management, and user-defined categories of information.

When an entry point application sends (forwards) data to its focal point, the entry point sends the data to the focal point over one of the NetView LU 6.2 transports. Data flows over LU 6.2 sessions in the form of MDS-MUs.

The architecture uses the concepts of local and remote focal points.

- A *local focal point* is an MS application acting as a focal point in a node.

  From the perspective of the node which contains the focal point application, the focal point is a local focal point application (it is local to, or resides within, that node).
- A *remote focal point* resides in another node.

  From the perspective of an entry point whose focal point resides in another node, the focal point is a remote focal point.

For example, if an operations management focal point application (OPS-MANAGEMENT-NETOP) resides in node A, then node A is said to contain a local focal point for the operations management category. Suppose a second node, node B, contains an operations management entry point (EP-OPS-

MANAGEMENT). If the node B focal point for operations management is node A, node B has a remote focal point for operations management.

A remote focal point can be further classified as a *primary* or a *backup* focal point. The *current* focal point for a node is the active remote focal point to which entry point applications forward data. The current focal point can never be the primary and the backup at the same time.

- If the primary focal point is available, it is the current focal point.
- If the primary focal point is not available, a defined backup focal point is the current focal point.

Architectural focal points can forward the information they receive from their entry points on to their focal points. This is called *focal point nesting*.

The following sections describe the focal point-related MS applications provided by the NetView program.

Refer to the "Focal Point™ Concepts" section of *SNA Management Services Reference* for information from an architectural point of view.

The MS-CAPS Application

The term MS-CAPS is sometimes used as shorthand for MS capabilities, the name of the architected focal point function set. In this book, however, *MS-CAPS* refers to the MS application that establishes, ends, and communicates focal-point-to-entry-point relationships.

The MS-CAPS application communicates with MS-CAPS applications in other nodes and with other applications by sending major vector X'80F0' over the MS transport. If you want to write applications that interact with MS-CAPS, familiarity with the following subvectors within the X'80F0' major vector is required:

- Focal Point Authorization Request (X'61'). A focal point sends this subvector to an entry point to request that the entry point enter the focal point's sphere-of-control.
- Focal Point Authorization Reply (X'62'). An entry point sends this subvector back to the focal point to accept or reject an authorization request. When a focal point receives a X'62' accept subvector from an entry point, the entry point is added to the focal point's sphere-of-control.

  An entry point can also send this subvector to the focal point to revoke an established relationship. When a focal point receives a X'62' revoke subvector, the entry point is dropped from the focal point's sphere-of-control.
- Entry Point Authorization Request (X'63'). An entry point sends this subvector to the focal point to request inclusion in the focal point's sphere-of-control.
- Entry Point Authorization Reply (X'64'). A focal point sends this subvector back to the entry point to accept or reject an authorization request. If the focal point sends a X'64' accept subvector to the entry point, the entry point is added to the focal point's sphere-of-control.
- Focal Point Notification (X'E1'). When a focal point category changes, MS-CAPS sends the X'E1' subvector to all MS applications that have registered with interest in the category. When a network node changes its focal point, MS-CAPS on the network node sends the X'E1' subvector to MS-CAPS on all served end nodes, notifying them of the change. For example, a change of focal point can result from an operator entering a NetView FOCALPT CHANGE command or from a session failure.

It is the responsibility of the MS-CAPS Application to keep track of the current focal point for all categories. NetView MS-CAPS support is based on the MS_CAPS Function Set architecture presented in *SNA Management Services Reference*.

MS-CAPS saves current focal point information in the VSAM Save/Restore database. MS-CAPS saves the identities of the focal points defined by the DEFFOCPT statement and focal point changes because of a FOCALPT command or a session loss. The DSISVRT task must be active before you can save this information. If you stop and restart NetView, MS-CAPS can use the information to reacquire the most recent primary and backup focal points upon DSI6DST initialization.

At DSI6DST initialization, MS-CAPS reads the DEFFOCPT statements contained in the DSI6INIT member. In the DEFFOCPT statements, you can specify a primary focal point and up to eight backup focal points. When all DEFFOCPT statements have been read, MS-CAPS compares the focal points defined to the focal point details returned from the Save/Restore task. If the DEFFOCPT statements have not been modified since the last time the DSI6DST task was initialized and the OVERRIDE keyword has not been specified, MS-CAPS uses the focal point names returned by the Save/Restore task; that is, MS-CAPS tries to acquire these focal points for their respective categories. In all other cases, MS-CAPS uses the focal point names defined by the DEFFOCPT statements; that is, MS-CAPS first tries to acquire the primary focal point, and if it is unavailable, tries to acquire one of the backup focal points. The backups are processed in the order specified by the DEFFOCPT statements until a backup focal point is acquired.

If a backup is unavailable and the MS-CAPS application is running in an end node, MS-CAPS informs its local applications to send data to the domain focal point (see "MS-CAPS in the Advanced Peer-to-Peer Networking Environment" for additional information). When the primary focal point is unavailable, MS-CAPS sets a timer, and when the timer expires MS-CAPS again tries to acquire the primary. If a focal point (primary or backup) is acquired, MS-CAPS informs its local applications to send data to the acquired focal point.

When MS-CAPS detects a conflict between the focal points defined for a category, MS-CAPS issues a message to the task initiator and the authorized receiver, indicating what action MS-CAPS has taken because of the discrepancy.

MS-CAPS in the Advanced Peer-to-Peer Networking Environment

In an Advanced Peer-to-Peer Networking environment, network nodes provide services for end nodes. One of these services is to inform all end nodes, except migration nodes, of the name and status of the *domain focal point*.

An end node can also use DEFFOCPT statements or the FOCALPT command to establish implicit or explicit focal points. In this case, the end node forwards data to the domain focal point only if it is unable to send data to its primary or backup focal point. An end node cannot control the domain focal point and cannot drop the domain focal point.

Failure Processing

If the MS-CAPS application at an entry point receives notification of an error in communication with a primary focal point for an architectural category, MS-CAPS does the following:

- Sends notification of the failure to all applications that registered interest in the category.

- Sets a timer for an attempt to reacquire the primary focal point. The attempt takes place after a period specified by the REACQPRI option on the DEFAULTS command. Refer to NetView online help for a complete description of DEFAULTS REACQPRI.
- Attempts to acquire the backup focal point if one exists. If the system is unable to establish a session with the backup focal point, it attempts a session with the next backup focal point (if you defined more than one) and so on, until a session is established. When MS-CAPS successfully acquires a backup, it sends notification to local entry-point applications for the category, informing them of their new focal point.
- Attempts to reacquire the primary focal point when the timer expires. If the attempt succeeds, MS-CAPS sends notification to local entry-point applications and sends a revocation notice to the backup. If the attempt fails, MS-CAPS resets the timer and continues to try to regain the primary focal point.

If a backup focal point is the current focal point and MS-CAPS receives notification of a failure in communicating with the backup focal point, MS-CAPS sends notification to all applications that registered interest in the category.

Focal Point Nesting

MS-CAPS provides support for focal point nesting, which permits a NetView node to have both local and remote focal points at the same time. A local focal point receives information from local applications that act as entry points, and the local focal point can then act as an entry point itself by forwarding this information to the remote focal point. If the focal point nesting is incorrectly set up, data can be forwarded in an infinite loop. MS-CAPS detects and breaks such loops. When a loop is detected, the node that detects the loop drops its focal point for the specified category, breaking the loop.

When a FOCALPT command is entered, MS-CAPS performs the function requested by the FOCALPT command. For example, for a FOCALPT ACQUIRE, MS-CAPS acquires a new focal point and revokes (drops) the previous focal point. The FOCALPT command can change or drop the primary and backup focal points.

In the REGISTER QUERY output in Figure 139 on page 374 the MS-CAPS application is identified as `MS_CAPS` in the `APPL` column.

Sphere-of-Control with Architected Focal Points

While architected focal points and the entry points they serve make it possible to establish, end, and communicate focal-point-to-entry-point relationships, you still need to manage those relationships. For example, to control and maintain focal-point-to-entry-point relationships, view those relationships from a centralized point. If a focal-point-to-entry-point relationship fails, a centralized manager is needed to recover that relationship.

NetView provides an architectural function set at the focal point called the sphere-of-control manager (SOC-MGR) that acts as a centralized manager for focal-point-to-entry-point relationships. The SOC-MGR manages all entry points in its sphere-of-control. A *sphere-of-control* is defined as all of the entry points that have or must have an established relationship with a registered focal point.

Sphere-of-Control Functions at the Focal Point

When an application registers as a focal point, it specifies a category of management services data for which it is to be a focal point. A focal-point-to-entry-point relationship can then be established for that particular category. The MS-CAPS application within the focal point and entry point is

responsible for establishing the relationships between the focal point and the entry point. The SOC-MGR, which is part of the MS-CAPS application at the focal point, enables MS-CAPS to provide automated management of the sphere-of-control. The SOC-MGR also enables an operator at the focal point to manage all entry points in the focal point's sphere-of-control.

MS-CAPS Management of the Sphere-of-Control: To provide automated management services for sphere-of-control, MS-CAPS must:

- Maintain a list of entry points that are within a focal point's sphere of control.
- Maintain the state of each entry point within the focal point's sphere of control.
- Attempt to reestablish a relationship with an entry point when the relationship between the entry point and the focal point is lost. This attempt depends on the type and state of the entry point.
- Read information from the sphere-of-control configuration file (DSI6SCF) during NetView initialization and use this information to set up the sphere-of-control environment.
- Restore the sphere-of-control environment during NetView recovery.

Operator Management of the Sphere-of-Control: The SOC-MGR makes it possible for an operator at the focal point to perform the following management functions for the sphere-of-control environment:

- Delete entry points from the sphere-of-control using the FOCALPT DELETE command.
- Display the names and states of entry points in the sphere-of-control using the FOCALPT DISPSOC command.
- Initialize the sphere-of-control environment using the FOCALPT REFRESH command.
- Add entry points to the sphere-of-control configuration file after the SOC-MGR has been initialized, and then dynamically read the changes into the SOC-MGR using the FOCALPT REFRESH command.

Sphere-of-Control Types

The sphere-of-control type is maintained for each entry point by the SOC-MGR at the focal point. The sphere-of-control type defines how the entry point is obtained into the sphere-of-control. Use the FOCALPT DISPSOC command to display the sphere-of-control type for entry points. The sphere-of-control types are:

EXPLICIT
: The focal point has initiated a relationship with an entry point because of an operator command or because the entry point was defined in the sphere-of-control configuration file. The focal point attempts to establish a relationship with the entry point until it is successful. When the relationship is established, the entry point is responsible for reestablishing the relationship if it is lost.

IMPLICIT
: The entry point has initiated a relationship with the focal point. If the relationship is lost, the entry point is responsible for reestablishing the relationship.

An EXPLICIT sphere-of-control type has a higher priority than an IMPLICIT sphere-of-control type when focal-point-to-entry-point relationships are established. For example, when a focal point initiates a relationship with an entry point, the entry point is considered to be explicitly obtained into the focal point's

sphere-of-control. This entry point is then considered to have an EXPLICIT sphere-of-control type in the information maintained by the SOC-MGR.

The entry point that was explicitly obtained into the focal point's sphere of control can then initiate a relationship with the same focal point. The entry point request to initiate a relationship with the focal point is completed successfully. However, because a sphere-of-control type of EXPLICIT has a higher priority than a sphere-of-control type of IMPLICIT, the SOC-MGR continues to list this entry point with an EXPLICIT sphere-of-control type.

Sphere-of-Control States

The SOC-MGR at the focal point maintains information about the state of an entry point in the sphere-of-control. The state of an entry point is determined by:

- The entry point sphere-of-control type
- The previous state of the entry point
- The event that affected the entry point

Use the FOCALPT DISPSOC command to display the sphere-of-control state for entry points. The entry point states are:

ADD PENDING
The focal point has attempted to acquire the entry point into its sphere of control, but the focal point has not yet received the reply from the entry point. The entry point enters the ACTIVE state when the focal point receives a reply indicating that its request has been accepted by the entry point.

ACTIVE
The focal point is actively providing services for the entry point. A focal-point-to-entry-point relationship is established, and the entry point is considered to be in the focal point's sphere-of-control.

DELETE ADD PENDING
While the entry point was in the ADD PENDING state, the operator at the focal point issued a FOCALPT DELETE command. The entry point remains in the focal point's sphere-of-control and continues receiving services from the focal point until another focal point takes over services for the entry point, or until the session between the focal point and the entry point is lost.

DELETE PENDING
While the entry point was in the ACTIVE state, the operator at the focal point issued a FOCALPT DELETE command. The entry point remains in the focal point's sphere-of-control and continues receiving services from the focal point until another focal point takes over services for the entry point, or until the session between the focal point and the entry point is lost.

INACTIVE
While the entry point was in the ACTIVE state, the focal-point-to-entry-point relationship was lost. The entry point remains in the INACTIVE state until the relationship is reestablished, or until the entry point issues a request to drop the focal point.

INACTIVE RETRY
The focal-point-to-entry-point relationship was lost while the entry point was in the ADD PENDING state. The focal point attempts to reestablish the focal-point-to-entry-point relationship.

UNKNOWN
This state is applicable only to entry points with an IMPLICIT sphere-of-control type or to EXPLICIT entry points with a state of DELETE PENDING or DELETE ADD PENDING. An entry point enters an UNKNOWN

state after NetView at the focal point performs a recovery operation. Because it is the responsibility of the entry point to reestablish the focal-point-to-entry-point relationship during recovery, the focal point does not know whether the entry point is aware of the loss of that relationship. If the entry point is aware of the loss, it can establish a relationship with another focal point. If the entry point is not aware of the loss, it continues to maintain a relationship with the focal point. If the entry point does reestablish a relationship with the focal point, the entry point state changes to ACTIVE.

Setting Up the Sphere-of-Control Environment

The sphere-of-control configuration file, DSI6SCF, defines which entry points are explicitly obtained into a focal point's sphere-of-control. This file is read during NetView initialization to set up the focal-point-to-entry-point sphere-of-control environment. The sphere-of-control configuration file can also be updated anytime after NetView initialization to refresh or change focal-point-to-entry-point relationships. The sphere-of-control configuration file contains:

- The entry point name
- The name of the primary focal point category
- The primary focal point name
- The backup focal point name (optional)

Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for more information about defining the sphere-of-control configuration file.

Updating or Changing the Sphere-of-Control Environment: The FOCALPT REFRESH command enables you to dynamically refresh or change focal-point-to-entry-point relationships after NetView has been started. When you issue the FOCALPT REFRESH command, the MS-CAPS application at the focal point reads the sphere-of-control configuration file and updates the current sphere-of-control environment. Focal-point-to-entry-point relationships defined in the sphere-of-control configuration file take precedence over relationships in the current sphere-of-control environment. For example, because the sphere-of-control configuration file defines EXPLICIT entry points, any entry point with an IMPLICIT sphere-of-control type in the current sphere-of-control environment is changed to an EXPLICIT sphere-of-control type when the FOCALPT REFRESH command is issued.

Additionally, if an EXPLICIT entry point exists in a focal point's sphere of control in the current environment, but is not defined in the configuration file when the FOCALPT REFRESH command is issued, the entry point is deleted from the focal point's sphere-of-control.

Restoring the Sphere-of-Control Environment

The MS-CAPS application at the focal point saves information about the entry points in its sphere-of-control in a VSAM Save/Restore database. When an entry point leaves a sphere-of-control, information about the entry point is deleted from the VSAM Save/Restore database.

When NetView or the DSI6DST task ends and then recovers, MS-CAPS checks the VSAM Save/Restore database. If Save/Restore information exists, MS-CAPS uses the information to restore the most current focal-point-to-entry-point environment.

If an entry point has an IMPLICIT sphere-of-control type, or an EXPLICIT entry point with a state of DELETE PENDING or DELETE ADD PENDING, the entry point is restored with an UNKNOWN sphere-of-control state prior to ending. It is then the responsibility of the entry point to reestablish a relationship with the focal

point. If the entry point is aware that the relationship with the focal point is lost, it reestablishes the relationship, and the entry point sphere-of-control state changes to ACTIVE. If the entry point was not aware of the loss, the entry point does not reestablish a relationship with the focal point, and the entry point sphere-of-control state remains UNKNOWN.

How to Define an Architected Focal Point (DEFFOCPT)

Figure 141 illustrates typical focal point definitions for the alert, operations management, and user-defined categories in DSI6INIT (CNMS1040). It also illustrates a typical operations management entry point definition. Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information about DEFFOCPT and DEFENTPT. Note that DEFENTPT only applies to the operations management category.

```
DSTINIT FUNCT=OTHER,XITDI=DSI6IDM
DEFFOCPT TYPE=ALERT,PRIMARY=NETA.CNM02
DEFFOCPT TYPE=OPS_MGMT,PRIMARY=NETA.CNM02,BACKUP=NETB.CNM99
DEFFOCPT TYPE=OPS_MGMT,BACKUP=CNM03
DEFFOCPT TYPE=USERCAT,BACKUP=NETB.CNM99
DEFFOCPT TYPE=USERCAT,PRIMARY=NETA.CNM02,OVERRIDE
DEFFOCPT TYPE=USERCAT,BACKUP=*.CNM05
DEFFOCPT TYPE=USERCAT,BACKUP=CNM03
DEFENTPT   EPONLY=YES
     END
```

Figure 141. Typical Focal Point and Entry Point Definition Statements in DSI6INIT

The DEFFOCPT and DEFENTPT statements are processed by MS-CAPS at DSI6DST task initialization.

The ALERT-NETOP Application

When the hardware monitor BNJDSERV task is initialized, it registers the hardware monitor as ALERT-NETOP, an architected alert focal point. NetView accomplishes the registration automatically; no definitions are required. NetView ALERT-NETOP support is based upon the ALERT_NETOP Function Set and EP_ALERT Function Set architecture. For more information, refer to the *SNA Management Services Reference*.

The NetView program's ALERT-NETOP implementation can receive alerts from other applications over the MS transport in the form of CP-MSUs within MDS-MUs. Such applications act as an EP-ALERT application, and they may reside in a NetView node or a non-NetView node. For example, the IBM SAA Networking Services/2 program for OS/2 acts as an EP-ALERT application and can send alerts to ALERT-NETOP.

The NetView ALERT-NETOP can act as an EP-ALERT application, and can forward alerts to other ALERT-NETOP applications (NetView or non-NetView) over the MS transport in the form of CP-MSUs within MDS-MUs. Therefore, ALERT-NETOP *receives* and *forwards* alerts over the MS transport. This includes the ability to send and receive alerts over CP-CP sessions through the MS transport.

ALERT-NETOP can also receive alerts from the NetView program-to-program interface in the form of CP-MSUs. The CP-MSUs can contain one or more alert major vectors. The hardware monitor splits up the alert major vectors and processes each one individually. See "NetView Program Hardware-Monitor Data and MSU Routing" on page 96 for information about how the hardware monitor processes major vectors.

ALERT-NETOP is displayed as ALERT in the APPL column of the REGISTER QUERY command output. (See Figure 139 on page 374.) The ALERT is short for ALERT-NETOP, the architected name for an alert focal point application. Notice that NetView does not register an EP-ALERT application. ALERT-NETOP acts as an EP-ALERT application; therefore, it is not necessary for NetView to register an explicit EP-ALERT application.

Displaying Alerts Forwarded with LU 6.2

The hardware monitor Alerts Dynamic, Alerts Static, and Alerts History panels display an @ indicator beside alerts that were forwarded to ALERT-NETOP from remote node applications over LU 6.2. Applications that reside in the NetView node are considered local applications, and with few exceptions alerts sent from local applications do not have an @ indicator. Refer to *IBM Tivoli NetView for z/OS User's Guide: NetView* for additional information.

Specifying Architected Alert Forwarding with LU 6.2

NetView supports, through the ALERT-NETOP application, receiving alerts sent over the LU 6.2 transport. NetView ALERT-NETOP acts as an architectural ALERT-NETOP application to receive alerts sent from applications that act as an EP-ALERT.

To forward alerts over LU 6.2 using the NetView ALERT-NETOP application, specify the SNA-MDS option on the ALERTFWD statement in the CNMSTYLE member. The ALERTFWD statement enables you to choose how NetView forwards alerts: through SNA-MDS/LU 6.2 (for ALERTFWD SNA-MDS) or NV-UNIQ/LUC (for ALERTFWD NV-UNIQ). Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information about the ALERTFWD statement.

If you choose NV-UNIQ, ALERT-NETOP can receive alerts over LU 6.2, but it cannot forward alerts over LU 6.2; it can forward alerts only over LUC, as described in "Alert Forwarding with LUC" on page 390. If you choose SNA-MDS, then ALERT-NETOP acts as an architectural ALERT-NETOP and EP-ALERT. As such, it can forward alerts over LU 6.2 to its focal point. The following sections describe SNA-MDS alert forwarding (also called *architectural alert forwarding*, *LU 6.2 alert forwarding*, or *forwarding alerts over LU 6.2*).

Forwarding Alerts to a Non-NetView Focal Point

You can choose a non-NetView product, such as an AS/400, as the NetView alert focal point. From the perspective of a non-NetView product, the alerts it receives from an entry point NetView are in the following categories:

- Alerts that conform to the architecture

  For example, Generic Alert major vector X'0000' with subvector X'92'.
- Alerts that do not conform to the architecture

  For example, OSI Alarms in a X'1330'/X'132F' double major vector.

  Also falling into this category are alerts which conform to the architecture but which the receiving non-NetView product does not support. For example, the architecture permits Alert Resolution major vector X'0002's to be forwarded to an ALERT-NETOP, however some non-NetView products might not support receiving them because these products have not implemented that subset of the architecture.

Non-NetView Focal Points and Architected Alerts: These are properly processed by the non-NetView product. A non-major vector alert, such as a RECFMS, might be displayed with a probable cause of `UNDETERMINED`. Consult the product documentation for more information.

Non-NetView Focal Points and Unarchitected Alerts: Because the focal point is a non-NetView product, the focal point may not know how to process non-architected records; it depends on the focal point product. For example, nongeneric Alert major vector X'0000's (which do not contain subvector X'92') are not architected to be sent to a focal point, however the AS/400 product supports receiving them. Most likely, if the non-NetView product receives an unarchitected record it does one or more of the following, depending on the product:

- Issue an error message.
- Send an MDS Error Message (a X'1532' major vector within an MDS-MU) or an Application Error Message (a X'1532' major vector within a CP-MSU) back to the entry point NetView.

  When the entry point NetView receives the MDS Error Message or Application Error Message, the entry point issues the BNH094I or BNH095I message in accordance with the option specified on the ALERTFWD statement in the CNMSTYLE member.

  Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about ALERTFWD and refer to the *IBM Tivoli NetView for z/OS Installation: Getting Started* for information about the CNMSTYLE member.
- Ignore (discard) the unarchitected alert.

Non-architected alerts may not be properly processed by non-NetView focal points; consult the product's documentation for more information.

Note: If all alerts forwarded from an entry point NetView are to be properly processed by the focal point, the focal point must be a NetView Version 3 or later.

Forwarding Alerts from User-Defined Applications

As described in "User-Defined Categories and User-Defined Applications" on page 389, you can create user-defined applications. User-defined applications can send alerts to ALERT-NETOP. To do so, when your user-defined application registers with the MS transport, it must register with interest in category ALERT.

Once registered, MS-CAPS sends the application a notification (an MDS-MU with major vector X'80F0' and subvector X'E1') which contains the current alert focal point's fully-qualified name: its netid name, nau name, and application name. (The current alert focal point is normally the NetView ALERT-NETOP.) After your application has received the notification, it can send alerts to the alert focal point, and by doing so, it is acting as an architected EP-ALERT. The alerts must be encapsulated within a CP-MSU, and the CP-MSU must be encapsulated within an MDS-MU. All alerts sent must conform to the architecture defined in the Systems Network Architecture library.

When ALERT-NETOP receives alerts that were sent over LU 6.2 from local applications, these alerts are processed as normal local alerts. For example, the @ indicator is not present on the Alerts Dynamic panel for such alerts, because they were not forwarded from a remote node.

Defining a NetView Intermediate Node Focal Point

If NetView has an alert focal point, and NetView receives alerts forwarded with LU 6.2, such alerts are forwarded again by ALERT-NETOP to the NetView focal point. In this case, NetView is an *intermediate node focal point*, also known as a *nested focal point*, because entry points forward alerts to it and it forwards these alerts again to its focal point. You can have zero, one, or more intermediate node focal points, and if you do accidentally construct a loop the MS-CAPS application detects and breaks the loop. To understand how intermediate node focal points forward alerts using LU 6.2, see Figure 142.

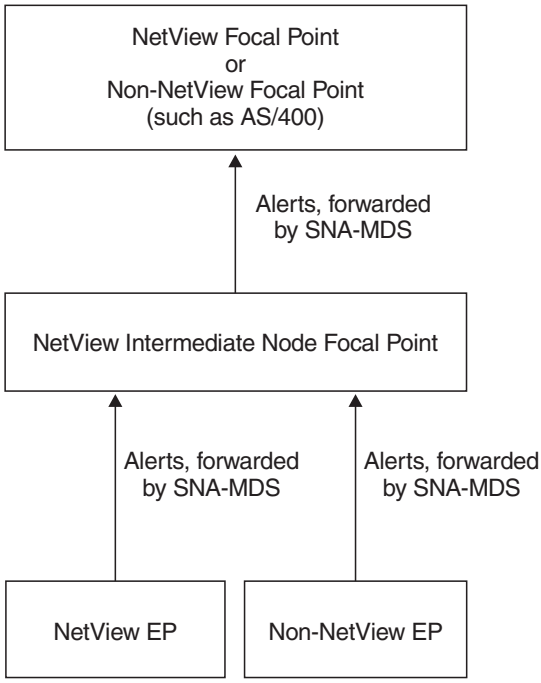


Figure 142. NetView Intermediate Node Focal Point Forwards Alerts with LU 6.2

Only alerts forwarded over LU 6.2 can be forwarded again by an intermediate node focal point. The intermediate node focal point, which receives such alerts, may forward them again, using either the SNA-MDS/LU 6.2 or NV-UNIQ/LUC alert forwarding method. Alerts forwarded over LUC are not forwarded again, they are forwarded only once from the entry point to the focal point. The receiving focal point is not permitted to forward them again. You can think of LUC alert forwarding as a *one hop* alert forwarding method.

If you do not want an intermediate node NetView to record data to the hardware monitor database, but to simply *pass through* an intermediate node, specify the ALRTINFP NORECORD statement in BNJMBDST. The ALRTINFP setting only applies to alerts forwarded with LU 6.2 from remote nodes; all other alerts are unaffected. Refer to ALRTINFP in the *IBM Tivoli NetView for z/OS Administration Reference* for more information.

At the ultimate (topmost in the diagram) NetView focal point, the domain name that the entry point alert is recorded against in the hardware monitor database is obtained as follows:

Note: This is the domain name displayed under the DOMAIN column on the Alerts Dynamic, Alerts Static, and Alerts History panels, and is displayed in the pictorial hierarchy at the top of several other hardware monitor panels.

- If the entry point is a Version 3 or later NetView and the ultimate focal point is a Version 3 or later NetView, when the alert appears on the Alerts Dynamic panel at the ultimate focal point, the domain name present under the DOMAIN heading is the entry point domain name.

  The alert is recorded in the focal point database against the entry point NetView domain name. Only a single alert record is recorded in the database, the complete set of data is present at the entry point database. Recording a single alert record to the database saves database storage and processor time.

  An operator at the ultimate Version 3 or later focal point can retrieve hardware monitor data from the entry point database through the Distributed Data Base Retrieval function by entering `SEL# M` from the Alerts Static panel, and through the SDOMAIN command. For additional information, refer to the *IBM Tivoli NetView for z/OS User's Guide: NetView* and the NetView online help.

  The presence of zero, one, or more intermediate nodes does not matter, so long as an LU 6.2 session can be established between the ultimate focal point and the entry point. If an LU 6.2 session cannot be established between the ultimate focal point and the entry point, the Distributed Database Retrieval function fails. However, the SDOMAIN command might complete successfully because it attempts to establish an LUC session or an OST-NNT session after it determines that it cannot establish an LU 6.2 session.
- If the entry point is a non-NetView and the ultimate focal point is a NetView, then when the alert appears on the Alerts Dynamic panel at the ultimate focal point, the domain name present under the DOMAIN heading is the ultimate focal point domain name.

  This is true regardless of the version and release level of the ultimate focal point. The alert is recorded in the focal point database as if it were a local alert.
- If the entry point is a Version 3 or later NetView and the ultimate focal point is a pre-V3R1 NetView, then when the alert appears on the Alerts Dynamic panel at the ultimate focal point, the domain name present under the DOMAIN heading is the ultimate focal point domain name.

  This is because pre-V3R1 NetViews treat all LU 6.2 forwarded alerts they receive as if they were forwarded from a non-NetView. The alert is recorded in the focal point database as if it were a local alert.

Recording Filters for SNA-MDS/LU 6.2 Forwarded Alerts

Alerts forwarded with LU 6.2 from non-NetView entry points or from local applications have the hardware monitor recording filters applied to them as if they were local alerts. If these alerts pass the recording filters, a complete set of data is recorded to the hardware monitor database. This data consists of zero, one, or more event records, statistics records, and alert records, among others.

Alerts forwarded with LU 6.2 from remote-node NetView entry points also have the hardware monitor recording filters applied to them as if they were local alerts; however, the AREC and ESREC recording filters are always forced to PASS. Each of these alerts is recorded in the hardware monitor database as a single alert record, and the complete set of data is available only at the entry point. This process is known as *alert-only* recording, and alerts forwarded with LUC are also recorded as alert-only. The focal point does not quickly fill up the database and uses less processor time.

You can use the hardware monitor ROUTE recording filter to designate the alerts NetView must forward. However, an alert must pass the ESREC and AREC filters before it goes to the ROUTE filter, and alerts already forwarded once by LUC are never forwarded again. You can use the SRFILTER command to specify filter

settings from the hardware monitor, or you can use the SRF action to specify them from the automation table. The automation table SRF action can override the recording filters for all alerts except alerts forwarded with LUC. For example, you can use the SRF action to record non-NetView entry point alerts as alert-only, or record entry point NetView alerts with the complete set of data (not alert-only).

See "Filtering Alerts" on page 299 for more information about the SRFILTER command and "Actions" on page 209 for more information about the SRF action.

Queueing Alerts When the Focal Point Is Unavailable

Alerts received by a NetView entry point during the time that its focal point is unavailable are marked as *held* in the alert cache. Refer to ALCACHE in the *IBM Tivoli NetView for z/OS Administration Reference* for more information about defining an alert cache. If MS-CAPS later successfully reacquires the focal point, MS-CAPS notifies ALERT-NETOP that the focal point has been reacquired, and ALERT-NETOP loops through the alert cache and processes each of these held alerts. This processing involves first reapplying the ROUTE recording filters to this now-held alert, because when the ROUTE recording filter was initially applied to the alert, the alert was not marked as held. If the ROUTE recording filters are passed, the alert is forwarded to the focal point.

An alert cache might not be defined, or held alerts may roll off the alert cache before a new focal point is acquired. Such alerts are not forwarded to the focal point, however a count is kept of the number of these alerts. (This count wraps at 10000.) If the focal point is later reacquired, the DSI382I message is issued and it displays this count.

A focal point can be flooded with held alerts forwarded from one or more NetView entry points. If you want to prevent flooding, you can set the ROUTE recording filter at the NetView entry point so that held alerts are blocked and not forwarded to the focal point when the focal point is reacquired; use the NPDA SRFILTER ROUTE BLOCK E HELD command. This command is commented out in the CNMSTYLE member (see the NPDA.PDFILTER statement). You can uncomment it so that such held alerts are not forwarded to the focal point. Held alerts from NetView entry points must be blocked at the entry point and not at the focal point, whereas held alerts from non-NetView entry points must be blocked at a NetView focal point. See "Recording Filters for SNA-MDS/LU 6.2 Forwarded Alerts" on page 386 and refer to the SRFILTER command in NetView online help for more information.

If a focal point NetView is flooded with held alerts from non-NetView entry points, the focal point AREC recording filters can be set to filter out such held alerts. For example, the NPDA SRFILTER AREC BLOCK E HELD command blocks incoming alerts which contain a subvector 92 with its held bit set. The hardware monitor default AREC recording filters block many, but not all, held alerts, and you can see these by issuing the NPDA DF AREC command.

Distributed Database Retrieval for SNA-MDS/LU 6.2 Forwarded Alerts

At a NetView focal point, when Distributed Database Retrieval occurs for a selected alert, either the MS transport or the LUC transport is used to retrieve the data from the entry point hardware monitor database. The transport used in Distributed Database Retrieval is the same transport over which the focal point received the alert. For example, if the focal point received an alert over LU 6.2, then whenever Distributed Database Retrieval occurs for this alert it also uses LU 6.2.

Refer to the *IBM Tivoli NetView for z/OS User's Guide: NetView* for more information about Distributed Database Retrieval. Distributed Database Retrieval can fail when intermediate nodes are involved, as described in "Defining a NetView Intermediate Node Focal Point" on page 385 and "Alert Forwarding with LUC" on page 390.

Secondary Recording for SNA-MDS/LU 6.2 Forwarded Alerts

With LUC alert forwarding, hardware monitor secondary recording is prevented from occurring at the focal point. With SNA-MDS/LU 6.2 alert forwarding, secondary recording is enabled. Refer to the *IBM Tivoli NetView for z/OS User's Guide: NetView* for more information concerning secondary recording.

XITCI Exits and SNA-MDS/LU 6.2 Forwarded Alerts

Refer to *IBM Tivoli NetView for z/OS Programming: Assembler* for information concerning XITCI exits and SNA-MDS/LU 6.2 forwarded alerts.

Services Provided by MS-CAPS and FOCALPT Command

Because ALERT-NETOP acts as an architectural EP-ALERT, the services provided by MS-CAPS and the FOCALPT command are available to ALERT-NETOP. See "The MS-CAPS Application" on page 376 for more information concerning the functions provided by MS-CAPS. Also, see "Changing, Dropping, and Listing Focal Points" on page 397.

The LINK-SERVICES-NETOP Application

When the hardware monitor BNJDSERV task initializes, the hardware monitor is registered as LINK-SERVICES-NETOP, an architectural link event (major vector X'0001') focal point. NetView accomplishes the registration automatically; no definitions are required. The NetView LINK-SERVICES-NETOP support is based on the LINK_SERVICES_NETOP function set architecture described in *SNA Management Services Reference*.

The NetView LINK-SERVICES-NETOP function can receive link events from other local applications over the MS transport in the form of CP-MSUs within MDS-MUs. The sending applications must be local applications, which reside in the same node as NetView.

LINK-SERVICES-NETOP is displayed as LINKSERV in the APPL column of the REGISTER QUERY command output in Figure 139 on page 374. The LINKSERV is short for LINK-SERVICES-NETOP, the architectural name for a link event focal point application.

The OPS-MGMT-NETOP and EP-OPS-MGMT Applications

NetView also registers as an architectural focal point for the operations management category (OPS-MANAGEMENT-NETOP), unless you add a DEFFOCPT statement to specify another focal point or a DEFENTPT EPONLY=YES statement in DSI6INIT (CNMS1040). The NetView OPS-MANAGEMENT-NETOP and EP-OPS-MANAGEMENT support is based on the OPERATIONS_MGMT_NETOP Function Set and EP_OPERATIONS_MGMT function set architecture described in *SNA Management Services Reference*.

Regardless of whether the operations management focal point is registered, NetView automatically registers one of its facilities as an architectural operation-management entry point (EP-OPS-MANAGEMENT). If the node is to have an entry point but not a focal point (its focal point is remote), you can define primary and backup focal points for operations management in DSI6INIT (CNMS1040).

If the NetView OPS-MANAGEMENT-NETOP application is registered, a REGISTER QUERY command shows it as OPS_MGMT. EP-OPS-MANAGEMENT shows up as EP_OPS (see Figure 139 on page 374).

User-Defined Categories and User-Defined Applications

Your NetView MS applications can serve as both focal points and entry points for user-defined categories of information. When one of your applications has registered with the MS transport as either a focal point or an entry point in a user-defined category, operators can use the NetView FOCALPT command to control the node used as the category's focal point.

To register an application as a focal point, use the REGISTER command, macro, or service routine with the name of the category as your MS application name and a FOCALPT=YES operand. Then, an operator or command procedure can establish a focal-point to entry-point relationship for the category. For example, if you register a focal point application with a name of USERDATA, you can issue a FOCALPT command for the USERDATA category.

To establish a node as an entry point for a user-defined category, use the REGISTER command, macro, or service routine with an FPCAT parameter that specifies the category. Your application is registered as an entry point and receives information from MS-CAPS about the current focal point. You can have more than one entry point application for a category in each node.

Refer to the NetView online help for REGISTER command syntax, the *IBM Tivoli NetView for z/OS Programming: Assembler* for DSI6REGS macro syntax, and to *IBM Tivoli NetView for z/OS Programming: PL/I and C* for the syntax of the CNMRGS service routine.

You can define multiple user-defined entry point and focal point applications and categories. An advantage to registering user-defined applications with the MS transport is that such applications use the services provided by the MS-CAPS application (including the SOC-MGR function set) and the FOCALPT command. For example, if you have one or more user-defined entry point applications for a user-defined category, MS-CAPS notifies all such applications when the current focal point for that category changes. NetView operators and automation can use the FOCALPT command to control which systems act as the focal point in each category. MS-CAPS and FOCALPT functions also apply to communication with non-NetView applications. For example, a user-defined application registered with the NetView MS transport can serve as a focal point for non-NetView systems in a given user-defined category, and likewise, can serve as an entry point and accept non-NetView focal points.

NetView-Unique Focal Point Support

As explained in the previous section, the NetView architectural focal point support allows NetView to act as an entry point and as a focal point for the alert, operations management, and user-defined categories of information using the LU 6.2 transports. This support, based upon the *SNA Management Services Reference* architecture, permits interoperability with NetView and non-NetView systems.

The NetView program also provides focal point support for the alert and status categories, which is unique to NetView. With this NetView-unique focal point support, the entry points and focal points must be NetView programs. The NetView-unique focal point support provides less function than the architectural

focal point support, because the NetView-unique focal point support cannot use the services that are provided with the architectural focal point support. For example, focal points that are unique to the NetView program cannot use the services provided by the MS-CAPS application (including the SOC-MGR support).

Alert Forwarding with LUC

You can use NetView to forward alerts to a focal point using LUC sessions (NV-UNIQ/LUC method). You can forward alerts using the architectural SNA-MDS/LU 6.2 forwarding method, except when the focal point is most often a pre-V3R1 NetView. Unlike OST-NNT sessions, LUC sessions are established automatically. If you have established appropriate system definitions, NetView opens LUC sessions as necessary to forward alerts. Each NetView can have one focal point for alerts. For more information, refer to *Advanced Network and Systems Management*.

Command and Message Forwarding

To manage your distributed systems, operators and automation applications on the central system often must issue commands to the distributed systems. You can forward commands with the RMTCMD command. The RMTCMD command can forward any command that NetView normally processes, except commands that produce full-screen output. To issue commands for special tasks, such as initialization and shutdown of a distributed system, you can use the System Automation for z/OS licensed program. See "Establishing Remote Operation" on page 17.

Message forwarding relates closely to command forwarding. You can forward messages using distributed autotasks that RMTCMD sets up or using the same OST-NNT sessions employed by the ROUTE command to link operator station tasks (OSTs) and NetView-NetView tasks (NNTs).

Forwarding with the RMTCMD Command

With the RMTCMD command, you specify the command to forward and the target NetView LU name. Unless you already have a session with a distributed autotask on the target system, NetView sets up a session automatically before forwarding the command. Any messages that the command generates return to you.

On the target system, you must have an operator ID that the RMTCMD command can use. When you issue the RMTCMD command, you can specify the ID. If you do not specify an ID, the RMTCMD command uses an ID equal to your ID on the sending system. If the ID is not yet active, the RMTCMD command starts the ID as a distributed autotask and processes the forwarded command on that autotask. Thereafter, you have an association with the distributed autotask on the target system.

The target NetView forwards all messages received by the distributed autotask back to the system from which you issued the RMTCMD command. The OST that issued the RMTCMD command receives the forwarded messages. Forwarded messages include any responses to your forwarded commands. Forwarded messages also include any other miscellaneous messages that the target system might send to the distributed autotask.

Flexibility in Communication: Distributed autotasks provide flexible communication. Suppose you want to forward messages from a distributed system to a central system for exception notification, to inform operators of problems that local automation encounters. You can issue the RMTCMD command from the

central system to forward a command, possibly just a dummy command, to the distributed system. This sets up a distributed autotask. After that, automation can send messages to the distributed autotask any time it needs to forward information to the central system.

After you issue the RMTCMD command, your distributed autotask remains active until you issue the ENDTASK command or logoff. By issuing the RMTCMD command from an autotask that never logs off, you can establish a permanent session for message forwarding.

Depending on your design, automation on a distributed system might need to forward a message when no distributed autotask yet exists. In this case, the distributed system itself might issue the RMTCMD command and forward an MSG command to the central system to issue the message. This method gives you a new DSI039I message on the central system with the text of your choice. However, it does not allow you to forward an existing message, complete with associated automation internal function request (AIFR) data.

Nesting RMTCMD Commands: To forward an existing message when you have no distributed autotask, the distributed system can use nested RMTCMD commands to have a RMTCMD command sent back from the central system. This sets up a distributed autotask that you can use for message forwarding. A REXX automation procedure can issue the command in Figure 143 at NETVDS to establish message forwarding from NETVDS up to NETVCS.

```
'RMTCMD LU=NETVCS,OPERID=AUTO1 EXCMD AUTO2',
   'RMTCMD LU=NETVDS,OPERID=AUTO3 MSG AUTO3,Dummy Message'
```

Figure 143. RMTCMD Example

Issuing this command sends an EXCMD command to AUTO1 on the central system, which routes a second RMTCMD to AUTO2. AUTO2 then issues RMTCMD to establish a session with AUTO3 on the distributed system, and message forwarding can begin. It is assumed here that AUTO2 is already active; if not, you can first issue an AUTOTASK command to start it.

Each RMTCMD distributed autotask can connect to only one master OST at a time. However, a master OST can have as many distributed autotasks as you want. You can use RMTCMD commands nested within each other to forward commands and messages to their destinations through intermediate nodes. In this case, you can use the EXP parameter to determine whether the commands and messages go through the automation table on the intermediate nodes. Refer to NetView online help for the syntax of the RMTCMD command.

Forwarding with OST-NNT Sessions

A second way to forward commands and messages is with OST-NNT sessions. To use these, you begin by issuing a START DOMAIN command from the central system to start a session with a target system NNT.

An NNT can be any available operator ID that is not currently logged on. The same operator ID can be used by an operator, a distributed autotask, a regular autotask, or an NNT, depending on how you start the task. An OST can log on to NNTs in several domains at the same time, but only to one NNT per domain. An NNT cannot connect to more than one OST at a time.

Once you have established an OST-NNT session, you can use the ROUTE command to send a command from the OST to the NNT on the second NetView.

Any messages received by the NNT, including responses to a forwarded command, go back to the OST on the central NetView. Therefore, NNTs act in much the same way as the RMTCMD command's distributed autotasks, and you can use NNTs for message forwarding.

All automation messages sent across OST-NNT sessions are rebuilt at the target domain. All automation action flags except HOLD, BEEP, and DISPLAY are reset during this message rebuilding process. Preservation of the HOLD, BEEP, and DISPLAY actions enables cross-domain messages to be automated at the target domain.

Attention: If you are using extended multiple console support (EMCS) consoles, use the RMTCMD command and LU 6.2 sessions for all cross-domain sessions to prevent loss of data. Otherwise, if the sessions are established between an OST and an NNT, messages are sent without any appended message data block (MDB) data structures. These data structures contain special information about a message, such as the highlighting (including color) assigned to the message. These data structures contain some DOM information that is associated with the message. Therefore, such information in the MDB data structures is lost on the OST-NNT sessions.

The RMTCMD command is the method to use for command and message forwarding. The RMTCMD command uses the LU 6.2 protocol for better performance and does not require operators to manually start sessions before forwarding commands. Refer to the NetView online help for the syntax of the RMTCMD, START DOMAIN, and ROUTE commands.

Using an Intermediate Focal Point for Message Forwarding

An intermediate focal point can collect data from several distributed systems and forward it to the focal point. The distributed systems assigned to an intermediate focal point treat it as their focal point. They set up sessions with the intermediate system and send messages to it just as they would to a focal point system.

You do not need operators at the intermediate system. The intermediate system can use a NetView autotask, command lists, and the automation table to function without intervention.

Using intermediate focal points helps to concentrate sessions. Many distributed systems can have sessions with one intermediate system, which can establish a single session with the focal point. Therefore, you limit the number of systems that must communicate with the focal point directly. The intermediate system might also perform external automation and recovery for its distributed systems, reducing the load on the focal point.

Intermediate focal points are especially valuable in multisite environments. Strategically placed intermediate focal points can reduce the overhead associated with switched lines or the cost associated with leased lines.

Message/Alert Forwarding with OST-NNT

In addition to the SNA-MDS/LU 6.2 and NV-UNIQ/LUC alert forwarding mechanisms, there is a third mechanism for forwarding alerts. It is an older method, and with it the OPER filter is used to convert alerts to BNJ146I messages. You can then use message forwarding to transmit the BNJ146I message to another NetView and the GENALERT command to reconstitute a similar alert.

Full-Screen Functions and the Terminal Access Facility

Other NetView functions can help you manage distributed systems from a central location. These functions include full-screen functions and the terminal access facility (TAF).

Using the SDOMAIN Command While Monitoring

The hardware monitor and the session monitor can assist in centralized operations, because they can display data from other domains. For example, operators can issue the hardware monitor command SDOMAIN to switch the domain they are monitoring. If the forwarded alerts do not provide enough information about a particular situation, operators can use the SDOMAIN command to get additional hardware monitor information from a distributed system. Similarly, the session monitor accepts an SDOMAIN command that enables operators to view session data on distributed systems.

Using a TAF Session to Shift Domains

Another option for shifting the domain you monitor is to use a TAF session. Focal-point operators can use a TAF session to log on to other NetView domains in either full-screen or operator-control mode. Automation routines can also use TAF, but only in the operator-control mode. See Table 17 on page 430 for suggestions about using TAF for automation.

Logging on to a Distributed System Directly

Of course, if none of these methods solves a problem, you can log on to NetView on a distributed system directly. NetView Access Services can assist you if you want to log on to a large number of systems simultaneously.

Limitations

When you use an SDOMAIN command, work with a full-screen TAF session, or directly log on to a distributed system, you do not see consolidated data from several domains on a single panel. Another disadvantage of full-screen methods is that automation cannot use them. Therefore, full-screen methods are better suited to problem determination in exceptional cases than to continuous monitoring.

Choosing a Forwarding Method

You can transmit information between a distributed system and a focal point with command forwarding, message forwarding, alert forwarding, status forwarding, and the LU 6.2 transports. In addition, you can obtain extra information for problem determination by using full-screen methods.

The following guidelines can help you to determine which method of forwarding information is appropriate for you.

- Using an MVS system for a focal point, status forwarding can effectively provide operators with information about the state of your network. Although you must provide definitions and choose focal points, the forwarding is automatic, and you have the advantage of a graphical interface.
- If you prefer to work with messages, use the RMTCMD command and distributed autotasks. This technique allows you to correlate asynchronous data using the PIPE command, and enables you to track all active remote tasks using the RMTSESS command processor.

- For forwarding exception notifications, you can choose message forwarding, LUC alert forwarding, or the LU 6.2 transports.
- If you prefer to work with alerts, use SNA-MDS/LU 6.2 alert forwarding. However, if the alert focal point is most often a pre-V3 NetView, then use NV-UNIQ/LUC alert forwarding.
- The LU 6.2 transports provide a flexible communication option if you are willing to do some customization. You can use them for exception notification and other data transmission you require. Use the MS transport for low-volume transmissions that require high reliability, such as exception notification. Use the high-performance option of the MS transport for high-volume transmissions, where speed is important.

  Refer to the *IBM Tivoli NetView for z/OS Application Programmer's Guide* for more information about choosing between the two versions of the LU 6.2 transport.
- A need to communicate with NetView programs prior to Version 2 Release 2 might restrict your options. Table 16 shows the release of NetView needed for each forwarding mechanism.

Table 16. NetView Forwarding Options by Release

Option	Release Required
Command Forwarding	
Using the ROUTE Command	Any
Using the RMTCMD Command	V2R2 or later
Data Forwarding	
Forwarding Messages with OST-NNT Sessions (START DOMAIN)	Any
Forwarding Alerts with LU 6.2 Sessions	V3R1 or later
Forwarding Alerts with LUC Sessions	Any
Forwarding Status	V2R1 or later
Forwarding Messages with Distributed Autotasks (RMTCMD)	V2R2 or later
LU 6.2 Transports	V2R2 or later
Changing Focal Points	
Using the CHANGEFP Command	Any
Using the FOCALPT CHANGE Command	V2R2 or later

Choosing a Configuration

When choosing a configuration for the centralized-operations environment, consider both the physical connections that connect your focal points with distributed systems and the type of session you must use. You might also want to include backup or intermediate focal points in your design.

Leased and Switched Lines

You can attach a focal point to a distributed system with a leased line, such as a channel, or with a switched line. Whereas *leased lines* are permanent connections

between systems, you establish a connection over a *switched line* by dialing. With switched lines, either a focal point or a distributed system can initiate and end sessions between the two. There can also be communication controllers between the two.

In most instances, use leased lines, particularly when the focal point and the distributed systems are in close proximity. Leased lines require less CPU utilization by the NetView and VTAM programs.

However, switched lines can be much less expensive than leased lines. Switched lines are often appropriate when you have several distributed systems at remote sites or when you expect very little traffic between a distributed system and its focal point. Similarly, they can be useful in connecting distributed systems to a backup focal point. Switched lines can help you minimize line costs without sacrificing the advantages of interconnected multisystem automation. Figure 144 illustrates a switched-line configuration.

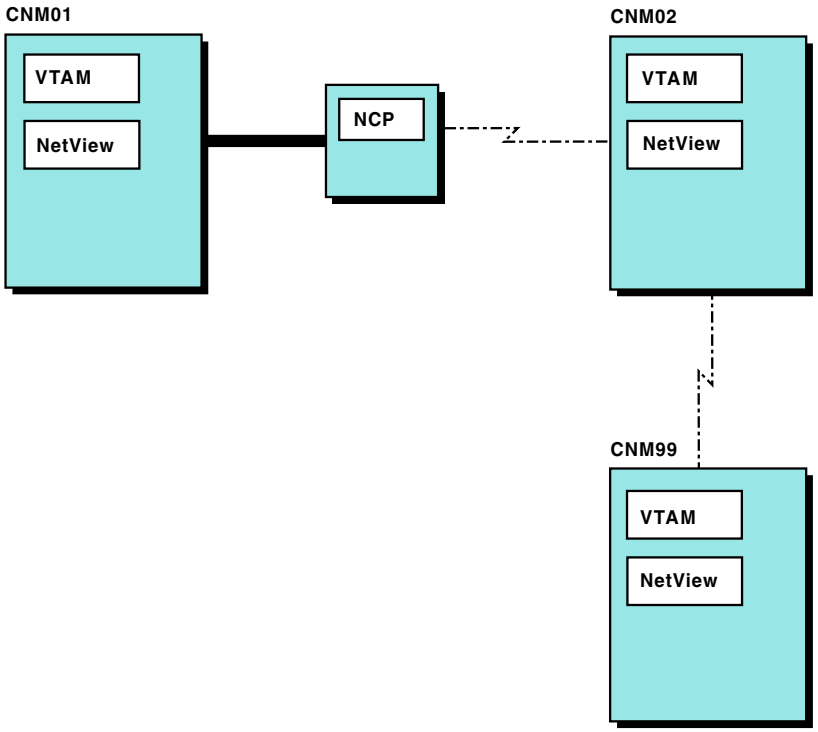


Figure 144. Switched Line Support

If you perform message forwarding with the NetView samples or LUC alert forwarding, NetView establishes the dialed connections automatically. For instructions about switched-line configurations, refer to *IBM Tivoli NetView for z/OS Installation: Getting Started*. If you perform LUC alert forwarding over a switched line, distributed database retrieval also uses the switched line. With the RMTCMD and LU 6.2 transports (including SNA-MDS/LU 6.2 alert forwarding), dialing is left to the VTAM program. Status forwarding requires leased lines.

An operator or autotask with access to the program operator interface (POI) can explicitly activate or deactivate a switched link by issuing a VTAM command:

- `V NET,DIAL,ID=linkstation_name` activates a link.
- `V NET,HANGUP,ID=linkstation_name` deactivates a link.

For more information about the syntax of these VTAM commands, refer to the z/OS Communications Server library.

Automation can take advantage of a switched connection from NetView if you have your command procedures issue the VTAM commands to activate and deactivate links. Before starting a session, NetView can determine whether to activate a switched connection. If so, it can activate the line before requesting the session. The NetView command list DIALCDRM (CNME7023 and CNME1502) performs a dial and shows examples of how you can use the VTAM DIAL and HANGUP commands in a NetView command procedure.

Persistent and Nonpersistent Sessions

NetView can automatically establish communication between a distributed system and its focal point. When a distributed system recognizes that it has information to send to the focal point, NetView can establish a session and forward the data. Depending on your definitions, NetView opens either a persistent or a nonpersistent session.

- A *persistent* session remains active after data is forwarded.
- A *nonpersistent* session ends after a user-specified time, if NetView does not forward additional data.

In general, persistent sessions are used in an environment of leased lines or channels and a large amount of forwarded traffic. Nonpersistent sessions, however, are usual when leased lines connect the distributed and focal-point systems.

Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for the definitions necessary for choosing between persistent and nonpersistent sessions. For each domain that NetView communicates with, you can make a separate choice of whether LUC sessions must be persistent or nonpersistent. LUC sessions are used for alert forwarding, status forwarding, distributed database retrieval, and cross-domain viewing with the session monitor or the hardware monitor using the SDOMAIN command. When you use the NetView samples to set up message forwarding, you can also choose whether they use persistent or nonpersistent sessions.

However, NetView does not control whether the RMTCMD command and the LU 6.2 transports (including SNA-MDS/LU 6.2 alert forwarding) use persistent or nonpersistent sessions. You must use VTAM to make this decision.

The rules that apply to lines and sessions between a distributed system and its focal point also apply to lines and sessions between a distributed system and its backup focal point. That is, you can use switched or leased lines and persistent or nonpersistent sessions.

However, use nonpersistent sessions for message forwarding with the NetView samples if you expect forwarding to the primary focal point to quickly resume. This is because a persistent session continues to carry data to the backup focal point after the primary focal point becomes available, unless you explicitly end the session.

Using More Than One Focal Point

When forwarding information from a distributed system to a focal point, it is common to choose a single focal point for all types of data. This design enables an

operator or automation application at the focal point to gather all of the relevant data about a given distributed system. However, you can use several focal points if you prefer.

A distributed system can have separate focal points for each category of forwarded data: messages, alerts, status information, and operations management data. In addition, the system can have one focal point for each user-defined category of MS application.

If you want to divide data in some way other than by these categories, use one of the mechanisms that enable you to implement customized designs, such as RMTCMD message forwarding or the LU 6.2 transports. For example, you might want to send low-priority notifications to one focal point and high-priority notifications to another. In this case, you might write an application that establishes RMTCMD sessions with each message recipient and determines which recipient is to receive each message. Similarly, you can use the LU 6.2 transports to direct information to the application and the system of your choice.

Changing, Dropping, and Listing Focal Points

The FOCALPT CHANGE and FOCALPT ACQUIRE commands enable you to change focal points for both architectural and NetView-unique focal points. To change a focal point for alerts (for both SNA-MDS/LU 6.2 and NV-UNIQ/LUC alert forwarding), status, operations management, or a user-defined category of MS application, use FOCALPT CHANGE or FOCALPT ACQUIRE.

You issue the CHANGEFP or the FOCALPT CHANGE command from the new focal-point system and specify a target system. The domain from which you issue the command becomes the primary focal point of that target system until you issue another change command or stop and restart NetView on the target system. For messages, alerts, operations management data, and user-defined MS categories, you can also specify a new backup focal point. Issue the FOCALPT ACQUIRE command to specify new focal point systems from an entry-point system.

Depending on the ALERTFWD statement specified in the CNMSTYLE member (refer to the *IBM Tivoli NetView for z/OS Administration Reference* for information about ALERTFWD), a Version 3 or later entry point NetView forwards alerts with either SNA-MDS/LU 6.2 alert forwarding or NV-UNIQ/LUC alert forwarding. Therefore, the FOCALPT CHANGE and ACQUIRE commands establish alert forwarding from a Version 3 or later entry point NetView to its focal point by one mechanism or the other. When NV-UNIQ/LUC alert forwarding is used, and if the focal point and entry point are both NetView programs, the NetView DSICRTR task on the focal point sends a REQUEST message to its counterpart on the distributed system. The alert-forwarding LUC session between the distributed system and its old focal point ends after all alerts already queued for the session are sent. Other alerts are sent to the new focal point.

When SNA-MDS/LU 6.2 alert forwarding is used, the MS-CAPS application establishes the entry point-focal point relationship. Once the relationship is established, the ALERT-NETOP application forwards all alerts it receives to its focal point, which is also an ALERT-NETOP application, over the MS transport.

For example, with either SNA-MDS/LU 6.2 or NV-UNIQ/LUC alert forwarding, you can change the CNMDS alert focal point to CNMFP and its backup to CNMBA by issuing the following command:

```
FOCALPT CHANGE TARGET=CNMDS BACKUP=CNMBA FPCAT=ALERT
```

By substituting an operand of FPCAT=OPS_MGMT, FPCAT=STATUS, or FPCAT=*user-defined*, you can change an operations management or status focal point, or a focal point for a user-defined MS category. Requests in the operations-management and user-defined categories are handled by the MS-CAPS application, as are requests in the alert category that you send to a non-NetView target or send to a Version 3 or later NetView target that has "ALERTFWD SNA-MDS" coded in the CNMSTYLE member. See "The MS-CAPS Application" on page 376 for more information about categories.

Changing a status focal point is a lengthy process, because the new focal point has to start by collecting initial status information. Change the status focal point only if you expect the primary focal point to be out of service for an extended time. You cannot use the BACKUP operand with status focal points.

To change and add backup focal points, use the FOCALPT ACQUIRE command. This command enables you to:

- Change the backup focal point name.
- Define a new backup list for a data category.
- Add backup focal points to an existing backup list.
- Remove focal points from the backup list.

You can also use FOCALPT ACQUIRE to restore the focal points to those defined in DSI6INIT.

Use the FOCALPT QUERY or LIST FOCPT command to list a system's focal points. You can issue the FOCALPT DROP command on a distributed system to stop forwarding a category of information to a focal point, except for status information or messages. You can also issue FOCALPT DROP to remove one or more focal points from the backup list. You can issue the ENDTASK command to end message forwarding by deactivating a distributed autotask. To stop forwarding messages with an OST-NNT session, you can send a LOGOFF command to the NNT.

Part 7. Additional NetView Automation Topics

Chapter 27. Automating Other Systems, Devices, and Networks

The previous chapters describe automation of processors that are capable of running the NetView program. They describe automating devices and networks that use SNA protocols and report to the NetView program through the VTAM program. In this chapter, NetView automation capabilities for automated management of many other IBM and non-IBM systems, devices, and networks are described.

NetView automation capabilities for a non-NetView system or non-SNA device depend on the capabilities of the system or device. The system or device must be able, directly or indirectly, to send problem reports and other information to the NetView program in a form (messages or MSUs) that can be automated and to receive commands from NetView.

For information about managing non-SNA networks through automation, refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

Often a product that cannot be automated directly can be automated with an appropriate interface product. This chapter describes a few examples of interface products that implement service points and enable you to expand the scope of your NetView automation:

- Tivoli NetView for UNIX service point
- Event/Automation Service
- Service point application (SPA) router and remote operations service (ROPS)
- NCP frame relay switching equipment

See Chapter 2, "Overview of Automation Products," on page 21 for some additional examples of interface products.

Tivoli NetView for UNIX Service Point

The Tivoli NetView for UNIX service point is an interface program that can assist NetView in managing non-SNA devices. The Tivoli NetView for UNIX service point runs on a RISC System/6000 machine.

The Tivoli NetView for UNIX service point provide services to applications that manage outboard devices such as a private branch exchange (PBX), LAN, or T1 multiplexer. The service point application is the active agent that communicates with the outboard device, formats alerts, sends them to NetView, and receives and responds to commands. Therefore, the automation capabilities available to you are those supported by the service point application. One example of a service point application is the Host Connection function of Tivoli NetView.

The primary task of a Tivoli NetView service point in automation is to send alerts to the NetView system. The Tivoli NetView for UNIX service point does not have local management capabilities but instead acts as an operatorless gateway between NetView and a non-SNA network.

When an alert reaches NetView, the NetView automation table can issue a command procedure in response. This capability is discussed in Chapter 22,

"Automating Messages and Management Services Units (MSUs)," on page 317. The command procedure that is issued can attempt to solve the problem indicated by the alert by sending commands to the service point application.

The commands that your automation can send to the service point application are the same ones a NetView operator can send:

LINKPD Asks the service point application to do problem determination on the specified link.

LINKTEST Asks the service point application to test the specified link.

RUNCMD Sends a command string to the service point application for execution. The data that is placed in the command string depends on the service point application and is not necessarily the same across applications. You can expand the types of commands and responses supported with RUNCMD by appropriately programming the service point application and updating your NetView automation to take advantage of the added functions. For example, actions such as retry or reconfigure can be taken only if they are supported in the service point application.

NMVTs carry alerts, commands, and responses. Refer to the Tivoli NetView (for UNIX) library for data formats and contents, and background information.

Event/Automation Service

The Event/Automation Service serves as a gateway for event data between the NetView for z/OS management environment, managers, and agents that handle Event Integration Facility (EIF) events, and SNMP managers and agents. With this gateway function, you can manage all network events from the management platform of your choice. Alerts received by the hardware monitor can be translated either to EIF events or to SNMP traps and forwarded to the respective event manager. Messages received by the NetView program can be translated to EIF events and forwarded to a designated event server. Conversely, SNMP traps or EIF events can be translated to alerts and forwarded to the hardware monitor.

For alerts, only a portion of the original alert data is forwarded to the Event/Automation Service. NetView adds information to the alert and forwards it to the Event/Automation Service. The combined information is used by the alert adapter service, confirmed alert adapter service, or the alert-to-trap service of the Event/Automation Service to create the EIF event or SNMP trap. You can customize the contents of the outgoing events or traps by customizing the information that is forwarded from the NetView program. For more information, see "Forwarding Alerts."

Messages are processed similarly. The entire message is combined with additional information created by the NetView program and is forwarded to the Event/Automation Service. The combined information is used by the message adapter service or the confirmed message adapter service of the Event/Automation Service to create the EIF event. You can customize the contents of the EIF event by customizing the information that is forwarded from the NetView program. For more information, see "Forwarding Messages" on page 405.

Forwarding Alerts

If you want to forward a hardware monitor alert without changing how the Event Integration Facility (EIF) event or SNMP trap is built, use the hardware monitor

recording filters to choose which alerts the NetView program must forward. The TECROUTE filter selects alerts for forwarding to the designated event server and the TRAPROUT filter selects alerts for forwarding to an SNMP manager. For an alert to be forwarded, either the TECROUTE or TRAPROUT filter must be set to PASS. However, an alert must pass the ESREC and AREC filters before it goes to the TECROUTE or TRAPROUT filter. You can use the SRFILTER command to specify filter settings from the hardware monitor, or you can use the SRF action to specify filter settings from the automation table.

To customize how the EIF event or SNMP trap is built when an alert is forwarded:

- Set the TECROUTE or TRAPROUT filter to PASS using either the hardware monitor SRFILTER command or the automation table SRF action.
- Write a command that performs your customization. For information of how to write the command, see the NetView samples CNMEALUS and CNMSIHSA.
- In the NetView automation table, specify the name of your command in the *cmdstring* parameter of an automation table IF-THEN statement. Add the keyword TECROUTE to the beginning of *cmdstring* as a prefix.

  Notes:

  1. Only one such prefixed command is supported for a given alert; it must handle all TECROUTE and TRAPROUT actions. This command is driven only once even if the TECROUTE and TRAPROUT filters are both passed.
  2. If you are using the confirmed alert adapter service, the *cmdstring* parameter must begin with TECROUTE, followed by the PIPE command or the name of a command that will ultimately invoke the pipe PPI stage using the TECRTCFM keyword.
- Customize the alert adapter service class definition statement file (sample IHSAACDS), the confirmed alert adapter service class definition statement file (sample IHSABCDS), or the alert-to-trap server class definition statement file (IHSALCDS).
- Customize any baroc files that have been applied to event servers. For more information, refer to the*IBM Tivoli NetView for z/OS Customization Guide*. This step is not necessary for the confirmed alert adapter service.

Forwarding Messages

To forward a message from the NetView program without changing how the EIF event is built, specify a NetView automation table IF-THEN statement with this information in *cmdstring*:

```
'PIPE SAFE * | PPI TECROUTE PPI_receiver_ID'
```

In this command, *PPI_receiver_ID* is the name of the PPI receiver associated with the Event/Automation Service. The default value is IHSATEC. Specify a value in *PPI_receiver_ID*, even if you use the default. Note that no messages are output in this example, even if the PPI stage fails. NetView sample CNMEMSUS has examples that use the secondary output of the PPI stage to output error messages.

Note: To use the confirmed message adapter, use the following information in the *cmdstring*:

```
'PIPE SAFE * | PPI TECRTCFM PPI_receiver_ID'
```

To customize how the EIF event is built when a message is forwarded to the designated event server or to handle error messages:

- Write a command that performs your customization. For information about how to write the command, refer to the NetView samples CNMEMSUS and CNMSIHSA.
- In the NetView automation table, specify the name of your command in the *cmdstring* parameter of an automation table IF-THEN statement.
- Customize the message adapter service message format file (sample IHSAMFMT) or the confirmed message adapter service message format file (sample IHSANFMT)
- Customize any baroc files that have been applied to event servers. For more information, see the*IBM Tivoli NetView for z/OS Customization Guide*. This step is not necessary for the confirmed message adapter service.

NCP Frame Relay Switching Equipment Support

NetView supports architected major vectors and subfields that contain protocol information for NCP frame relay switching equipment.

- The X'0E' subfield within the X'52' subvector contains frame relay status information.
- The X'0F' subfield within the X'52' subvector contains frame relay configuration information.

The X'0E' and the X'0F' subfields can be present in the X'1332' major vector. The X'0E' subfield can be present in the X'0000' major vector.

The frame relay vectors are not displayed by the hardware monitor, but they are passed to the automation table for processing as MSUs. An automation table statement is shipped, commented out, in the sample automation table DSITBL01. This statement can be used to conditionally run a command processor to process the frame relay information. NetView does not ship a sample command processor of this type.

Refer to the NCP library for the format of the vector and the subfields.

Chapter 28. Automation Using the Resource Object Data Manager

The Resource Object Data Manager (RODM) is an in-storage data cache that stores configuration data and resource status information. You can use RODM for both network and system automation.

Before designing an automation project that uses RODM, be familiar with RODM terminology and concepts. For more information about the object-oriented terms used by NetView to describe RODM and its data model, refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

This chapter describes automation using RODM by explaining basic concepts and providing a detailed automation scenario. The concepts are:

- Managing one or more RODM data caches
- Issuing NetView commands from RODM methods
- Verifying commands issued from RODM methods
- Accessing RODM from the NetView automation table, NetView high-level language, and assembly language programs

NetView offers a dedicated NetView task and a series of services that allow you to use RODM. With these NetView services, you can automate with RODM more easily than by using the basic RODM APIs. This set of NetView services is referred to as the ***RODM automation platform***.

An automation-in-progress indicator is maintained by NetView in RODM for resources undergoing automation. This enables operators using a NetView graphical display to wait until automation is finished for a resource before attempting to solve a problem.

You can use the RODMView tool to view and manipulate data and objects in RODM. RODMView also includes an application programming interface to RODM. Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for more information about RODMView.

Managing Multiple RODM Data Caches

RODM resides separately from the NetView application address space or the NetView subsystem address space. Multiple RODMs can reside on a single system; each RODM resides in its own address space.

Managing RODM data caches includes, connecting to, disconnecting from, and checkpointing your RODM data caches. A ***checkpoint*** is a request to save to DASD all of the current data contained in RODM. You can write applications to manage RODM data caches, or you can manage your RODM data caches from the NetView address space using the NetView DSIQTSK task.

Managing RODM Using the DSIQTSK Task

The DSIQTSK task is dedicated to communicating with the RODM address space and to managing specified RODM data caches. Each RODM that you want to manage from the NetView address space must be defined to DSIQTSK. DSIQTSK

is a NetView optional task (OPT). DSIQTSK is defined in the CNMSTYLE member and is started with the START TASK command.

In addition to managing your RODM data caches, DSIQTSK can:

- Receive commands sent by RODM
- Dispatch commands to NetView autotasks that are defined to DSIQTSK, based on the autotasks' workload

If you have more than one NetView program on a single host, each NetView program has a DSIQTSK task. Each DSIQTSK must use a unique receiver name. The DSIQTSK task automatically registers as the receiver for commands sent from RODM.

If you want DSIQTSK to receive the commands sent from RODM, use the DSIQTSKI initialization member to define a receiver name and the names of the autotasks to which the commands are dispatched.

Defining RODM Using the DSIQTSKI Initialization Member

The DSIQTSKI initialization member of DSIPARM contains keywords that define administrative details about how DSIQTSK manages your RODM data caches. These keywords define RODM attributes, autotask names, and the command receiver name that RODM uses when sending commands to DSIQTSK.

The keywords are defined briefly here. For more information, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

CMDRCVR
The name of the program-to-program interface queue that RODM uses to send commands to the NetView address space. The DSIQTSK task is a dedicated receiver for this queue. Commands sent from RODM to NetView are placed on the program-to-program interface with the RODM method called EKGSPPI. Refer to the description of the methods that are supplied with the NetView program in the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

REP The name of a RODM. Use one REP keyword for each RODM you want to manage.

These are the parameters on the REP keyword:

CONN Indicates whether RODM is to be connected to the NetView address space when DSIQTSK is activated. This parameter is entered as CONN=Y or CONN=N.

AO Indicates whether this RODM is the default RODM for the ORCONV command, the CNMQAPI high-level language service routine, and the DSINOR assembly language macro.

PASS Indicates the password or password phrase for this RODM. This password or password phrase is used when DSIQTSK attempts to connect to this RODM.

CMD Indicates a command that DSIQTSK uses when connecting to RODM.

T Indicates the amount of time that requests issued using ORCONV, CNMQAPI, or DSINOR wait if RODM is checkpointing and unavailable to process those requests. If

the wait-time expires, the requests fail with a return code indicating that a checkpoint is in progress.

ID Indicates the application ID that DSIQTSK uses to identify itself to RODM.

TASK The name of a NetView autotask.

When DSIQTSK receives a command from RODM, DSIQTSK dispatches that command to a NetView autotask. Use one TASK statement for each autotask you want to be available to DSIQTSK.

Managing RODM Using the ORCNTL Command

After you define RODM data caches in the DSIQTSKI initialization member and activate the DSIQTSK task, use the NetView ORCNTL command to:

- Connect to a specified RODM.
- Disconnect from a specified RODM.
- Change the connection password or password phrase for a specified RODM.
- Change the default RODM for the NetView ORCONV command, the CNMQAPI high-level language service routine, and the DSINOR assembly language macro.
- Initiate a checkpoint for a specified RODM.
- List the status of autotasks under the control of DSIQTSK.
- List the status of all RODM data caches managed by DSIQTSK.

For information about the syntax and usage of the ORCNTL command, refer to the NetView online help.

Issuing Commands from RODM Methods

Use the EKGSPPI method to issue commands from your RODM methods. The EKGSPPI method uses the program-to-program interface to send commands to the NetView DSIQTSK task. These commands include any command that can be run from a NetView autotask. For example, if a resource fails, you might want to trigger a method to attempt activation of that resource automatically using the VTAM VARY command. The VARY command cannot be run from the RODM address space. Therefore, the command is sent to the NetView address space. The DSIQTSK task in the NetView address space dispatches the commands to NetView autotasks for execution.

For more information, refer to the description of the methods that are supplied with the NetView program in the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

When you define RODM, include the name of the program-to-program interface queue that RODM uses to send commands to the NetView address space. The DSIQTSK task is a dedicated receiver for that queue.

Sending commands over the program-to-program interface is enabled only for RODM data caches defined to DSIQTSK.

Verifying Commands Issued from RODM Methods

After writing a RODM method that triggers the EKGSPPI method to send commands to DSIQTSK, test your method without actually executing the commands. Instead of dispatching the commands to an autotask, DSIQTSK can display the commands as messages and enable an operator to edit, discard, or issue the commands for actual execution. This is called issuing the commands in ***assist mode***.

To help you use assist mode:

- The ASSIST parameter of the EKGSPPI method

  In your method, you can pass the ASSIST parameter to EKGSPPI to specify that EKGSPPI is to issue commands in assist mode. If a command is issued in assist mode, DSIQTSK converts the command to a message (message DWO670I) rather than executing the command.

  Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for information about calling EKGSPPI.

- The SAVECMD command

  In the automation table, use the SAVECMD command to route message DWO670I to an operator. SAVECMD saves command and text information for the ASSISCMD command.

  The ASSISCMD command is invoked by an operator. ASSISCMD uses the NetView VIEW facility to create a full-screen panel of the commands and text stored by SAVECMD. The operator can then approve, change, or discard the commands.

  The SAVECMD command list is run when the DSIQTSK task receives automation message DWO670I. Use online command help for the correct format of the SAVECMD command.

  Figure 145 shows how to use the SAVECMD command list in the automation table:

```
IF MSGID='DWO670I' THEN
EXEC(CMD('SAVECMD') ROUTE(ONE NETOP1));
```

Figure 145. Using the SAVECMD Command List in the Automation Table

- The ASSISCMD command

  After a command is routed to an operator, the operator sees message CNM436I. Subsequent commands routed to the same operator are put in a queue for that operator and message CNM436I is not displayed. After the operator has processed all the saved commands and the queue is empty, message CNM436I is displayed for the next command routed to the operator.

  Message CNM436I indicates that the operator is to enter ASSISCMD. The operator can use ASSISCMD to manipulate the commands saved by SAVECMD. Using ASSISCMD, the operator can:
  - Delete the command
  - Edit and reissue the command
  - Run the command as it is

  Refer to the NetView online help for the syntax of the ASSISCMD command.

Accessing RODM from NetView

RODM data caches that are managed by the DSIQTSK task can be accessed using three methods:

ORCONV A NetView command
CNMQAPI A high-level language service routine
DSINOR An assembly language macro

CNMQAPI and DSINOR can be issued from command processors and installation exits. You can use the ORCONV command and the application programming interfaces only with RODM data caches managed by DSIQTSK. If you are managing RODM data caches in some other way, use the EKGUAPI application programming interface to access RODM.

The ORCONV Command

The ORCONV command changes fields and invokes methods in RODM from the following sources:

- The NetView automation table
- Command lists
- The command facility
- Procedures written in REXX, PL/I, or C

Refer to the online command help for parameter specifications and format of the ORCONV command.

Accessing RODM from High-Level Language and assembly language Programs

The high-level language (HLL) service routine, CNMQAPI, and the assembly language macro, DSINOR, are intended to be issued from HLL and assembler programs, respectively. CNMQAPI and DSINOR use the native RODM application programming interface, EKGUAPI. CNMQAPI and DSINOR can be used only on RODM data caches defined to DSIQTSK.

Refer to *IBM Tivoli NetView for z/OS Programming: PL/I and C* and to *IBM Tivoli NetView for z/OS Programming: Assembler* for more information about CNMQAPI and DSINOR.

A RODM Automation Scenario

This automation scenario incorporates the concepts and functions discussed in this chapter into a working example. You can see how the automation platform can be used to:

- Manage RODM
- Manipulate data in RODM
- Automate the recovery of a failing resource
- Dispatch work to autotasks
- Verify commands issued from RODM methods

This scenario has five parts:

- An outline of the scenario events
- A description of the various entities (such as RODM names, RODM classes, resources, and operator IDs) used in the scenario
- Steps to set up the scenario
- Steps for executing the scenario
- Excerpts, with explanations, from key sections of the change method (EKGCPPI) used in the scenario

The Scenario Events

These are major events in this scenario:

1. The DSIQTSK task automatically connects to two RODM data caches, based on the RODM definitions in the DSIQTSKI initialization member.
2. The ORCNTL command is issued to change the default RODM.
3. A network resource, A01A704, fails.
4. VTAM issues an IST105I message indicating that the resource has failed.
5. The IST105I message is trapped in the automation table.
6. As a result of the IST105I message, the automation table issues the ORCONV command to change the status of the resource in RODM.
7. The RODM change method invoked as a result of the status change checks a field called AOLEVEL to determine whether to issue any automation commands for this resource. In this scenario, the AOLEVEL field indicates that automation commands are issued in assist mode.
8. The RODM method sends a VTAM command to DSIQTSK to activate the resource. This command is issued in assist mode, using the EKGSPPI method. Refer to the description of the methods that are supplied with the NetView program in the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for a detailed description of EKGSPPI.
9. Because the command was issued in assist mode, DSIQTSK does not dispatch the VTAM command to an autotask. Instead, the command is saved. An operator can issue the ASSISCMD command to edit and re-issue, discard, or issue the command as displayed.

The Scenario Entities

The scenario refers to these entities:

Entity	Description
RODM1	Defined in the DSIQTSKI initialization member as the default RODM. The default RODM is the RODM that CNMQAPI, DSINOR, and ORCONV act on.
RODM2	Defined in the DSIQTSKI initialization member as an additional RODM.
TERMINAL	The name of a class contained in RODM2.
A01A704	The identifier of an object, of class TERMINAL, in RODM2. This object is a locally attached logical unit (LU) in the network.
STATUS	The name of a field in object A01A704. This field contains the status (UP or DOWN) of the resource. A RODM change method, EKGCPPI, is associated with this field. This method attempts to reactivate A01A704 when the status changes to DOWN.
AOLEVEL	The name of a field in object A01A704. This field contains a numeric value: 1, 2, or 3. In this scenario, the AOLEVEL field is used to determine whether automation commands associated with a resource are issued. This field is also used to determine whether assist mode is used. **1** The RODM method issues a VTAM command to activate the resource. The IST105I message that notified NetView that the resource was down does not receive any special handling.

2 The IST105I message that notified NetView that the resource was down is routed to an operator as specified by the ORCONV command, and RODM does not attempt to activate the resource.

3 The RODM method issues a VTAM command, in assist mode, to activate the resource, and the IST105I message that notified NetView that the resource was down does not receive any special handling.

CNM01 The name of the command receiver queue that RODM uses to send commands to DSIQTSK.

AUTO1, AUTO2, AUTO3
Three autotasks to which DSIQTSK dispatches commands.

Setting Up the Scenario

To set up the scenario:

1. Identify the "failing resource" for the scenario.

   This scenario uses resource A01A704 as its failing resource. You need to define A01A704 to VTAM, or decide to use another resource in your network and substitute your resource name for A01A704 in this scenario.
2. If you do not have an operator ID called NETOP1, define NETOP1 or substitute one of your operator IDs throughout this scenario. Be sure to include the operator ID on the MSGPARMS parameter of the ORCONV command.
3. Create and start three autotasks, using the AUTOTASK command. This scenario uses the names AUTO1, AUTO2, and AUTO3. If you do not have automated operators named AUTO1, AUTO2, and AUTO3 define them, or substitute three of your autotask names throughout this scenario.
4. Create an automation table called DSITBL01. You can also rename your automation table to DSITBL01, or you can substitute the name of your automation table throughout this scenario.
5. Define your RODM data caches to DSIQTSK.

   The DSIQTSKI initialization member, you can define various administrative details about the RODM data caches you want to manage. The DSIQTSKI member is located in DSIPARM. This sample DSIQTSKI member defines:

   CNM01
   A command receiver queue name.

   RODM1 and RODM2
   Two RODM data caches to be managed. RODM1 is the default.

   AUTO1, AUTO2, and AUTO3
   Three autotasks to which DSIQTSK can dispatch work.

   Figure 146 on page 414 shows the DSIQTSKI initialization member. Any row beginning with an asterisk is treated as a comment. Only one keyword can be defined on each line. For example, notice that the TASK statements for AUTO1, AUTO2, and AUTO3 are not on the same line.

```
*
*  Define the PPI command receiver, and make it APF-authorized.
*
CMDRCVR ID=CNM01
*
*  Define two resource object data managers (RODMs) to be managed
*  by the DSIQTSK.  RODM1 is the default RODM.  Both RODMs will
*  be connected automatically (using the password) when the DSIQTSK
*  task is started.
*
REP  RODM1,CONN=Y,AO=Y,PASS=PASSWORD,T=300,ID=APPL1
REP  RODM2,CONN=Y,AO=N,PASS=PASSWORD,T=300,ID=APPL2
*
*  Define three autotasks to which the DSIQTSK can dispatch work.
*
TASK AUTO1
TASK AUTO2
TASK AUTO3
*
```

Figure 146. Sample DSIQTSKI Initialization Member for the DSIQTSK Task

6. Add an automation table statement to DSITBL01. This statement traps message IST105I and issues the ORCONV command to RODM1 to change the resource status to DOWN. Figure 147 shows the automation table statement that accomplishes this.

```
IF MSGID = 'IST105I' THEN
  BEGIN;
   IF TOKEN(2) = 'A01A704' THEN
      EXEC(CMD('ORCONV TYPE=CHANGE,
      CLASS=TERMINAL,OBJECT=A01A704,FIELD=STATUS,DATA='DOWN',
      MSGFIELD='AOLEVEL',MSGPARMS='NETOP1,HELD=Y'')
      ROUTE(ONE NETOP1)) DISPLAY(Y);
  END;
```

Figure 147. Automation Table Statement to Trap IST105I and Issue ORCONV Command

   DISPLAY is set to Y because this message needs to be displayed to the operator, NETOP1. If DISPLAY were set to N, the message is never displayed anywhere (even if the ORCONV command specifies a destination).

7. Create a RODM change method to attempt the recovery of the failed resource, A01A704.

   A programmer creates a method that uses the EKGSPPI method to send the VARY NET,ACT command to DSIQTSK. The DSIQTSK task receives this command and dispatches it to one of the autotasks defined in DSIQTSKI.

   In this scenario, the method that calls EKGSPPI is the EKGCPPI method. EKGCPPI is written in PL/I. "Key Sections of Change Method EKGCPPI" on page 419 presents excerpts from EKGCPPI.

   The EKGCPPI method checks the AOLEVEL field to determine whether the VARY NET,ACT command is issued in assist mode. In this scenario, AOLEVEL is set to 3, so commands are issued in assist mode.

   Because the AOLEVEL field indicates assist mode, EKGCPPI requests assist mode when calling the EKGSPPI method. Assist mode means that any command issued by the RODM method is not run. Instead, this command is trapped by DSIQTSK and issued as NetView message DWO670I. You can trap this message in the automation table and save it using the SAVECMD command. Commands saved using SAVECMD can be displayed and manipulated using ASSISCMD. See step 9 on page 417 for an example of using ASSISCMD.

Without assist mode, DSIQTSK would have dispatched the command to autotask AUTO1 to be run.

After the change method is created, it must be compiled and link-edited into one of the libraries specified with the STEPLIB data definition (DD) statement of the RODM START procedure.

8. Create an automation table statement in DSITBL01. This statement traps message DWO670I and saves the command passed from RODM.

   Figure 148 shows the automation table statement that accomplishes this. This statement traps the DWO670I message and uses the SAVECMD command to save the VTAM VARY command issued by the RODM method. In this scenario, the SAVECMD is then routed to NetView operator NETOP1, if NETOP1 is logged on.

   When the SAVECMD command is routed to NETOP1, NETOP1 receives message CNM436I. NETOP1 can then use the ASSISCMD to edit, discard, or issue the saved command.

```
IF MSGID = 'DWO670I' THEN
   EXEC(CMD('SAVECMD')
        ROUTE(ONE NETOP1))
   DISPLAY(N) NETLOG(Y);
```

Figure 148. Sample Automation Table Statement to Trap DWO670I

9. Create an input file for the RODM loader to:
   - Install the EKGCPPI method and the EKGSPPI method.
   - Create the classes and fields used in the scenario.

   Figure 149 shows the input file used for this scenario. Lines beginning with the characters "--" are comment lines.

```
-- Install EKGSPPI and EKGCPPI Method
   OP EKG_Method HAS_INSTANCE EKGSPPI;
   OP EKG_Method HAS_INSTANCE EKGCPPI;
-- Create the class called TERMINAL under UniversalClass
   OP TERMINAL HAS_PARENT UniversalClass;
-- Create fields and subfields for Class TERMINAL
   OP TERMINAL HAS_FIELD (INTEGER) AOLEVEL;
   OP TERMINAL HAS_FIELD (CHARVAR) STATUS;
   OP TERMINAL.STATUS HAS_SUBFIELD CHANGE;
-- Create object instance A01A704 for class TERMINAL
   OP TERMINAL HAS_INSTANCE A01A704;
-- Set the value for the fields and subfield for the Object
   OP TERMINAL.A01A704.STATUS HAS_VALUE (CHARVAR) 'UP';
   OP TERMINAL.A01A704.AOLEVEL HAS_VALUE (INTEGER) 3;
   OP TERMINAL.A01A704.STATUS.CHANGE SUBFIELD_HAS_VALUE
      (METHODSPEC) ('EKGCPPI');
```

Figure 149. Input File for RODM Loader

Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for more information about the RODM load function.

Running the Scenario

To run the scenario:

1. Start RODM1 and RODM2. Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for information about starting multiple RODM data caches.
2. Run the RODM loader with the input file you created to load RODM2. This step:

- Installs the EKGCPPI method and the EKGSPPI method
- Creates the TERMINAL class, the A01A704 object, and the STATUS field in RODM2

Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for information about loading RODM and installing methods.

3. Log on to NetView as NETOP1.
4. Start the DSIQTSK task. Issue START TASK=DSIQTSK from a NetView operator panel, unless this task was started when NetView was initialized. When DSIQTSK starts, it automatically connects to RODM1 and RODM2, with RODM1 as the default RODM. The default RODM is also referred to as ***the current run-time RODM***. The CNMQAPI service routine, the DSINOR macro, and the ORCONV command act on RODM1, the default RODM, rather than RODM2.

   Note: DSIQTSK can connect to a RODM only if that RODM is active.
5. Change the default RODM. Because the class, object, and field this scenario acts on are in RODM2, you need to make RODM2 the default RODM. To change the default, use the command in Figure 150.

```
ORCNTL CHNG,OR=RODM2
```

Figure 150. Changing the Default RODM

   Now the ORCONV command you coded in the automation table acts on RODM2.
6. Activate the automation table. To start the automation table, use the command in Figure 151.

```
AUTOTBL MEMBER=DSITBL01
```

Figure 151. Activating the Automation Table

   If an IST105I message is received for A01A704, the ORCONV command is issued to change its status to DOWN. When the status is changed to DOWN, a change method is invoked. The change method checks AOLEVEL. The AOLEVEL field is set to 3, so the VTAM VARY command is issued in assist mode rather than being dispatched to an autotask for execution.
7. Use the DEFAULTS command to ensure that an operator is notified of a failed resource if recovery of that resource is not automated.

   The DEFAULTS command has a parameter called SENDMSG. You can use this parameter, in combination with the MSGFIELD and MSGPARMS parameters of the ORCONV command, to determine what to do with the IST105I message that caused the ORCONV command to be issued.

   In the scenario, the RODM change method checks the AOLEVEL field to determine whether to automate recovery of the failed resource. If recovery is not automated, an operator needs to be notified that the resource has failed. However, the RODM change method does not have access to this IST105I message. Instead, you can use the ORCONV command to examine the AOLEVEL field just as the change method did. Based on the value of this field, the ORCONV command routes the IST105I message to an operator you specify with the ORCONV command.

   First, set the SENDMSG parameter to a numeric value (or list of values). In this scenario, SENDMSG is set to 2. From the operator panel, enter the

command in Figure 152.

```
DEFAULTS SENDMSG=2
```

Figure 152. Setting the DEFAULT SENDMSG Parameter

The MSGFIELD parameter tells the ORCONV command to compare the value set by SENDMSG to the value of AOLEVEL (see Figure 147 on page 414). The ORCONV command compares the value of AOLEVEL to the value set by SENDMSG.

If the value of AOLEVEL matches one of the values set by the DEFAULTS SENDMSG command, the message that caused the ORCONV command to be issued is routed to the destination defined with the MSGPARMS parameter of ORCONV. In step 6 on page 414, the ORCONV command is issued as a result of an IST105I message. Also, Figure 147 on page 414 shows that MSGFIELD is set to AOLEVEL and MSGPARMS is set to NETOP1. Therefore, the IST105I message is routed to the operator NETOP1 if AOLEVEL is set to 2.

If the value of AOLEVEL does not match one of the values set by the DEFAULTS SENDMSG command, the IST105I message is not routed to the destination specified by MSGPARMS. If AOLEVEL were set to 1, the RODM method changes the status of A01A704 to DOWN, and the IST105I message is not routed to the destination specified by MSGPARMS. The RODM method attempts to automate the recovery of the resource.

8. Deactivate the resource A01A704.

   To continue this scenario, deactivate a resource. From NetView, issue the command in Figure 153.

```
V NET,INACT,ID=A01A704,F
```

Figure 153. Example of Inactivating Resource A01A704

9. After the resource is inactivated, you see message CNM436I. This message indicates that you need to enter the ASSISCMD command. Figure 154 shows the first ASSISCMD panel.

```
assispn1             Commands for Operator Assistance

 1.            V NET,ACT,ID=A01A704
          MORE This command was sent by the change method EKGCPPI to act
 2.
 3.
 4.
 5.
 6.









 Command===>
  PF1 = Help              PF2 = Exit
  PF6 = Roll
```

Figure 154. Example Screen for the ASSISCMD Command

If the command had not been issued in assist mode, the CNM436I message was not received and the VARY NET,ACT command would have been issued without being displayed to the operator.

The first panel displayed by the ASSISCMD command shows the last six commands issued in assist mode. Any informational text associated with the commands is also displayed. This informational text displayed is the same text that is associated with the command when calling the EKGSPPI method, as shown in Figure 161 on page 424. As each of the commands is processed by the operator, the next saved command is displayed on the panel. Up to 20 commands can be queued for display. This number can be changed by modifying the SAVECMD command list.

10. After the first ASSISCMD panel appears, the operator can type one of these letters next to the command on the panel:

    E Run the command as it is displayed. This option can be entered from the first or second panel.

    D Delete the command. This option can be entered on the first or second panel.

    M Modify the command, or display more information. If there is more information to be displayed, the word MORE appears on the first panel as the first word on the line immediately following the command. The M option displays the second ASSISCMD panel. The operator can view the entire command and any informational text associated with the command.

    In Figure 154 on page 417, the word MORE appears under the command. This word indicates that more details about the command are available.

11. Enter **M** next to the command as shown in Figure 155.

```
assispn1              Commands for Operator Assistance

 1.  M         V NET,ACT,ID=A01A704
         MORE This command was sent by the change method EKGCPPI to act
 2.
 3.
 4.
 5.
 6.









 Command===>
  PF1 = Help              PF2 = Exit
  PF6 = Roll
```

Figure 155. Example Screen for the ASSISCMD Command--Enter M for More Detail

12. After you enter **M**, the second ASSISCMD panel, shown in Figure 156 on page 419, is displayed. This panel provides an explanation of why the command was issued. This text was created in the change method EKGCPPI (see Figure 161 on page 424) and passed to the method EKGSPPI as a parameter.

```
assispn2          Full Detail of Command for Operator Assistance

       V NET,ACT,ID=A01A704

  This command was sent by the change method EKGCPPI to activate a
  resource.  This command was sent because both of the following
  conditions have occurred:  (1) The status of the resource has been
  set to DOWN in RODM.  (2) RODM indicates that recovery of this
  resource should be attempted automatically.









 Command===>
  PF1 = Help              PF2 = Exit          PF3 = Previous Panel
  PF6 = Roll
```

Figure 156. Example Screen for the ASSISCMD Command--More Detail About Command

13. To process the command, enter **E** next to the command, as shown in Figure 157. If you want to edit the command before executing it, type over the command, and then type **E** to run the command.

```
assispn2          Full Detail of Command for Operator Assistance

  E    V NET,ACT,ID=A01A704

  This command was sent by the change method EKGCPPI to activate a
  resource.  This command was sent because both of the following
  conditions have occurred:  (1) The status of the resource has been
  set to DOWN in RODM.  (2) RODM indicates that recovery of this
  resource should be attempted automatically.









 Command===>
  PF1 = Help              PF2 = Exit          PF3 = Previous Panel
  PF6 = Roll
```

Figure 157. Example Screen for the ASSISCMD Command--Enter E to Execute Command

Key Sections of Change Method EKGCPPI

This section describes selected parts of the change method EKGCPPI. EKGCPPI is a NetView sample on the distribution tape.

These excerpts from EKGCPPI are intended to help you understand the change method. Examine the entire method in addition to these excerpts. Each excerpt is followed by explanations of some of the fields or statements. For information about writing change methods, refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide*.

Procedure Statement

```
  ⋮
EKGCPPI: PROCEDURE (IN_FLD_ID,IN_LLP,IN_SLP,IN_DATATYPE,
                    IN_CHARLEN, IN_DATAPTR)
         OPTIONS (REENTRANT); 1
```

Figure 158. Procedure Statement for Change Method EKGCPPI

Key	Explanation
1	Because this is a change method, it is responsible for physically making the change to the VALUE subfield of the STATUS field in RODM. To make this change, the method needs to know which field to change and what the new value is for that field. This information is passed to the change method by RODM, and must be defined as parameters on the method's procedure statement. In this scenario, the ORCONV command attempted to change the value of the STATUS field to DOWN. However, because the change method EKGCPPI is associated with the STATUS field, RODM triggers the EKGCPPI method and passes the name of the field and the new value to EKGCPPI.

Local Variables

```
/*****************************************************************/
/*                                                               */
/*                       LOCAL VARIABLES                         */
/*                                                               */
/*****************************************************************/
  ⋮
                                     /* Selfdefining data for      */
                                  /*  OI method EKGSPPI 1   */
DCL 1 EKGSPPI_BLK UNALIGNED,
      3 TOTAL_LEN  Smallint,           /* Not including its length   */
      3 RCVRID_CHARVAR, 2       /* Receiver id CharVar        */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CHAR_LEN),     /* CharVar len */
        5 CHAR_DATA  CHAR(8) INIT('CNM01'),           /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B),       /* Null data   */
      3 ASSIST_CHARVAR, 3      /* Assist information CharVar  */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CHAR_LEN),     /* CharVar len */
        5 CHAR_DATA  CHAR(8),                         /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B),       /* Null data   */
      3 TASKINFO_CHARVAR, 4    /* Task information CharVar    */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CHAR_LEN),     /* CharVar len */
        5 CHAR_DATA  CHAR(8) INIT('ONLYANY'),         /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B),       /* Null data   */
      3 TASKNAME_CHARVAR, 5    /* Task name CharVar          */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CHAR_LEN),     /* CharVar len */
        5 CHAR_DATA  CHAR(8) INIT('AUTO1'),           /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B),       /* Null data   */
      3 SENDER_CHARVAR, 6      /* Sender token CharVar       */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CHAR_LEN),     /* CharVar len */
        5 CHAR_DATA  CHAR(8) INIT('EKGCPPI'),         /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B),       /* Null data   */
      3 CMD_CHARVAR, 7         /* Command Charvar            */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CMD_LEN),      /* CharVar len */
        5 CHAR_DATA  CHAR(MAX_CMD_LEN),               /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B),       /* Null data   */
      3 CMD_DESC_CHARVAR, 8    /* Command Description Charvar */
        5 DATA_TYPE  Smallint INIT(EKG_DT_CharVar),   /* Data type   */
        5 CHAR_LEN   Smallint INIT(MAX_CMD_DESC_LEN), /* CharVar len */
        5 CHAR_DATA  CHAR(MAX_CMD_DESC_LEN),          /* CharVar data*/
        5 NULL_DATA  BIT(8) INIT ('00000000'B);       /* Null data   */
  ⋮
```

Figure 159. Local Variables for Change Method EKGCPPI

Key	Explanation
1	EKGSPPI is the method that places commands on the program-to-program interface to send the commands to the NetView task DSIQTSK. This self-defining string contains seven parameters that are passed to the object-independent method EKGSPPI. All leading blanks are deleted from these input parameters before they are processed.
2	The RCVRID_CHARVAR statement defines the command receiver name as CNM01. This is the receiver name that EKGSPPI uses when sending commands to DSIQTSK over the program-to-program interface. This is the same receiver name defined in the DSIQTSKI initialization member.
3	The ASSIST_CHARVAR statement defines the variable that passes either

ASSIST or NOASSIST to EKGSPPI. When the value of this variable is ASSIST, any commands sent to DSIQTSK are issued in assist mode. In this scenario, the value of this variable is based on the value of the AOLEVEL field in RODM.

4 The TASKINFO_CHARVAR statement specifies whether DSIQTSK dispatches commands to a specific autotask, or to the next available autotask. Its CHAR_DATA statement has the following attributes:

Attribute	Meaning
ONLYANY	A specific autotask is used, unless that autotask is not available. If it is not available, the next available autotask (after the most recently used autotask) defined to DSIQTSK issues the command. Autotasks are used in the order in which they are defined in the DSIQTSKI member of DSIPARM.
ONLY	A specific autotask is used. If this autotask is busy, the command is queued for the autotask. If the specified autotask is not available, the command is not issued.
ANY	The next autotask (after the most recently used autotask) defined to DSIQTSK issues the command. Autotasks are used in the order in which they are defined in the DSIQTSKI member of DSIPARM.

5 The TASKNAME_CHARVAR statement specifies that DSIQTSK dispatches the command specified in CMD_CHARVAR to autotask AUTO1.

6 The SENDER_CHARVAR statement identifies the method that is sending a command to EKGSPPI. In this scenario EKGCPPI is used as the identifier or ***sender token***.

7 The CMD_CHARVAR statement defines the variable that contains the name of the command passed from EKGSPPI to DSIQTSK.

8 The CMD_DESC_CHARVAR statement defines the variable that contains text describing the command sent from EKGSPPI to DSIQTSK. This text is displayed if the command is issued in assist mode and an operator enters ASSISCMD.

Constants

```
/*********************************************************************/
/*                                                                   */
/*                     CONSTANTS                                     */
/*                                                                   */
/*********************************************************************/
   ⋮
DCL $ASSISTON  FIXED BIN(31) INIT(3); 1  /* Send cmd to DSIQTSK*/
DCL $ASSISTOFF FIXED BIN(31) INIT(1); 2  /* Send cmd to DSIQTSK*/
   ⋮
DCL FIELD_AO     CHAR(7) INIT('AOLEVEL'); 3  /* Field name for AOLEVEL */
DCL FIELD_MyName CHAR(6) INIT('MyName'); 4   /* MyName Field           */
DCL EKGSPPI_NAME CHAR(7) INIT('EKGSPPI'); 5  /* EKGSPPI OI method name */
                                     /* CMD Value                */
DCL CMD_VALUE CHAR(MAX_CMD_LEN - RESOURCE_NAME_SIZE) VARYING
     INIT('V NET,ACT,ID='); 6
                                     /* CMD description          */
DCL CMD_DESC_VALUE CHAR(MAX_CMD_DESC_LEN) VARYING; 7
   ⋮
```

Figure 160. Constants for Change Method EKGCPPI

Key	Explanation
1	The $ASSISTON constant is used to determine whether the AOLEVEL field is set to 3. The value 3 indicates that the object-independent method EKGSPPI issues commands in assist mode.
2	The $ASSISTOFF constant is used to determine whether the AOLEVEL field is set to 1. A value of 1 indicates that the object-independent method EKGSPPI sends commands to DSIQTSK without assist mode. That is, the commands are sent to DSIQTSK, dispatched to an autotask, and processed.
3	The FIELD_AO constant contains the name of the field that EKGCPPI checks to determine if commands are issued in assist mode. In this scenario, the field is AOLEVEL.
4	The FIELD_MyName constant contains the name of the field that EKGCPPI queries to find out the name of the object. The field is MyName.
5	The EKGSPPI_NAME constant contains the name of the object-independent method that this change method (EKGCPPI) triggers to send commands to DSIQTSK over the program-to-program interface. In this scenario, the method is EKGSPPI.
6	The CMD_VALUE constant contains the actual command that EKGSPPI sends to DSIQTSK over the program-to-program interface. In this scenario, the command is V NET,ACT,ID=*opid*. Later in this method, the resource name is determined and concatenated to this command string.
7	The CMD_DESC_VALUE constant contains the descriptive text associated with the command. This is the text that is displayed when an operator enters ASSISCMD. The value of this constant is not initialized. It is assigned later in the method.

Initialization

```
/*********************************************************************/
/*                      Initialization                               */
/*********************************************************************/
                                     /* Set change subfield fid    */
F1403_FUNC_BLK.Function_ID = EKG_ChangeSubfield; 1
                                     /* Trigger OI method func id  */
F1416_FUNC_BLK.Function_ID = EKG_TriggerOIMethod; 2
                                     /* Set the query subfield fid */
F1502_FUNC_BLK.Function_ID = EKG_QuerySubfield; 3
   :
                                     /* Set the cmd desc value     */
CMD_DESC_VALUE = 'This command was sent by the change method ' 4
         ||      'EKGCPPI to activate a resource.  '
         ||      'This command was sent because both of the '
         ||      'following conditions have occurred:  '
         ||      '(1) The status of the resource has been set '
         ||      'to DOWN in RODM.  '
         ||      '(2) RODM indicates that recovery of this '
         ||      'resource should be attempted automatically. ';
```

Figure 161. Initialization of Change Method EKGCPPI

Figure 161 shows the part of the method that assigns a value to the function identifier fields in the RODM function blocks that is used throughout this method. These functions are used in this method:

- Change a subfield
- Trigger an object-independent method
- Query a subfield

These are standard RODM functions. Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for a description of these RODM functions.

Key	Explanation
1	The F1403_FUNC_BLK statement assigns a function identifier for changing a subfield.
2	The F1416_FUNC_BLK statement assigns a function identifier for triggering an object-independent method. In this scenario, the method started is EKGSPPI.
3	The F1502_FUNC_BLK statement assigns a function identifier for querying a subfield. In this scenario, the EKGCPPI method queries two fields: • The VALUE subfield of the AOLEVEL field • The VALUE subfield of the MyName field
4	The CMD_DESC_VALUE statement defines the text that is associated with the command defined by the CMD_VALUE constant in Figure 160 on page 423.

Changing a Subfield

```
/*********************************************************************/
/* Change the value subfield of the input                            */
/*********************************************************************/
   ⋮
                                     /* Set datatype of change fld */
F1403_FUNC_BLK.Data_type = IN_DATATYPE; 1
                                     /* Set char data length       */
F1403_FUNC_BLK.New_char_data_length = IN_CHARLEN; 2
                                     /* Set new data pointer       */
F1403_FUNC_BLK.New_data_ptr = IN_DATAPTR; 3
   ⋮
                                     /* Change subfield            */
CALL EKGMAPI(TRANS_INFO_BLK,F1403_FUNC_BLK,RESPONSE_BLK); 4
   ⋮
```

Figure 162. Changing a Subfield with Change Method EKGCPPI

Figure 162 shows the part of the method that changes the value of the STATUS field. The STATUS field is the field with which the method is associated.

Key	Explanation
1	IN_DATATYPE specifies the data type of the STATUS field. This parameter is specified on the procedure statement in Figure 158 on page 420.
2	IN_CHARLEN specifies the length of the new data. In this scenario, the new value is DOWN. This parameter is specified on the procedure statement in Figure 158 on page 420.
3	IN_DATAPTR specifies the pointer to the new data. This parameter is specified on the procedure statement in Figure 158 on page 420.
4	This statement calls the RODM API EKGMAPI. EKGMAPI performs the actual change to the status field. Function identifier 1403 is defined as `EKG_ChangeSubfield` in Figure 161 on page 424.

Querying a Field

```
   /*********************************************************************/
   /*         Query the AOLEVEL field to see it is 1                    */
   /*********************************************************************/
   ⋮
   ⋮
                                /* Length is set              */
FIELD_ACCESS_INFO_BLK.Field_name_length = LENGTH(FIELD_AO); 1
                                /* Set the field ptr          */
FIELD_PTR = ADDR(FIELD_AO); 2
   ⋮
                                /* Query a AOLEVEL subfield   */
CALL EKGMAPI(TRANS_INFO_BLK,F1502_FUNC_BLK,RESPONSE_BLK); 3
   ⋮
                                /* Send the command or not    */
IF TEMP_DATA_VALUE = $ASSISTON |
   TEMP_DATA_VALUE = $ASSISTOFF  THEN
DO; 4                    /* Prepare to send command     */
                               /* Is ASSIST ON               */
IF TEMP_DATA_VALUE = $ASSISTON THEN
   EKGSPPI_BLK.ASSIST_CHARVAR.CHAR_DATA = 'ASSIST'; 5
ELSE
   EKGSPPI_BLK.ASSIST_CHARVAR.CHAR_DATA = 'NOASSIST'; 6
```

Figure 163. Querying a Field with Change Method EKGCPPI

Figure 163 on page 425 shows the part of the method that determines whether the object-independent method, EKGSPPI, issues commands in assist mode. The change method, EKGCPPI, examines the AOLEVEL subfield associated with the failed resource.

- If the value of AOLEVEL is 1 (the constant $ASSISTOFF), EKGSPPI is called with the ASSIST option.
- If the value of AOLEVEL is 3 (the constant $ASSISTON), EKGSPPI is called with the NOASSIST option.
- For any other value of AOLEVEL, EKGSPPI is not called.

Key **Explanation**

1 This statement defines the length of the field name.

2 This statement defines the address of the AOLEVEL field name. FIELD_AO was given the value AOLEVEL in Figure 160 on page 423.

3 This statement calls EKGMAPI to query the AOLEVEL field. Function identifier 1502 was defined as `EKG_QuerySubfield` in Figure 161 on page 424.

4 The TEMP_DATA_VALUE was set based on the result of the subfield query. This IF statement checks the value of TEMP_DATA_VALUE to determine whether commands are sent to DSIQTSK with assist mode or without assist mode.

5 This statement determines whether the value of TEMP_DATA_VALUE is $ASSISTON. If so, the ASSIST_CHARVAR variable (see Figure 159 on page 421) is set to ASSIST. The command sent to DSIQTSK is issued in assist mode. An operator can use the ASSISCMD command to display, modify, and issue the commands.

6 This statement determines whether the value of TEMP_DATA_VALUE is $ASSISTOFF. If so, the ASSIST_CHARVAR variable (see Figure 159 on page 421) is set to NOASSIST. The command sent to DSIQTSK is dispatched to an autotask for execution; assist mode is not used.

Querying an Object Name

```
/*********************************************************/
/*   Query the Object name for V NET,ACT,ID=objectname   */
/*********************************************************/
  ⋮
                            /* Length is set             */
FIELD_ACCESS_INFO_BLK.Field_name_length
       = LENGTH(FIELD_MyName); 1
                            /* Set the field ptr         */
FIELD_PTR = ADDR(FIELD_MyName); 2
  ⋮
                            /* Query a Object name subfld */
CALL EKGMAPI(TRANS_INFO_BLK,F1502_FUNC_BLK,RESPONSE_BLK); 3
  ⋮
```

Figure 164. Querying an Object Name with Change Method EKGCPPI

This section of the method determines the object name for the failed resource. In this scenario, the object name is the resource name. This name is then concatenated with the value of CMD_VALUE and sent to EKGSPPI.

Key **Explanation**

1 This statement defines the length of the field name.

2 This statement defines the address of the MyName field name. (FIELD_MyName was given the value MyName in Figure 160 on page 423.)

3 This statement calls EKGMAPI to query the VALUE subfield of the MyName field. Function identifier 1502 was defined as `EKG_QuerySubfield` in Figure 161 on page 424.

Triggering an Object-Independent Method

```
/*****************************************************/
/*   Prepare to trigger OI method EKGSPPI to send the */
/*   command V NET,ACT,ID=objectname to DSIQTSK       */
/*****************************************************/
   ⋮
                                      /* Set the cmd with           */
                                      /*  MyName field value        */
EKGSPPI_BLK.CMD_CHARVAR.CHAR_DATA =
    CMD_VALUE || SUBSTR(CHARVAR_RESP_BLK.CHAR_DATA,
                        1,CHARVAR_RESP_BLK.CHAR_LEN); 1
                                      /* Set the cmd description    */
EKGSPPI_BLK.CMD_DESC_CHARVAR.CHAR_DATA =
    CMD_DESC_VALUE; 2
                                      /* Set OI method name         */
F1416_FUNC_BLK.Method_name = EKGSPPI_NAME; 3
                                      /* Set selfdefining parm      */
F1416_FUNC_BLK.Method_parms = ADDR(EKGSPPI_BLK); 4
   ⋮
                                      /* Message triggered MAPI call*/
CALL EKGMAPI(TRANS_INFO_BLK,F2009_FUNC_BLK,
             RESPONSE_BLK); 5
   ⋮
```

Figure 165. Triggering an Object-Independent Method with Change Method EKGCPPI

This section of the method triggers the object-independent method EKGSPPI. EKGSPPI sends the command defined in CMD_CHARVAR to DSIQTSK, over the program-to-program interface.

Key **Explanation**

1 This statement concatenates the command to be issued with the resource name. The complete command is then put into the variable CMD_CHARVAR, which is part of the self-defining string sent to EKGSPPI.

2 This statement puts the text associated with the command (see Figure 161 on page 424) into the variable CMD_DESC_CHARVAR, which is part of the self-defining string sent to EKGSPPI.

3 This statement defines EKGSPPI as the name of the object-independent method to be triggered.

4 This statement defines the address of the parameter list (a self- defining string) for the object-independent method EKGSPPI.

5 This statement calls EKGMAPI to start EKGSPPI.

Chapter 29. Automation Using the Terminal Access Facility

The NetView terminal access facility (TAF) is a VTAM relay function that permits a NetView operator's terminal to appear as an LU1 or LU2 type terminal to any application supporting those protocols. The applications include, but are not limited to, the Customer Information Control System (CICS) and Information Management System (IMS). TAF enables both NetView operators and autotasks to view messages and issue commands as if they were logged on to the subsystem console or master terminal for that application. Messages coming across the TAF LU1 sessions that link NetView to the applications are processed by the automation table and are available for automation using the standard NetView automation capabilities.

TAF is especially useful in cases where messages from the automated application to its own console or master terminal are not available to the operating system message processing facilities. On MVS systems, for instance, both CICS and IMS generate messages to their console or master terminal that are not broadcast on the subsystem interface and are not available to NetView for automation except through TAF.

Overview

TAF offers two modes of operation. One mode of operation, ***operator-control***, or OPCTL, sessions are LU1 sessions and emulate SNA 3767 terminals. Operator-control sessions transmit messages and commands in line-by-line mode rather than full-screen mode. Messages usually viewed by the operator on the application subsystem console or master terminal are sent to NetView across the LU1 session and are processed by the automation table. Automation involving TAF is done with operator-control (LU1) sessions. Using those sessions, autotasks can enter any transaction that is possible from a 3767 terminal directly attached to the application, including CICS and IMS control functions that normally are entered from the CICS and IMS master terminals.

Another mode of operation, ***full-screen***, or FLSCN, sessions are LU2 sessions and emulate SNA 3270 terminals. By establishing a full-screen session with an application, NetView operators can view the application screens from the NetView terminal just as if they are logged directly onto the application itself from a locally attached subsystem console or master terminal. Because the data that appears on the NetView screen is transmitted in full-screen rather than line-by-line format, the data is not available to NetView automation processing facilities. Messages received over the LU2 session and viewed on the NetView screen do not pass through the automation table. Autotasks by definition do not have physical terminals; therefore, they cannot view the full-screen session.

For these reasons, you cannot use full-screen sessions for automation. You might want to use them, however, in instances where your automation does not yet handle all control functions for an application. In those instances, a NetView operator can perform the actions required by using TAF full-screen sessions. This ability plays an important part in consolidating subsystem consoles, enabling you to operate several subsystems and applications from a single NetView console. It can be important in focal-point operation, where operators at a central system controls applications on several systems, which might be remote.

Both LU1 and LU2 sessions are types of VTAM sessions. All TAF sessions, whether operator-control or full-screen, require that the VTAM program be active.

How TAF Works

TAF works by establishing a session between a TAF virtual terminal, which is called a ***source LU*** (SRCLU), and the application. The source LU is the secondary logical unit, and the application is the primary logical unit. The same source LU can establish sessions with more than one application at a time. Each operator station task (OST), whether an operator or an autotask, can use one source LU. The operator can control several applications by starting a TAF session between this source LU and each application. All messages or displays returned from an application through a TAF session are received by the operator task associated with the source LU that initiated the session.

Table 17 lists some of the applications that you can control from the NetView program using TAF. TAF can also be used with other 3270 applications. You can also use TAF to log on to and to enter commands to another NetView system, in full-screen mode only.

Table 17. Terminal Access Facility Options

Subsystem	Operator-Control	Full-Screen
CICS	●	●
IMS	●	●
HCF DPPX	●	●
HCF DPCX		●
TSO		●
Remote NCCF		●
TCAM VER		●
DSX		●
NPM		●
SSP (THRU TSO)		●

Setting Up TAF

The system setup for TAF consists of adding definition statements to the VTAMLST data set for the TAF source LUs and adding terminal definitions to your applications, such as CICS and IMS, for those same source LUs. Using the sample definitions provided for the VTAMLST data set, CICS, and IMS in the following sections can help in setting up the system.

Adding VTAMLST Definitions

To set up TAF for automation, you need to add definitions to your VTAMLST data set for your TAF SRCLUs (virtual terminals or source LUs). The NetView sample network includes sample definitions for both operator-control sessions and full-screen sessions in the VTAMLST data set member A01APPLS (CNMS0013). In the samples, source LUs with names such as TAF01O00, TAF01O01, and TAF01O02 represent operator-control sessions. Full-screen sessions are represented by source LUs named TAF01F00, TAF01F01, TAF01F02, and so forth.

Figure 166 shows a sample VTAMLST definition for a TAF source LU from the NetView samples.

```
TAF01O00    APPL    MODETAB=AMODETAB,EAS=9                X
                    DLOGMOD=M3767
*                   STATOPT='TAFAPPL 000'
```

Figure 166. A Sample VTAMLST Definition for a TAF Source LU

The parameters in Figure 166 are:

Parameter	Meaning
MODETAB	Represents the table in the VTAMLST data set that defines a logmode for each terminal type
DLOGMOD	Specifies which logmode from the logmode table is to be used for the source LU when it tries to initiate a session
AMODETAB	Is the sample logmode table provided with the NetView sample network in VTAMLST data set member AMODETAB (CNMS0001)

The logmode specifies the bind parameters for the terminal from which the source LU session is started. It includes information such as screen size and color capabilities. For TAF operator-control sessions, the logmode must always be M3767, as shown in Figure 166. Because information over operator-control sessions is always displayed line-by-line, no physical terminal need be involved at all, as in the case of autotasks. If a physical terminal is involved, its physical characteristics do not matter.

For TAF full-screen sessions, use the correct logmode to establish the TAF session. The operator must use the same logmode when starting the full-screen session that the operator used when logging on to NetView, because the physical characteristics of the terminal determine the formatting of the full-screen display.

NetView does not keep track of the logmode used when the NetView operator logs on. Therefore, operators must know the logmode that is associated with each physical terminal type and specify that logmode when starting a full-screen TAF session. Ensure that you have sufficient source LUs defined in the VTAMLST data set, with each of the various logmodes that might be required, for all NetView operators who might need to start full-screen TAF sessions.

Because you have already defined both NetView and the target applications in your VTAMLST, no other definitions are required in either the VTAMLST data set or NetView to initiate TAF sessions. However, definitions also need to be added to the applications themselves to define the source LUs to the applications as terminals.

Adding CICS Terminal Definitions

Figure 167 shows a sample CICS definition that defines a TAF operator-control session to CICS:

```
DFHTCT TYPE=TERMINAL,TRMTYPE=3767,NETNAME=TAF01O00,TRMIDNT=TOFO,
              GMMSG=YES,RELREQ=(YES,YES),TIOAL=256,VF=YES,
              LOGMODE=M3767,BUFFER=256,TRMSTAT=TRANSCEIVE,RUSIZE=256,
              BMSFEAT=(NOROUTE,NOROUTEALL),PGESTAT=PAGE,
              PGESIZE=(12,80)
```

Figure 167. Defining TAF to CICS

The keywords in Figure 167 on page 431 are:

TRMTYPE The terminal type must always be 3767 for operator-control sessions, because TAF is then emulating 3767 terminals. For full-screen sessions, the terminal type must be whatever physical terminal is to be used by the NetView operator.

NETNAME Represents the TAF source LU (SRCLU). You need to set up a separate CICS definition for every TAF SRCLU that might at some point need to log on to the application.

LOGMODE must be M3767 for operator-control sessions, to match the source LU logmode. For full-screen sessions, the logmode must match the logmode in the source LU definition and is determined by the type of physical terminal that the NetView operator uses.

TRMSTAT must be TRANSCEIVE, to enable both sending and receiving over the session with TAF.

For all other parameters, consult your CICS application programmer for guidance.

To simplify the creation of these terminal definitions, you can use the AutoInstall feature of CICS to create definitions for both LU1 and LU2 type terminals. In creating definitions for LU1 type terminals, review the results to ensure that you are getting the security protection that you anticipated.

Adding IMS Terminal Definitions

Figure 168 shows an example of an IMS definition.

```
TYPE   UNITYPE=SLUTYPE1
TERMINAL NAME=TAF01O00
NAME TAF01O00
```

Figure 168. Defining TAF to IMS

Include definition statements in the IMS generation for as many SRCLUs as you expect to use in session with IMS. The statements are included with the terminal definitions that remain statically defined, such as printers or terminals with an extra logical name.

NetView Commands Used for TAF

An operator or autotask starts a TAF session with an application by entering the NetView BGNSESS command or one of the NetView command lists that simplify the BGNSESS syntax. (BOSESS starts operator-control sessions, and BFSESS starts full-screen sessions.)

Consider security when you have an autotask enter the password or password phrase to log on to an application. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for password and suppression character issues.

An autotask can establish an operator-control session and log on by issuing the BGNSESS or BOSESS command. The autotask must have the correct password or password phrase and enter it without recording it in a system or network log. For that reason, write logon procedures in the NetView command list language and suppress the command containing the password or password phrase by preceding it with the NetView suppression character. The default suppression character is a question mark (?).

Additionally, protect the command procedure with command authorization. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for information about protecting keywords on the LIST and BROWSE commands.

To enter a password or password phrase or send a command to an application over the TAF session you have established, use the SENDSESS command. You can send commands to any application with which you can establish operator-control sessions, including CICS/VS, IMS/VS, and HCF. Messages generated by the application in response to the SENDSESS command are received by the operator or autotask that issued the SENDSESS command.

To display information about active TAF sessions, use the LISTSESS command or the LSESS command list. To end a TAF session, use the ENDSESS command or the ESESS command list.

For information about the syntax of the commands used for TAF, refer to NetView online help.

Automating Applications Using TAF

For some applications, such as DB2®, all automation can be performed without the use of TAF. For other applications, such as IMS, some automation can be performed without TAF, and some requires the use of TAF.

In an IMS environment, operational messages can have three different paths:
- Some messages go to the console only.
- Some messages go to the IMS Master Terminal Operator (MTO) only.
- Some messages go to the console and the MTO.

Your automation design depends on the message flow.

On MVS systems, messages that are broadcast on the subsystem interface are available for NetView automation without the use of TAF. For example, you do not need TAF to automate messages involved in the following types of functions:
- Start system
- Shut down system
- Recovery or restart
- For extended recovery facility (XRF), maintaining USERVAR information between systems

For messages that are not broadcast on the subsystem interface, you need to use TAF for automation. For example, on IMS, error messages are sent to the IMS master console operator.

By establishing TAF sessions to the application and logging on to the application in emulation of the master terminal or application console, you forward those messages over the TAF session to NetView, making them available for automation.

When you have established an operator-control session to an application, messages from the application or subsystem are subject to NetView automation processing just as any other messages are. Automation facilities, such as timer commands or the automation table, can issue commands over the TAF session to the application.

Chapter 30. Automation Involving Common Base Events

This chapter introduces the concept of Common Base Events and the Common Event Infrastructure that manages Common Base Events.

Note: In this chapter, Common Base Events might also be called *events*.

Introducing Common Base Events

NetView selects messages and Management Services Units (MSUs) and uses these to generate Common Base Events. These Common Base Events can be used for functions such as reporting status change, configuration changes, performance reporting, and the creation or deletion of an object. The format of a Common Base Event is expressed as an XML document, the design of which is controlled by a defined schema. Each event contains the identification of the component that reported the event, the identification of the component that is affected by the event, and the situation that caused the event to be generated.

The Common Event Infrastructure is an IBM component technology used to manage these events. It provides a server that stores Common Base Events in a database and distributes copies of the events to interested listeners.

NetView creates events and passes them to the Common Event Infrastructure for distribution. NetView also receives events from the Common Event Infrastructure for automation.

The Common Base Event format is described in *IBM Tivoli NetView for z/OS Customization Guide*. It describes the XML elements of the Common Base Event.

Creating Common Base Events

There are two means of creating Common Base Events:

- Automating a message or a Management Services Unit (MSU) to create a Common Base Event. This is described in "Creating Common Base Events by Automating Messages and MSUs."
- Setting hardware monitor filters to create Common Base Events when MSUs are recorded by the Hardware Monitor. This is described in "Creating Common Base Events by Setting Hardware Monitor Filters" on page 436.

Creating Common Base Events by Automating Messages and MSUs

You can use the automation table CBE action to construct an event and sent it to the Common Event Infrastructure server for distribution and recording in the database. The CBE action is very similar to the EDIT action. It accepts an EDIT pipe stage specification that can be used to produce the event. The key portion of the edit specification is the CBETEMP global order, described in *IBM Tivoli NetView for z/OS Programming: Pipes*. This pulls in an XML template stored by NetView when the CBE.TEMPLATES statement is processed in the CNMSTYLE member. Other stages in the edit specification can modify the template to produce the specified event.

This is a sample automation table entry that uses the default message template *msgdefault*:

```
IF MSGID=.something. THEN
    CBE('CBETEMP /msgdefault/ COPY *');
```

In this example, the global order `CBETEMP / msgdefault/` reads the `msgdefault` template. It produces as output a complete XML document that can be sent to the Common Event Infrastructure server. This order makes this template the primary input for the EDIT stage, and the subsequent COPY order copies it to the output. The resulting XML document is then sent to the server.

The CNMSCBET templates sample that is supplied with the NetView program provides a set of templates for messages and alerts. Templates can refer to variable data that pulls in information from the message or alert at runtime when the CBETEMP order is processed.

You can create your own templates, and construct more complex EDIT stages to manipulate the resulting XML. The NetView sample CNMSCBEA shows some of the ways the edit specification can be used to add or delete elements, or change the values supplied by the template.

A process similar to this can be performed for a Management Services Unit by using the `msudefault` template.

Note: If VSAM correlation data is included (by including the `vsamcorr` extended data element), the event cannot be completed until the Hardware Monitor records the MSU. Consequently, the event is not produced immediately, but is produced during Hardware Monitor recording. Only one event is produced in these circumstances, regardless of filter settings. If VSAM correlation data is not included, a Common Base Event is constructed immediately; a second Common Base Event can be produced by NPDA by use of the filter settings.

Creating Common Base Events by Setting Hardware Monitor Filters

The hardware monitor filter CBEROUTE (similar to TECROUTE and TRAPROUT) allows you to select MSUs for Common Base Event conversion without the need to code automation table statements. These Common Base Events are built using the `msugeneric` template or the `msunongeneric` template. As with TECROUTE and TRAPROUT, these filters can be specified in the automation table, as well as by using the SRFILTER command.

Using Common Base Events in Automation

NetView can also receive Common Base Events for automation purposes. When NetView receives an event, the event is presented as a multiline message XML document. The XML document is preceded by a BNH875I message indicating that an event has been received. The ACQUIRE automation table conditional can be used to select information from the document for comparison purposes. For example, suppose that a Common Base Event has an extended data element named `failure_causes`. It is presented to the automation table in this format:

```
<extendedDataElements name="failure_causes" type="string">
<values>HUB_FAILURE</values>
</extendedDataElemeents>
```

The ACQUIRE conditional to check the value can be coded in many ways. This is an example:

```
IF ACQUIRE(FINDLINE /name="failure_causes"/FINDLINE/<values>/UPTO"</values>
                  "parse/<>word 3.*) = HUB_FAILURE" THEN
```

The BNH875I message contains two inserts, `type` and `msg`. The `type` insert is set only if the event was produced by another NetView. It contains either MSG or MSU as its value in such a case, indicating whether the event was generated by a message or by an MSU. If the event was not produced by NetView, this insert is set to `N/A`. The `msg` insert contains the value of the `msg` attribute on the CommonBaseEvent tag. This is generally a high-level text description of the event. For messages, it is the first (or only) line of message text. For alerts, it is the alert description text. If this attribute is not provided as part of the event, this insert is also set to `N/A`.

Correlating Common Base Events

When events are sent to the server, they can optionally pass through the correlation process described in "Correlating Messages and MSUs Using the Correlation Engine" on page 334. By default, they do not pass through the correlation process. To have events correlated, include the correlate extended data element with a value of `true`. (See the CNMSCBET sample for details.) When an event passes through the correlation process, the XML document is converted into the event format supported by the correlation processing. Field and attribute names are used as attributes in the event. Values are generally presented to correlation as string values, but in some cases other values such as `Long` or `stringArray` are used. The mapping of event data to correlation event attributes is shown in the table below. The value for the type attribute of the correlation event is always CBE.

Note that only values contained in the Common Base Event map to a correlation event attribute. Since most of the Common Base Event fields and attributes are optional, many of these fields are not present as event attributes. You must be familiar with the format of the event that is passing through correlation to understand what attributes can be used in the correlation rules. In general, the correlation attribute taken from an event corresponds to the uppercase attribute name defined in the Common Base Event schema. If an attribute name is unique within the event schema, it is used alone. For attribute names that are not unique, such as location in a ComponentType element (which can be used for reporterComponent or sourceComponent) the attribute is preceded by the containing element's name, such as REPORTERCOMPONENT.LOCATION. For those elements that are defined as arrays, such as extended data elements or context data elements, the value on the name= attribute is used to reference the array element, followed by a '.' followed by the uppercase attribute name defined within the extended data element or context data element. Note that the values taken from `name=` attributes are not uppercase. AssociatedEvents do not have a name attribute, so a numeric value (starting at zero for the first element of the array) is assigned to each member of the array, for example, ASSOCIATEDEVENTS0.RESOLVEDEVENTS.

When an event causes a correlation rule to trigger, the most common action to take is to send the event to the Common Event Infrastructure or NetView, depending on whether the event is being received from NetView or the NetView WebSphere® client. NetView provides two actions, CBEToCEI and CBEToNetView that can be used to route the events when correlation occurs.

Table 18. Common Base Event Correlation

Common Base Event attribute	Correlation attribute	Correlation value type
Common Base Event attribute	Correlation attribute	String
CommonBaseEvent.extensionName	EXTENSIONNAME	String
CommonBaseEvent.Version	VERSION	String
CommonBaseEvent.globalInstanceId	GLOBALINSTANCEID	String
CommonBaseEvent.localInstanceId	LOCALINSTANCEID	String
CommonBaseEvent.priority	PRIORITY	String
CommonBaseEvent.creationTime	CREATIONTIME	String
CommonBaseEvent.severity	SEVERITY	Long
CommonBaseEvent.repeatCount	REPEATCOUNT	Long
CommonBaseEvent.elapsedTime	ELAPSEDTIME	Long
CommonBaseEvent.sequenceNumber	SEQUENCENUMBER	Long
reporterComponentId.application	REPORTERCOMPONENTID.APPLICATION	String
reporterComponentId.component	REPORTERCOMPONENTID.COMPONENT	String
reporterComponentId.componentIdType	REPORTERCOMPONENTID.COMPONENTIDTYPE	String
reporterComponentId.componentType	REPORTERCOMPONENTID.COMPONENTTYPE	String
reporterComponentId.executionEnvironment	REPORTERCOMPONENTID.EXECUTIONENVIRONMENT	String
reporterComponentId.instanceId	REPORTERCOMPONENTID.INSTANCEID	String
reporterComponentId.location	REPORTERCOMPONENTID.LOCATION	String
reporterComponentId.locationType	REPORTERCOMPONENTID.LOCATIONTYPE	String
reporterComponentId.processId	REPORTERCOMPONENTID.PROCESSID	String
reporterComponentId.subComponent	REPORTERCOMPONENTID.SUBCOMPONENT	String
reporterComponentId.threadId	REPORTERCOMPONENTID.THREADID	String
sourceComponentId.application	SOURCECOMPONENTID.APPLICATION	String
sourceComponentId.component	SOURCECOMPONENTID.COMPONENT	String
sourceComponentId.componentIdType	SOURCECOMPONENTID.COMPONENTIDTYPE	String
sourceComponentId.componentType	SOURCECOMPONENTID.COMPONENTTYPE	String
sourceComponentId.executionEnvironment	SOURCECOMPONENTID.EXECUTIONENVIRONMENT	String
sourceComponentId.instanceId	SOURCECOMPONENTID.INSTANCEID	String
sourceComponentId.location	SOURCECOMPONENTID.LOCATION	String
sourceComponentId.locationType	SOURCECOMPONENTID.LOCATIONTYPE	String
sourceComponentId.processId	SOURCECOMPONENTID.PROCESSID	String
sourceComponentId.subComponent	SOURCECOMPONENTID.SUBCOMPONENT	String
sourceComponentId.threadId	SOURCECOMPONENTID.THREADID	String
AssociatedEvents[n].resolvedEvents	ASSOCIATEDEVENTS*n*.RESOLVEDEVENTS	String
AssociatedEvents[n].associationEngineInfo.name	ASSOCIATEDEVENTS*n*.ASSOCIATIONENGINEINFO.NAME	String
AssociatedEvents[n].associationEngineInfo.type	ASSOCIATEDEVENTSn.ASSOCIATIONENGINEINFO.TYPE	String
AssociatedEvents[n].associationEngineInfo.id	ASSOCIATEDEVENTSn.ASSOCIATIONENGINEINFO.ID	String
AssociatedEvents[n].associationEngine	ASSOCIATEDEVENTSn.ASSOCIATIONENGINE	String
contextname.type	contextname.TYPE	String
contextname.contextValue	contextname.CONTEXTVALUE	String
contextname.contextId	contextname.CONTEXTID	String
Any[*n*]	ANY*n*	String
MsgDataElement.MsgId	MSGID	String
MsgDataElement.MsgCatalog	MSGCATALOG	String
MsgDataElement.MsgLocale	MSGLOCALE	String
MsgDataElement.MsgCatalogType	MSGCATALOGTYPE	String
MsgDataElement.MsgCatalogId	MSGCATALOGID	String
MsgDataElement.MsgTokens	MSGTOKENS	StringArray
situation.categoryName	CATEGORYNAME	String
situation.reasoningScope	REASONINGSCOPE	String
situation.reportCategory	REPORTCATEGORY	String
situation.operationDisposition	OPERATIONDISPOSITION	String
situation.availabilityDisposition	AVAILABILITYDISPOSITION	String
situation.processingDisposition	PROCESSINGDISPOSITION	String
situation.successDisposition	SUCCESSDISPOSITION	String
situation.situationDisposition	SITUATIONDISPOSITION	String
situation.dependencyDisposition	DEPENDENCYDISPOSITION	String
situation.featureDisposition	FEATUREDISPOSITION	String

Table 18. Common Base Event Correlation (continued)

Common Base Event attribute	Correlation attribute	Correlation value type
situation.situationQualifier	SITUATIONQUALIFIER	String
situation.otherAny	OTHERANY	String
extendedDataElements.type	namehierarchy.TYPE	String
extendedDataElements.values	namehierarchy.VALUES	String or StringArray

Notes on the table:

- *n* is a numeric value used to reference elements in an array of AssociatedEvents[].
- *contextname* is the value of the `name=` attribute value of a contextDataElement member of the ContextDataElement[] array.
- *namehierarchy* is the hierarchy of `name=` values used to reference an extended data element and its children. For example, suppose an event had this extended data element:

```
<extendedDataElements
   name=sample
   type=string
  <values>top level value</values>
  <children name-secondLevel type=string>
    <values>secondLevelValues</values>
    <children name=thirdLevel typ=string>
        <values>thirdLevelValues</values>
    </children>
  </children>
</extendedDataElements>
```

The correlation attributes that are built for this extended data element are:

```
sample.TYPE=string
sample.VALUES=top level value
sample.secondLevel.TYPE=string
sample.secondLevel.VALUES=secondLevelValues
sample.secondLevel.thirdLevel.TYPE=string
sample.secondLevel.thirdLevel.VALUES=thirdLevelValues
```

Chapter 31. Using Automated Operations Network

Automated Operations Network (AON) provides comprehensive, drop-in, policy-based programs that can be customized and extended to provide network automation. The following components of AON provide consistent automation across network protocols:

- SNA Automation (AON/SNA)
- TCP/IP Automation (AON/TCP)

You can choose to run one or more of these components.

Note: Many IP-related functions that were previously implemented as part of the NetView Automated Operations Network (AON) component are now implemented as base NetView services. These functions, which are referred to as the AON IP functions, no longer require AON enablement and configuration. Instead, they can be enabled by using the IPMGT tower. However, AON IP functions that are already enabled using the AON TCP subtower do not need to be reconfigured. The AON TCP subtower and the IPMGT towers are mutually exclusive. Additional information on the IPMGT tower can be found in *IBM Tivoli NetView for z/OS Installation: Configuring Additional Components*.

AON components intercept alerts and messages that indicate problems with network resources. The components of AON monitor and attempt to recover failed resources. The AON components also record resource failures to enable you to track recurring network problems. AON uses many of the functions described in this manual such as global variables, automation table constructs, and timers. Implementing AON automation provides you with a base set of automation functions.

Understanding AON Automation and Recovery

The automation provided by AON is driven by messages, MSUs, or timers. Messages and MSUs are trapped by the automation table, which calls AON routines to take predetermined actions. These actions are governed by the automation policy specified in the control file. To understand AON, you must be familiar with the automation table and the control file. The components of AON provide the automation policy and automation table statements to trap messages and MSUs and take actions appropriate for the network.

The AON full-screen operator interface enables you to issue commands and receive responses for AON functions and other NetView facilities.

Although there are different types of networks and resources, the tasks that automate each of these networks are similar. AON provides automation modules for the tasks that are similar. For example, when a resource fails in an SNA or TCP/IP network, information about the resource, such as the resource type, connectivity, and status, must be gathered before automated recovery can continue for the resource. Each of these network types has a different program to gather the resource information, but they are called and processed the same way by the automation driver.

The benefits of having common automation drivers are increased reliability, reduced training, and simplified network problem determination.

Automation Table

The automation table detects messages, alerts, and resolutions and takes actions based on those messages and MSUs. It then drives AON failure or recovery processing, as appropriate.

The Control File

The control file contains the automation policy that determines how automation works in your particular network. All of the components of AON store their automation parameters in the control file. For coding flexibility, the control file data is accessible by key or data-content searches.

The control file is loaded into storage each time NetView is initialized. You can change the control file data while AON is running by issuing the appropriate commands from the command line or from the operator interface. These changes to the control file data are made in storage rather than in the control file. If you want these changes to be permanent, your systems programmer can change them in the control file using an editor.

The automation policy is contained in the control file to provide these advantages:

- Site-dependent variables (for example, the name of a resource to be recovered and monitored) remain separate from the routines that use the information.
- The information uses positional and KEYWORD=*value* parameters that are easy to code and independent of the language used for the routines.
- The design is flexible. A control file entry can be for specific resource name or for a resource type such as PU and NCP. If resource names or types are not specified, enterprise-wide defaults are used.

Note: Refer to the *IBM Tivoli NetView for z/OS Administration Reference* for more information about the control file entries.

Understanding Automated Operators

The NetView automation task design enables AON to divide its workload among separately defined automated operators providing concurrent processing. ***Automated operators*** are programs that have an operator ID and are known to NetView to be NetView operators. Automated operators do much of the automation work such as reactivating or deactivating resources, sending messages and MSUs to the operators when further action is required, and sending records to an automation log for tracking purposes. The operator status panels, called the Dynamic Display Facility (DDF), reflect these changes by updating the panel in real time. Each component adds additional automation tasks.

Understanding Notifications

An automation notification can consist of one or more of these responses:

- Message
- MSU
- Event Integration Facility (EIF) event
- Dynamic Display Facility (DDF) update

- NetView management console update, such as the `Automation In Progress` status
- Beeper or email request using Inform Policy definitions

These notifications describe significant actions detected or taken by AON. A notification informs the operator that a resource requires operator intervention or that a significant network event has occurred. Messages are routed to the appropriate operators (known as the *notification operators*) and optionally held on the command facility display until deleted.

AON message notifications occur based on the usage of message classes. You can define notification operators with sole responsibility for specific message classes. For example, define one or more notification operators to receive messages pertaining only to TCP/IP resources.

You can also define inform policy statements that enable you to notify appropriate personnel using pagers (numeric or alphanumeric) or email. The inform policy is customizable. For example, you can send e-mail to first shift operators and page third shift operators.

Understanding Automation Tracking

To improve problem determination productivity, AON provides these files for recording the automation process and the status of the network resources:

Status file
The status file time stamps and tracks the last 10 failures experienced by a resource. The status file also tracks current automation statuses and threshold exceptions. If operators were notified of the last failure, AON tracks which operator last acted on the resource.

Automation log
A record of automation activity is kept in an optional automation log that consolidates automation information in one place. AON writes availability records to the log when a resource has become unavailable and when the resource becomes available again. These records indicate whether the action was caused by automation, the help desk function, or by an operator. The information kept in the automation log is also recorded in the NetView log to be used for automation and reports.

Understanding Automation Notification Logging in the Hardware Monitor

AON logs its resource-related notifications in the hardware monitor database for use in problem determination and event correlation on specific resources. You can use the Alerts Dynamic and Most Recent Events facility, which enables a NetView graphical display to reflect AON automation activity.

Resource Recovery and Thresholds

You can set recovery criteria for an individual resource, for a group of resources, or for global levels. You can select many parameters and options, such as setting thresholds for the number of errors counted during a period of time or automating recovery based on the time of day or day of the week.

When a resource changes from available to unavailable, a message or alert is generated. The components of AON intercept this message or alert and begin

recovery actions for the resource. AON notes these outages in a status file. If the number of errors exceeds a user-defined number in a period of time, the AON components attempt to recover the resource and notify the appropriate operator that the resource is experiencing frequent or infrequent errors. This gives you the opportunity to proactively investigate resources with degraded availability.

When a resource demonstrates availability problems so often that the user-specified critical threshold setting is exceeded, automation is stopped to prevent reiterative, unproductive recovery attempts. The AON components continue to check the status of the resource and try to recover it at intervals defined in the control file.

The components of AON do much more than issue commands and recover a resource. Figure 169 on page 445 illustrates a basic example of AON recovery logic.

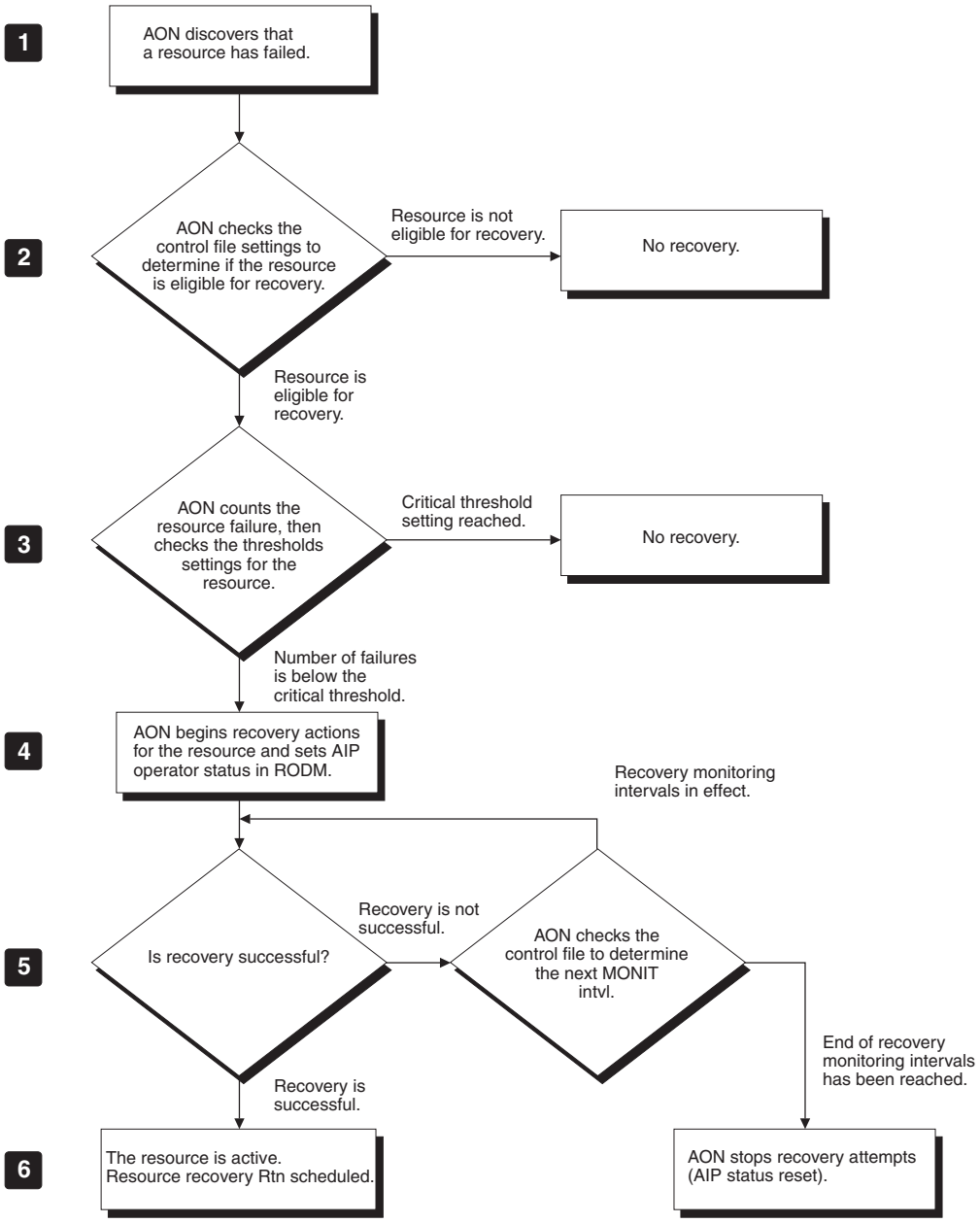


Figure 169. Automation Failure Logic

When a network failure occurs:

1 When the status of a resource changes from available to unavailable, a message is generated. AON intercepts this message and begins recovery actions for the resource.

2 AON determines whether recovery processing is active for the resource. You can set automation completely off for a resource or schedule times when automation does not occur.

3 AON uses thresholds to determine whether to notify operators of the resource failure and whether to continue recovery attempts. You can set three threshold levels: infrequent, frequent, and critical. At the infrequent or frequent threshold levels, AON notifies operators to give them the opportunity to proactively investigate resource failures. When the number

of failures exceeds the critical threshold, AON notifies operators and stops further recovery attempts to prevent reiterative, unproductive automation.

4 AON begins reactivation attempts. For all types of failed resources, AON monitors resource recovery at the intervals specified in the control file. For RODM users, the Automation in Progress (AIP) operator status is also set for resources that AON is recovering.

5 AON determines whether the recovery attempt is successful. If recovery is not successful, AON checks the control file to determine whether to continue recovery actions or stop recovery processing.

6 Recovery was successful and the resource is active. If proactive monitoring is defined, it begins at this point.

For each failure of a network resource (such as a line, PU, or CDRM for SNA resources), AON automation checks the control file for automation scheduling, threshold analysis, reactivation scheduling, and notification directions.

AON/SNA Automation

AON/SNA automates System Network Architecture (SNA) and offers automation functions for:

- VTAM/SNA subarea resource monitoring
- VTAM/SNA Advanced Peer-to-Peer Networking resources

Figure 170 shows the types of network resources that AON/SNA controls.

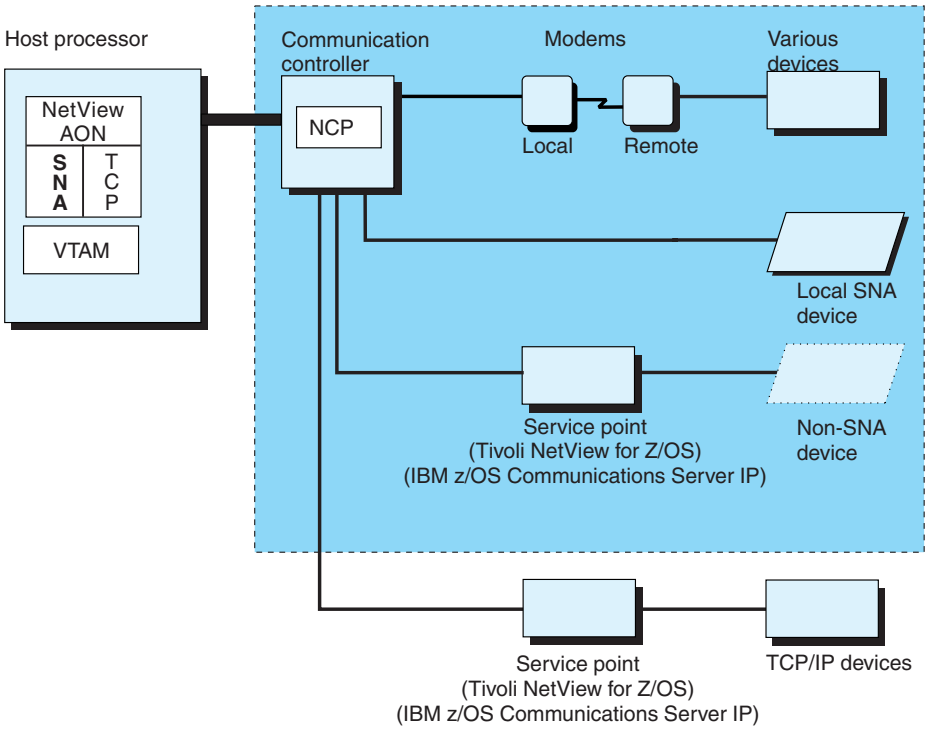


Figure 170. Resources Automated by AON/SNA

AON/SNA automates network recovery based on VTAM messages and alerts that indicate problems with network resources. AON/SNA intercepts critical VTAM messages and alerts that indicate problems with network resources. AON/SNA then issues commands to reactivate the failed resources and monitors the resources

until they are active again. By providing both management and control for your network, AON/SNA produces quantifiable savings.

Some of the capabilities of AON/SNA are:
- Real-time status display
- Operator-productivity functions from a simple operator interface
- Automated help desk
- Resource processing for status changes

To perform a task in AON/SNA, you can use the *operator interface* or panels. To bypass the operator interface, use the command line in NetView, AON, or any AON component that is installed and initialized on your system. You can also manage control of AON/SNA recovery from the workstation interface.

Understanding the AON/SNA Options

You can use AON/SNA to automate control of your network. As an operator, you can use the AON/SNA operator interface as follows:
- Display the online tutorial
- Resolve network problems
- List the resources on a domain
- Look at and update the VTAM start-up options
- See the status of resources
- Issue VTAM commands
- Monitor Advanced Peer-to-Peer Networking resources
- Manage leased lines
- Monitor X.25 switched virtual circuits
- Display NCP recovery definitions

These topics describe each of these options.

Using the AON/SNA Tutorials

An online tutorial provides an overview of each of the options on the SNA Automation: menu panel and specific instructions about using these options.

Using the AON/SNA Help Desk

AON/SNA provides an SNA-specific automated help desk component that enables inexperienced help desk operators to resolve network problems. To use the SNA help desk, you need only the terminal ID of the user. After you enter the terminal ID on the appropriate panel, the SNA help desk displays a picture of how the user's terminal is attached to the system.

Note: If NetView Access Services is installed on your system, the SNA help desk can determine the location of network problems by using the user ID. In this case, you do *not* need to know the terminal ID.

The SNA help desk automates the problem determination procedures. To solve network problems, you can select an action from a list of recovery procedures on the SNA help desk problem determination panels. To increase the productivity of help desk personnel, the SNA help desk:
- Reduces the amount of input you enter.
- Automates problem determination.
- Enables you to be productive immediately, even though you do not know the network configuration.
- Teaches problem determination skills while resolving network failures.

The SNA help desk enables you to view a resource and its connected higher nodes. With SNAMAP, you can display a resource and zoom to its connected lower nodes.

Using SNAMAP

AON/SNA provides a tool called SNAMAP to list the resources on a particular domain. To create a list of the resources, select a resource type or enter the name of a specific resource. When you select a category, AON/SNA displays a list panel that shows the resource names that fit that category.

SNAMAP can zoom to its connected lower nodes. In contrast, the SNA help desk provides a view of a resource and its connected higher nodes.

Managing VTAM Options

The VTAM options management function displays the current and default VTAM start-up options for a domain. You can scroll through this list of options and make changes by moving the cursor to the setting you want and typing over the information displayed. After you press Enter, AON/SNA processes the changes.

Using NetStat

NetStat displays a resource list that is created by selecting the type of resource, the status you want to display, and whether you want the recovery flag to be set. Using this information, NetStat displays a list of resources that fit the criteria you selected. This option is a display only option.

Issuing VTAM Commands

The VTAM commands option enables you to issue VTAM commands from a panel and display the results of the command on a subsequent panel. If the results of the command you issued are displayed on more than one panel, you can scroll through the panels to see all of the information. The VTAM commands option saves commands across user task sessions. If you type a command on the VTAM commands panel, the command is displayed in the same place on that panel when you return to that panel (in the same session or at a later session).

The VTAM commands option is cursor sensitive. AON/SNA issues the command at the position of the cursor. To issue a command, you can type a new command on the panel and press Enter or move the cursor to the command you previously entered and press Enter.

Monitoring X.25 Switched Virtual Circuits

AON/SNA includes monitoring SNA physical resources that make up X.25 switched virtual circuits (SVC). The operator interface enables you to add, change, and delete SVC links that AON/SNA is monitoring.

The X.25 commands also include LUDR pool management. This provides a mechanism to determine the logical unit count available for an NCP. AON/SNA can monitor this count and alert operators when the count drops below a defined threshold.

Displaying NCP Recovery Definitions

You can display the NCPRECOV control file definitions for a particular NCP or for all defined NCPs.

AON/SNA Subarea VTAM Resource Automation Support

AON/SNA supports SNA subarea networks by trapping VTAM messages indicating changes in the network. This includes resource status changes, storage problems, and outstanding replies requiring operator action. Resources managed

include NCPs, lines, physical units, cross-domain resource manager (CDRMs), cross-domain resources (CDRSCs), applications (ACBs), and sessions.

You can monitor AON/SNA subarea resources by:
- Passive monitoring
- Recovery monitoring
- Proactive monitoring

Passive monitoring occurs if you want AON/SNA to notify you when there is a problem with an AON/SNA resource, which is indicated by a VTAM message. Recovery monitoring consists of checking and trying to recover an unavailable resource at the intervals you defined in the MONIT control file entry. Proactive monitoring occurs if you want AON/SNA to periodically monitor and report on important network devices.

Monitoring Advanced Peer-to-Peer Networking Resources

AON/SNA Advanced Peer-to-Peer Networking is a powerful, flexible networking solution for client-server and distributed applications. AON/SNA Advanced Peer-to-Peer Networking provides proactive monitoring for Advanced Peer-to-Peer Networking resources, including control points and sessions between Advanced Peer-to-Peer Networking resources. AON/SNA Advanced Peer-to-Peer Networking provides an operator interface for common VTAM Advanced Peer-to-Peer Networking functions.

AON/SNA X.25 Monitoring Support

AON/SNA includes monitoring SNA physical resources that make up AON/SNA X.25 switched virtual circuits (SVC). The operator interface enables you to add, change, or delete switched virtual circuits (SVC) links that AON/SNA is monitoring.

Switched connections are monitored by using a full-screen panel. Each time there is a connection or a disconnection related to the monitored lines, the panel is updated.

The AON/SNA X.25 commands also include LUDRPOOL management. This provides a mechanism to determine the logical unit count available for a Network Control Program (NCP). AON/SNA monitors this count and alerts operators when the alert drops below a defined threshold.

Users of the X.25 NCP Packet Support Interface (NPSI) can obtain network event information from the hardware monitor. AON/SNA provides a communication network management (CNM) interface user exit. This user exit suppresses the alerts coming from NPSI and generates a BNJ146I message containing the original data given by NPSI. This message prompts AON/SNA to trigger programs that scan the error bytes and other information in the alert, find the meaning of the error bytes, and send an improved alert by the GENALERT command. This alert in the hardware monitor gives the operator a clear interpretation of the error, including specific actions. AON/SNA inserts a message in the NetView log. This message correlates the INOP message generated by VTAM and the corresponding NPSI alert. Also, each time the hardware monitor encounters an incorrect XID problem, an alert is sent to the monitor.

AON/TCP Automation

AON/TCP helps you manage your TCP/IP network. The connection between NetView and your TCP/IP network is the NetView for UNIX service point or TCP/IP for z/OS.

Some AON/TCP functions are:

- AON/TCP detects, reacts to, and notifies NetView operators of TCP/IP resource failures. You can instruct AON/TCP to use passive monitoring, proactive monitoring, or both to detect these network failures.
- AON/TCP uses ping responses during resource failure processing to detect name server failures in the TCP/IP network. If a name server is down, AON/TCP notifies operators so they can reconnect the appropriate name server to the TCP/IP network.
- With AON/TCP you can associate interfaces with routers. When a router interface fails, AON/TCP notifies operators that both the router and the router interface (link) are not fully operational.
- With the Dynamic Display Facility (DDF) and the operator interface, operators can manage the TCP/IP network on an exception basis by seeing and managing only problem resources, without sorting through resources that are active and operating properly.
- In large TCP/IP networks, you can limit the types of failures and resources that AON/TCP monitors. To use definitions in the AON control file, you can change these definitions as your network requirements change by using the AON/TCP operator interface.

When you are using the Tivoli NetView (AIX) product with AON/TCP:

- AON/TCP uses the NetView RUNCMD command to communicate with the NetView (AIX) management software. NetView (AIX) sends alerts to NetView to notify it of changes in the TCP/IP network. AON/TCP uses the alerts and RUNCMD commands to provide network management assistance at the NetView host.
- Besides failure detection, AON/TCP also watches for performance problems such as limited free disk space and excessive CPU utilization on TCP/IP hosts that report those kinds of problems to Tivoli NetView (AIX).
- AON/TCP counts security authorization failures and compares the number of failures to threshold values you specified. Operators are notified for every security authorization failure or when there are excessive failures on a node.

When you are using TCP/IP for z/OS with AON/TCP, you can:

- Manage IP connections across multiple TCP/IP stacks.
- Perform problem determination on hung sessions (such as TN3270 and FTP) and take corrective actions. For example, you can drop the session.
- Issue SNMP requests such as GET and SET to manage your IP resources.
- Manage TCP/IP resources using IPMAN.
- Perform MIB polling and thresholding on selected IP resources.
- Correlate IP interface and host traps
- Monitor and recover failed IP sockets
- Support (concurrently) both UNIX and TSO environments
- Provide support for starting and stopping these IP traces:
  - CTRACE

  - PKTTRACE
  - OSATRACE
- Use SNMP to manage
  - Your IP stack
  - Remote devices

Proactive monitoring is automatically started for each TCP/IP for z/OS stack defined in the control file. Figure 171 shows the part of the network that AON/TCP monitors.

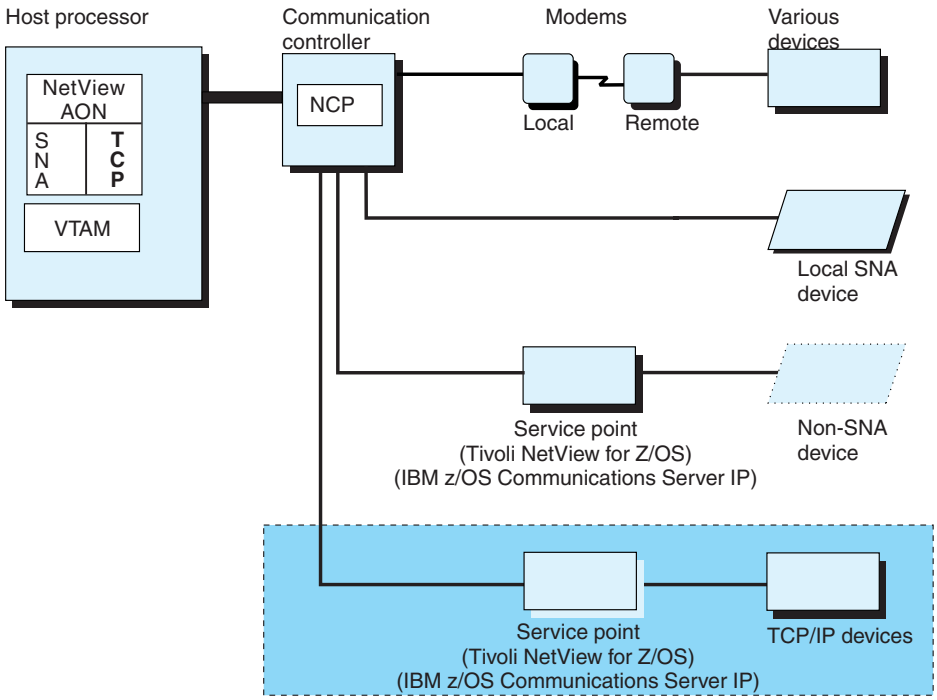


Figure 171. Tivoli NetView (AIX) monitors resources

The following sections describe the capabilities of AON/TCP in more detail.

Passive Monitoring in AON/TCP for Tivoli NetView (AIX)

Passive monitoring occurs if you want AON/TCP to only react to notifications sent to it by Tivoli NetView (AIX) about resource failures. Tivoli NetView (AIX) sends alerts to notify NetView of TCP/IP network changes. NetView stores these alerts in the hardware monitor database. AON/TCP traps these alerts in the NetView automation table and drives failure processing, recovery processing, or performance threshold analysis for the alerts it receives.

In large TCP/IP networks, you might not want AON/TCP to notify you about the connection or disconnection of every TCP/IP device. Adjusting alert processing in large networks is helpful because the rate of resource status change is directly proportional to the size of the networks and the number of important resources becomes a smaller percentage of the total resources.

To help you manage larger networks, AON/TCP can track and process only status changes and alerts for explicitly defined resources, groups of resources, or failure types. Tivoli NetView (AIX) can send up to three alerts for each connection or connection loss. To save CPU time, you can define which of these failure types you

want AON/TCP to process. You can also filter alerts that are sent to NetView with definitions on the Tivoli NetView (AIX) system. Filtering saves CPU time and DASD storage on the mainframe and decreases network traffic.

AON/TCP processes failures and recoveries using AON automation drivers, which update operators (including logs and DDF) and manage recovery and proactive monitoring. TCP/IP hosts can send alerts reporting CPU time and DASD storage. AON/TCP notifies operators only when defined CPU or DASD usage thresholds are exceeded. AON/TCP also traps security authorization failures. AON/TCP can notify operators for each such failure or only when there are excessive performance or security failures (for example, a threshold has been exceeded).

Proactive Monitoring

Continuously monitoring and reporting on network devices is called proactive monitoring.

To actively monitor your resources, place a definition in the AON/TCP control file. AON/TCP issues a PING command or SNMP GET request (z/OS only), which checks the status of these resources for availability at AON/TCP initialization, and at the predefined time intervals.

If a resource is not available (the status you defined in the control file is not the current status), AON tracks the outage, updates the operators, and manages recovery monitoring. Recovery monitoring is in effect during an outage so proactive monitoring is not necessary. Therefore, AON suspends proactive monitoring whenever it knows a resource is unavailable. When the resource is available again, the active monitoring timer is reinstated. AON/TCP issues messages that tell you whether failures are being detected by proactive or passive monitoring. This enables you to discontinue the overhead of proactive monitoring when passive monitoring is sufficient.

Recovery Monitoring

Recovery monitoring consists of checking the status of an unavailable resource at the intervals you defined in the AON control file.

AON/TCP starts recovery monitoring whenever it detects a failure for a resource. This monitoring continues at intervals defined in the AON/TCP control file. You must define these intervals to be irregularly spaced in order to monitor more frequently at the beginning of an outage, when recovery is most likely to occur. The longer the outage continues, the less frequently you need to monitor the resource for recovery. When a recovery is detected, AON recovery automation begins. AON updates operators, stops recovery monitoring, and reinstates the proactive monitoring that is defined in the control file.

Threshold values for AON/TCP with Tivoli NetView (AIX)

AON/TCP with Tivoli NetView (AIX) checks the threshold values you defined in the control file to determine what action it must take when a certain threshold is exceeded. AON/TCP with Tivoli NetView (AIX) checks:

- Resource failures
- CPU utilization
- Disk utilization
- Security authorization failures

- IP address to host name resolution failures (excessive failures of this type usually indicate a name server failure)

When the system exceeds any of the threshold values you defined, AON/TCP with Tivoli NetView (AIX) notifies operators about the exception. You can define AON/TCP to notify operators for each event (for example, every resource failure or security failure) or only for a threshold exception. You can also define the types of network changes for TCP/IP devices that compose a failure. For example, you can specify that AON/TCP considers host failures, interface failures, or link failures to be a failure.

MIB Polling and Thresholding (TCP/IP for z/OS only)

Using AON proactive monitoring, you can use SNMP requests instead of PING commands. SNMP proactive monitoring provides MIB polling functions. MIB polling queries the interface table of the device being monitored. If one or more interfaces have an incorrect status, AON/TCP sends a notification. You can code a user exit for further processing.

MIB thresholding can occur while a device is being proactively monitored. MIB thresholding queries MIB variables that you define and compare their expected values with the actual values. AON/TCP compares less than, less than or equal, equal, greater than or equal, and greater than conditions. When a threshold is exceeded, AON/TCP sends a notification. You can code a user exit for further processing.

As an example, you can use the MIB polling and thresholding functions for a router. MIB polling detects failed interfaces on the router. With MIB thresholding, you can define performance-related MIB variables and their thresholds. Unless AON/TCP cannot communicate with the device, the proactive monitoring continues.

For more information about setting up proactive monitoring, refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Operator Awareness

Operators can choose from several AON/TCP interfaces to receive updates about network exception conditions:

- Define operators as notification operators, so that they receive messages for each exception condition. They can clear these messages by refreshing their screen or using the AON DM command.
- Access DDF where all the exception conditions are color coded according to severity and organized by time, severity and resource type. You can quickly see only the TCP/IP resources that require attention and receive real-time updates. DDF display panels have function keys set to commands you can issue by moving the cursor to the resource and pressing the appropriate function key. The samples provide commands to ping the resource, send an AIX command, or show the NV6KVIEW summary data for the resource. From NV6KVIEW, you can manage AON/TCP recovery definitions and influence monitoring.
- Browse the NetView log or the AON automation log for messages regarding a resource in question, particularly if the resource is currently available.
- If you need data for a particular resource, use the AON AUTOVIEW or NV6KVIEW commands to display a summary of all data known by AON/TCP about the resource.

This includes control file data, status file data, DDF data, and ping results. This panel has a pop-up window with commands that an operator can use to change automation settings, send a ping, view NetView log data applicable only for this resource or change monitoring. With TCP/IP for z/OS, you can use IPMAN command to assist you with managing your network.

- Send all messages regarding a resource to the hardware monitor.

  You can view these messages from the hardware monitor or a NetView graphical display.

For more information about AON functions and customization capabilities, refer to the *IBM Tivoli NetView for z/OS User's Guide: Automated Operations Network*.

Chapter 32. Running Multiple NetView Programs Per System

You can run multiple NetView programs on a single system. When you install multiple NetView programs on a single system to separate the functions:

- You can separate system support from network support. For example, one NetView program can handle system operation and automation and another can handle network operation, automation, and problem determination.
- You can separate problem determination from automation. In this case, the automated NetView program includes system and network automation. If you do not want network management activities to affect automation performance, run automated NetView at a higher priority than any of the tasks it automates, and schedule the problem determination NetView so that it does not interfere with subsystems or network automation.

When two NetView programs are running on a single system, each NetView program can monitor and recover the other. If a single NetView program is limited by constraints for a critical resource, such as the command list data set or network log, a second NetView program can be beneficial. However, CPU utilization is not decreased in this situation.

When migrating from one NetView release to another, you can use one NetView program as a production system while using a second to test the new NetView release.

Running multiple NetView programs requires:

- Maintaining additional copies of libraries and logs
- Routing a message between NetView programs
- Ensuring a message is not automated more than once
- Using a different designator character for each NetView subsystem

Additional considerations are:

- Some NetView functions cannot be used on more than one NetView program per system or per VTAM program.
  - For a single VTAM, only one NetView program at a time can use the program operator interface (POI).
  - Only one NetView program can use the CNMI to receive unsolicited network management data from a given VTAM program, and only one can use the VTAM status monitor performance enhancement.
  - Only one NetView program in a system can use the hardware monitor local device interface.
- Because each NetView program maintains its databases separately, operators on the two NetView programs do not, for example, browse the same network log.

See “NetView Interfaces and Functions” on page 456 for more information about NetView function restrictions.

Running more than one NetView program requires additional processor storage. A small system might not have the processor storage capacity required to run two NetView programs. Also, NetView CPU utilization tends to be higher with more than one NetView program.

Installing Multiple NetView Programs

These are things to consider when installing multiple NetView programs in an automation environment:

- One NetView program might not need all the NetView libraries and data sets. However, keep all libraries and data sets in the NetView start procedure to avoid problems and to have them on hand in case recovery situations occur (that is, if one NetView program takes over the functions of another).
- Online help panels, VTAM files, hardware monitor panels, and hardware monitor color maps must be included in the NetView procedure so the NetView program can be initialized without errors.
- Multiple NetView programs cannot share any NetView databases. Each NetView program requires unique databases for the NetView functions it uses (and for any NetView functions it might take over during a recovery situation).
  - A NetView program that is to record alerts or display them on the hardware monitor requires the hardware monitor and network log data sets.
  - A NetView program that is not to record or display alerts requires only the network log data sets.
- NetView programs on a system can share all DSIPARM members. NetView load libraries and panel data sets can also be shared across two NetView programs.

NetView Interfaces and Functions

Some NetView interfaces or functions can be used by only one NetView program per operating system or by only one NetView program per VTAM program. Other NetView interfaces or functions can be used by more than one NetView program in a system but require special setup. The following sections describe considerations related to NetView functions and interfaces in the multiple-NetView environment.

Program Operator Interface (POI)

Through the POI, NetView receives unsolicited VTAM messages. You can use unsolicited VTAM messages for network automation. For a single VTAM program, only one NetView program at a time can use the POI. To use the POI, you must specify AUTH=PPO on the VTAM APPL statement for the NetView PPT (*domid*PPT).

Although only one NetView program at a time can use the POI in a single VTAM program, other NetView programs can be defined as secondary program operators (SPOs). NetView programs defined as SPOs can enter VTAM commands and receive solicited messages from the VTAM program (for example, responses to commands). For these NetView programs, specify AUTH=SPO on the VTAM APPL statement for the NetView PPT (*domid*PPT). If you want to automate unsolicited VTAM messages on an SPO NetView program, send the messages from the PPO NetView program.

If an active NetView is the primary program operator (PPO) for VTAM, VTAM sends all unsolicited messages to that NetView program only. Unsolicited VTAM messages are not available on the subsystem interface.

If the PPO is not active or AUTH=SPO is specified for all NetView programs, unsolicited VTAM messages go to the operating system console, where the messages are available on the subsystem interface to all NetView programs using

the subsystem interface. In this case, you must ensure that you do not automate an occurrence of a VTAM message more than once.

With releases of the VTAM program prior to V3R3, the NetView status monitor uses the POI to obtain status information. In this case, only the NetView program with the POI can have accurate status information.

Communications Network Management Interface (CNMI)

Only one NetView program can use the CNMI to receive unsolicited network management data from a given VTAM. This restriction is controlled by how you code the VTAM global routing table, which specifies which NetView task receives each of the following types of network management data:

- Network management vector transport (NMVT) (alert, unsolicited RTM data at session end)
- RECFMS
- Other CNM data records

You must specify that all of the types of network management data listed above are to be received by DSICRTR, the CNM router, as specified in the sample VTAM routing table ISTMGC00. DSICRTR then distributes the data to the appropriate NetView task (session monitor or hardware monitor).

In a multiple-NetView environment, NetView that uses the CNMI can be called a problem determination NetView program because the unsolicited network management data available to it is used for problem determination. Alerts flow across the CNMI, so network alert messages are generated only at NetView that is using the CNMI. If those alert messages are to be automated, they must be automated at the CNMI NetView program or routed to another NetView program for automation.

The NetView program that uses the CNMI (the problem determination NetView program) performs these actions:

- Starts the CNM router (DSICRTR) and the traditional network management tasks (session monitor, hardware monitor, and 4700 Support Facility tasks)
- Starts the VTAM APPL for DSICRTR and the VTAM APPLs for the network management tasks
- Accesses the NetView databases associated with the network management functions
- Receives MDS-MUs addressed to the VTAM CP name

A NetView program that is not using the CNMI can start DSICRTR and can use its own hardware monitor databases if necessary for system alerts (see “GENALERT” on page 459), but it must not activate the VTAM APPL associated with DSICRTR.

Note: Changing the CNMI from one NetView program to another requires bringing down the traditional network management tasks and their VTAM APPLs.

Hardware Monitor Local-Device Interface

Only one NetView program in a system can receive local-device records from the operating system. These local-device records include system-formatted alert records, such as:

- Outboard record (OBR)

- Miscellaneous data record (MDR)
- Machine check handler (MCH)
- Channel recovery word (CWR) record
- Second level interrupt handler (SLIH) records

If you automate these records, you must automate them on the NetView identified to receive the hardware monitor local-device records, or first route them to another NetView program. Messages related to the errors might also be available on the MVS subsystem interface for all NetView programs using the subsystem interface.

For NetView that is to receive the hardware monitor local-device records:
- Use a start procedure with PGM=BNJLINTX.
- Activate BNJMNPDA.

For a NetView program that does not receive hardware monitor local-device records:
- Use a start procedure with PGM=DSIMNT.
- Do not activate BNJMNPDA.

Note: Changing the receiver of the hardware monitor local-device records from one NetView program to another requires bringing down both NetView programs.

MVS Subsystem Interface

The subsystem interface is used to receive system messages and enter system commands. With extended multiple console support (EMCS) consoles, the subsystem interface is used to receive commands, not messages. In a single system, multiple NetView programs can use the subsystem interface. Using the subsystem interface is optional. If you do not need to receive system messages or enter system commands from a NetView program, you do not need to have that NetView program use the subsystem interface.

Each NetView program that uses the subsystem interface requires:

- Two NetView address spaces: a NetView subsystem address space and a NetView application address space. The start procedures for each pair of address spaces must begin with the same 4 characters (the subsystem name). The subsystem name must be different from that of any other pair of NetView address spaces in the same system. The NetView subsystem address space is used for receiving MVS system messages and entering system commands. It is also used for the NetView program-to-program interface.
- An active CNMCSSIR task.
- Subsystem-allocatable consoles for any operator or automation task that issues MVS commands. This step is required if you are not using EMCS consoles to receive MVS system messages.
- A unique designator character that is used to prefix NetView commands that are entered at an MVS console. The designator character is specified on the MVSPARM.Cmd.Designator statement in the CNMSTYLE member. The default designator character is a percent sign (%).

If multiple NetView programs in a single system run on the subsystem interface, they can communicate with each other over the subsystem interface by issuing write-to-operator messages (WTOs). Ensure that an occurrence of a message is automated by only one of the NetView programs.

Note: Use of the subsystem interface must be controlled by starting and stopping the NetView CNMCSSIR task rather than starting and stopping the NetView subsystem address space.

GENALERT

Generation of alerts using GENALERT does not require the VTAM CNMI. Therefore, two or more NetView programs in a single system can generate alerts. A NetView program without the CNMI can still have its own hardware monitor database for handling alerts. This can be useful if you are using alerts to display information about problems that you do not yet handle automatically. See "Monitoring Alerts with the Hardware Monitor" on page 362 for more information.

Any NetView program using the GENALERT interface requires:

- Unique hardware monitor databases
- Active BNJDSERV task
- Active DSICRTR task
- If this is NetView that is using the CNMI:
  - FUNCT=BOTH must be specified on the DSTINIT statement in BNJMBDST.
  - FUNCT=CNMI must be specified on the DSTINIT statement in DSICRTTD.
  - VTAM applications for DSICRTR and BNJHWMON must be activated.
- If this NetView program is not using the CNMI:
  - FUNCT=VSAM must be specified on the DSTINIT statement in BNJMBDST.
  - FUNCT=OTHER must be specified on the DSTINIT statement in DSICRTTD. Program temporary fix (PTF) UY25868 is required for this.
  - VTAM applications for DSICRTR and BNJHWMON must not be activated.

For a NetView program that does not use GENALERT or the hardware monitor, it is not necessary to activate the BNJDSERV task or associate the VTAM APPL definitions and NetView databases with that task. However, you can still require the DSICRTR task to be used by the session monitor.

Status Monitor and Log Browse

Only one NetView program can use the VTAM status monitor interface.

You can use an O SECSTAT statement in DSICNM to specify a secondary status monitor. This action prevents a NetView program from using the VTAM status monitor performance enhancement. Otherwise, the first NetView program to start uses the VTAM status monitor performance enhancement.

For more information about the VTAM status monitor performance enhancement, refer to *IBM Tivoli NetView for z/OS Installation: Getting Started*.

Using the Interfaces

To run two or more NetView programs on a system, use the techniques described in this section.

Functions are commonly split among NetView programs in one of two ways:

- Network functions versus system functions
- Problem determination versus automation

Separating Network Functions from System Functions

Separating network functions from system functions helps improve organization. If you have separate groups for system support and network support, separate NetView programs can provide independence for those groups.

One NetView program can handle system operation and automation, while another handles network operation, automation, and problem determination. In this case, the system automation NetView program uses the subsystem interface. Optionally, the system automation NetView program can have a GENALERT interface. The network NetView program uses the POI, the CNMI, and the hardware monitor local-device interface. Optionally, the network automation NetView program can also use the subsystem interface and GENALERT.

The advantages of this approach are:

- Network support and system support each have an automation table.
- Network messages, network alerts, and local alerts need not be sent to another NetView program for automation.

The disadvantages of this approach follow:

- Some error situations might require coordination of network and system automation.
- Network automation competes for resources with problem determination functions in the problem determination NetView program according to NetView task priorities.
- Correlation of data among logs might be required.
- Procedures might be required to shut down network management tasks during network failure situations to improve performance.

Separating Problem Determination Functions from Automation Functions

Separating traditional network management functions, such as problem determination from automation functions, can improve performance. If you do not want network-management activities to affect automation performance, separating those functions in different NetView programs enables the automated NetView program to run at a higher priority than any of the tasks it automates, while the network management and problem determination NetView program can run at a priority that does not interfere with subsystems and applications. The automation NetView program can include both system and network automation.

With this kind of separation, the problem-determination NetView program uses the POI, the CNMI, and the hardware monitor local-device interface, and optionally, the subsystem interface. The problem-determination NetView program also runs the status monitor. The problem-determination NetView program does some automation, such as recovering the automated NetView program and routing unsolicited VTAM messages, network alert messages, and local alert messages to the automation NetView program.

The automation NetView program does the bulk of the automation and can optionally generate alerts for any system messages and problems that it cannot fully automate. In this case, the automation NetView program uses the subsystem interface, and possibly the GENALERT and log-browse functions.

Migration

When migrating from an earlier release of NetView to the current release, NetView can run network-management or problem-determination functions while a test NetView program can run automation functions. When this migration situation occurs, the production NetView program uses the POI, the CNMI, and the hardware monitor local-device interface. The test NetView program uses the subsystem interface and optionally the GENALERT interface.

Communication between Two NetView Programs

You might need to establish communication between two NetView programs in a system because of NetView function restrictions. For example, a problem-determination NetView program might need to send unsolicited VTAM messages, local device alert messages, and network alert messages to an automation NetView program. You might also need communication between two NetView programs for backup and recovery purposes.

The vehicles available for setting up communication between two NetView programs in a single system are:

- LUC alert forwarding
- Command and message forwarding
- The LU 6.2 transports
- The MVS subsystem interface

LUC Alert Forwarding

You can use the NetView LUC alert-forwarding function to route alerts from one NetView program to another. Use the hardware monitor ROUTE filter to specify which of the records in the alert database you want to forward.

When an alert passes through the automation table, the DUIFECMV command processor is invoked. This command processor sends information to GMFHS and initiates GMFHS processing of the alert. If you are running multiple copies of GMFHS, you can change the domain to which GMFHS forwards alerts with the DUIFECMV parameter GMFHSDOM. Refer to the *IBM Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer's Guide* for more information.

Command and Message Forwarding

You can use the RMTCMD command to send commands from one NetView program to another. Messages resulting from the command you issue return to you on the issuing NetView program. By issuing a command that sends a message, such as the MSG command, you can also use the RMTCMD command for message forwarding.

Another way to forward commands and messages is with OST-NNT sessions. After an OST on one NetView program logs on as an NNT on the other NetView program, you can use the ROUTE command to forward commands. The OST can issue commands to the NNT and receive messages in response, and messages arriving at the NNT go across to the OST. You can use the NetView message-forwarding samples to accomplish OST-NNT communication, or you can create a forwarding scheme.

LU 6.2 Transports

You can use the MS transport or the high-performance transport for communication between two NetView programs, including two NetView programs on a single system. You can create LU 6.2 applications for both NetView programs

and send MSUs between them. Or you can write an application for one of the NetView programs and communicate with an application that is supplied with the NetView program on the other. For example, NetView supplies an NVAUTO application that can receive MSUs and pass them directly to the automation table.

MVS Subsystem Interface

To communication among more than two NetView programs, each program must use the subsystem interface. Therefore, do these activities:

- Define each NetView program as a subsystem
- Run a subsystem address space in each
- Start the CNMCSSIR task in each

See "MVS Subsystem Interface" on page 458 for further information about the requirements in each NetView program. Each NetView program can issue WTOs that go onto the subsystem interface, where they are available to other NetView programs. This approach does not depend on the VTAM program.

Automated Recovery of NetView

When running two NetView programs, each NetView monitors and recovers the other. Each NetView can look for system messages indicating that the other NetView has abnormally ended. You can also use proactive monitoring. For example, you can use an EVERY command to periodically send a message from one NetView program to the other and check for a response. An RMTCMD session or the subsystem interface can carry the message. If a response does not come back within a specified time, the issuing NetView program can assume that the other program needs recovery. With proactive monitoring, be sure to provide a way to turn the automation off so that you can bring down one of the NetView programs normally.

If two NetView programs are to monitor each other on MVS, they both need the subsystem interface, because each might need to issue an MVS command to restart the other NetView program. The subsystem interface is not required for issuing MVS system commands if you are using EMCS consoles.

Priorities

The dispatching priority of the NetView subsystem address space must be high.

The dispatching priority of a NetView application address space that performs system automation must be below that of the subsystem address space but higher than that of any address space it automates, except the resource management facility (RMF™) and generalized trace facility (GTF) address spaces.

Set the dispatching priority of a NetView application address space that performs network automation and network management functions below the priority of the VTAM program. Also, balance the priority of the NetView application against application priorities.

Chapter 33. Automation Tuning

These are methods that you can use to tune your automation and optimize performance:

- Log analysis program
- Multitasking and task priorities
- Automation table processing
- Hardware monitor alerts

For other methods, refer to the *IBM Tivoli NetView for z/OS Tuning Guide*.

You can generate an automation table usage report by using the AUTOCNT command. This report provides information about compare items and the level of automation taking place in your system. See "Automation-Table Usage Reports" on page 238 for more information.

You can also use the TASKMON and TASKUTIL commands to analyze automation workloads. Refer to the *IBM Tivoli NetView for z/OS Tuning Guide* for more information.

Log Analysis Program

When setting up your automation environment, decrease the amount of message traffic occurring within the network by suppressing unnecessary messages at their point of origin. Automate messages that occur frequently, but that do not require operator intervention. A sample log analysis program that analyzes both JES2 and JES3 logs helps identify those messages that are good candidates for suppression or automation.

NetView sample CNMS62J2 provides sample JCL to run the log analysis program. The JCL is set up to process a JES2 log. Customize the JCL for your environment. Changes you might need to make include changes in the sort routine, sort files, input file volume, and PARMS options. Although the program is written to analyze the message frequency for JES2 and JES3 logs, you can modify it to analyze the message frequency for any log. Instructions for changing the program to analyze additional logs are included in the comments at the beginning of the program.

Options available for the program include specifying the type of log to be processed, start and stop times, time intervals, and filtering. In the example in Figure 172 on page 464, the parameter values passed into the program are a log type of JES2, a starting time of 01:00 (which means that only those records with a time stamp equal-to or later-than 01:00 are to be considered for the report), a stop time of 03:00 (which means that only those records with a stop time of 03:00 or earlier are to be considered for the report), and a time interval of one hour (which means that the report is to be broken into subreports every hour). In addition, filtering was turned off.

```
06/12/10 (000612)              JES2 SYSLOG MESSAGE FREQUENCY ANALYSIS                       01:01:17
-------------------------------------------------------------------------------------------------------
PARAMETER VALUES:
-------------------------------------
LOG TYPE:      JES2
START TIME:    01:00:00
STOP TIME:     03:00:00
TIME INTERVAL: 01:00:00
FILTERING:     OFF
06/12/10 (000612)              JES2 SYSLOG MESSAGE FREQUENCY ANALYSIS                       01:01:17
-------------------------------------------------------------------------------------------------------
+++++ ANALYSIS STATISTICS +++++
CURRENT INTERVAL: 000612 01:01:17 - 000612 02:01:16
NUMBER OF SECONDS ANALYZED IN LOG:        3600
TOTAL NUMBER OF MESSAGE LINES:           27876
TOTAL NUMBER OF UNIQUE MESSAGE IDS:         63
AVERAGE NUMBER OF MESSAGES/SECOND:        7.74
+++++ MESSAGE ID FREQUENCY ANALYSIS +++++
MESSAGE              INDIVIDUAL  CUMULATIVE
   ID     FREQUENCY  PERCENTAGE  PERCENTAGE  PARTIAL TEXT OF FIRST OCCURRENCE
=======================================================================================================
$HASP250    6300        22.60       22.60    RWJBB13  IS PURGED
$HASP530    6130        21.99       44.59    TJRMB13  ON L9.ST1             2 RECORDS
$HASP540    6002        21.53       66.12    RWJBB13  ON L9.SR1             2 RECORDS
$HASP534    3884        13.93       80.05    L9.ST1   INACTIVE
IKJ574I     2342         8.40       88.46    NO SPACE IN BROADCAST DATA SET FOR MAIL
ACT510I      472         1.69       90.15    #CLR019V.MSG      STEP WAS NOT EXECUTED
$HASP100     288         1.03       91.18    U$MASSU  ON L37.JR1     PROTO.USING
IEC141I      254         0.91       92.09    COMMSP.AS2.NMSD4W.ACNMCLST
ICH70001     171         0.61       92.71    DATAMGT  LAST ACCESS AT 01:00:29 ON TUESDAY, SEPTEMB
$HASP373     171         0.61       93.32    #CLR032V STARTED - INIT  9 - CLASS 0 - SYS C33R
:
:
=======================================================================================================
TOTAL:     27876
```

Figure 172. Log Analysis Program Output

The example in Figure 172 shows only the first 10 messages that were found in the log in the first time interval. As you can see, 27876 messages were found between 01:01:17 and 02:01:16 with 63 different message IDs. At that rate, the operator monitoring the extended multiple console support (EMCS) console sees an average of 7.74 messages per second displayed on the console.

By looking at the cumulative percentage, you can see that the first 10 messages listed in the report account for 93.32% of the messages written to the log in that period, which is not unusual. Generally, the 10 most frequently generated messages account for at least 80% of the messages in the log.

Upon analyzing the messages in the report, you can identify messages that are not really necessary for an operator to see to manage the environment. As a result, you might choose to suppress those messages from the operator's view. If you choose to suppress several of the messages, you might want to rerun the program with filtering on. By listing the messages that you want to have filtered in the filter file, you can simulate what happens if the messages are actually suppressed.

In Figure 173 on page 465, the filter file contains five messages to be filtered. You can use a message processing facility (MPF) file as filtering input, because the format of the filter file can accommodate the MPF file format.

```
$HAP250
$HAP530
$HAP540
$HAP100
$HAP373
```

Figure 173. Messages to be Filtered

The resulting report starts with a listing similar to the one in Figure 172 on page 464, but with filtering set to ON. In addition, a second listing gives information about the messages in the log as if the filtered messages did not exist. Figure 174 shows the second listing.

```
06/12/10 (000612)              JES2 SYSLOG MESSAGE FREQUENCY ANALYSIS                        01:01:17
-------------------------------------------------------------------------------------------------------
+++++ FILTERING ANALYSIS +++++
ATTEMPTED FILTERING MESSAGE IDS:
-------------------------------------------------------------------------------
$HASP100       $HASP250       $HASP373       $HASP530       $HASP540
-------------------------------------------------------------------------------
+++++ ANALYSIS STATISTICS +++++
CURRENT INTERVAL: 000612 01:01:17 - 000612 02:01:16
NUMBER OF SECONDS ANALYZED IN LOG:        3600
TOTAL NUMBER OF MESSAGE LINES:            8985
TOTAL NUMBER OF UNIQUE MESSAGE IDS:         58
AVERAGE NUMBER OF MESSAGES/SECOND:        2.50
+++++ MESSAGE ID FREQUENCY ANALYSIS +++++
MESSAGE              INDIVIDUAL  CUMULATIVE
   ID     FREQUENCY  PERCENTAGE  PERCENTAGE  PARTIAL TEXT OF FIRST OCCURRENCE
=======================================================================================================
$HASP534    3884        43.23       43.23    L9.ST1   INACTIVE
IKJ574I     2342        26.07       69.29    NO SPACE IN BROADCAST DATA SET FOR MAIL
ACT510I      472         5.25       74.55    #CLR019V.MSG      STEP WAS NOT EXECUTED
IEC141I      254         2.83       77.37    COMMSP.AS2.NMSD4W.ACNMCLST
ICH70001     171         1.90       79.28    DATAMGT  LAST ACCESS AT 01:00:29 ON TUESDAY, SEPTEMB
$HASP395     170         1.89       81.17    #CLR019V ENDED
IEF404I      170         1.89       83.06    #CLR019V - ENDED - TIME=01.02.01
IEF403I      170         1.89       84.95    #CLR032V - STARTED - TIME=01.02.06
IKJ144I      147         1.64       86.59    UNDEFINED USERID(S) NCPRACF
$HASP520     131         1.46       88.05    NOTIFY   ON L35.JT2
:
=======================================================================================================
TOTAL:      8985
```

Figure 174. Log Analysis Program Output with Filtering

With filtering, only 8985 messages were found between 01:01:17 and 02:01:16 with 58 unique message IDs. By filtering five of the 63 unique message IDs in the file, you reduce the number of messages displayed to the operator by 68%, making the average number of messages that an operator sees each second drop from 7.74 to 2.50.

Analyzing messages that are being written to the log must be an ongoing process. Analyze different time intervals. For example, if you have batch job streams running on second shift, a different set of messages might be generated for that shift than are generated on first shift. Because your system and network are constantly changing, so are the messages that you receive.

Resource Controls, Task Priorities, and Multitasking

NetView consists of many subtasks, each competing for storage and CPU cycles. The relative priorities for the different NetView subtasks affect the order in which work is done. The way you define NetView to the system affects its performance and therefore affects the performance of all NetView subtasks.

Resource Controls

NetView provides ways to measure automation resource usage and enables you to control the order of processing, reduce backlogs, and optimize performance. Specific limits can be set for any task using the DEFAULTS and OVERRIDE commands. The command parameters interact. Therefore, you might want to change the parameters in the following order: CPU, storage, message queuing, and I/O. Monitor the effects on all values after each change.

Attention: Excessively low values can degrade system performance.

CPU Usage

Use the MAXCPU parameter of the DEFAULTS and OVERRIDE commands to limit the CPU usage of a task. The TASKMON command and SMF data can be used to analyze CPU usage on a task-by-task basis.

Limiting the CPU usage for a task might result in better performance than altering the dispatch priority because limiting usage affects how tasks interleave work, instead of which task uses the CPU.

Storage Usage

The DEFAULTS command has MAXSTG, SLOWSTG, AVLSLOW, and AVLMAX parameters that can be used to change how tasks respond when using storage above certain limits. The OVERRIDE command can set SLOWSTG and MAXSTG values for each task. The TASKMON command and SMF data can be used to analyze storage usage on a task-by-task basis.

Use the AVLSLOW and SLOWSTG parameters to intentionally slow down automation that is using large amounts of storage. Limiting the storage for a task might result in a reduction of throughput, but can prevent sudden storage outages that cause loss of automation.

Use the MAXSTG and AVLMAX parameters to limit runaway growth by a specific task. This might result in the automation being disrupted for that task, but the benefit is that the rest of the tasks are not be affected.

Monitor the BNH162I and BNH163I messages. These messages tell you when the NetView address space is too small for the workload. If you receive BNH162I or BNH163I messages, consider raising the region size the next time you start NetView.

Message Queuing

The DEFAULTS and OVERRIDE commands have MAXMQIN and MAXMQOUT parameters that control the rate of message flow between tasks. You adjust flow rates to avoid excessive storage use or queuing delays caused by message traffic. Use the TASKMON command and SMF data to analyze message flow rates on a task-by-task basis.

The MAXMQOUT parameter controls how fast (KB per minute) a task can send data using the NetView message queuing service. Use the MAXMQOUT parameter for tasks that primarily send data (for example, CNMCSSIR).

The MAXMQIN parameter controls how fast (KB per minute) all tasks can send data to another task. Use the MAXMQIN setting for tasks that primarily receive data (for example, DSILOG).

Input/Output Usage

The DEFAULTS and OVERRIDE commands have a MAXIO parameter to control how fast (I/Os per minute) a task can run. The TASKMON command and SMF data can be used to analyze I/O rates to determine which tasks are heavy I/O users.

The use of LOADCL might help decrease the I/O rate.

Task Priority

Task priority in NetView is:

PPT The primary program operator interface task (PPT) has the highest priority of all tasks (including data services tasks (DSTs) initialized at priority 1).

OSTs and NNTs
Lower in priority than the PPT, but higher in priority than autotasks, operator station tasks (OSTs) and NetView-NetView tasks (NNTs) have a relative priority of 4.

Autotasks
Lower in priority than PPT and OST/NNTs, autotasks have a relative priority of 5.

DSTs Although DSTs are not a direct part of automation, they can influence automation performance, depending on their level of priority. Priorities for DSTs are specified in the CNMSTYLE member or by the START command. Information about the CNMSTYLE member can be found in *IBM Tivoli NetView for z/OS Installation: Getting Started*.

If high-priority automation procedures do not contain commands that the PPT cannot run, you can run these procedures under the PPT. However, you must run most automation procedures under an autotask. When invoking a procedure from the automation table, you can use the ROUTE keyword on the EXEC action to specify the executing task.

Multiple Autotasks

Use multiple autotasks to distribute the automation workload across multiple processors, taking advantage of multitasking to improve throughput. Throughput for an automation workload can be constrained by contention for system processors or by contention for the command list data control block (DCB), which synchronizes I/O to the command list data set. When the command lists are preloaded using the LOADCL command, no I/O is necessary to the command list data set, and therefore processor capacity is the only constraint on automation throughput. When multiple processors are used, additional autotasks beyond the number of processors probably do not offer a significant throughput improvement.

Multiple NetView Programs

If you plan to use additional NetView programs on a system, as described in Chapter 32, “Running Multiple NetView Programs Per System,” on page 455, set

the task priorities for the automation NetView program above the VTAM program and the applications that NetView is to automate. You can give the other NetView program a lower priority than your critical applications to minimize the impact of the second NetView program on application response time. Dividing the NetView workloads between different address spaces might not decrease the overall NetView system processor utilization and can increase overall NetView storage usage. However, dividing the workloads can improve automation responsiveness and availability.

Automation-Table Processing

Careful design of your automation table can yield substantial savings of processing time, because NetView periodically checks the table if you receive a large number of messages and MSUs. "Design Guidelines for Automation Tables" on page 231 describes a number of principles for good design of automation tables. This section summarizes the principles that directly affect processing efficiency:

- Use the message processing facility of your operating system to suppress as many messages as possible before they reach the automation table.
- Use BEGIN-END sections to structure your table and reduce the number of comparisons required for each message or MSU.
- Order the BEGIN-END sections according to frequency. Place the sections that handle frequently received data at the beginning of the automation table.
- Order statements within each BEGIN-END section according to frequency. Place the statements that handle frequently received data at the beginning of the section.
- For frequently received data that you do not automate, you can stop the NetView program from processing the whole table by placing a statement at the beginning of the table that specifies no action. For example, if you do not automate commands issued at a NetView terminal, you can add the statement in Figure 175 at the top of your table.

```
IF HDRMTYPE = '*' THEN ;
```

Figure 175. Preventing the Automation Table from Processing Commands

- Isolate slow functions to avoid calling them more times than necessary. Potentially slow functions include the DSICGLOB and MSUSEG, as well as any lengthy automation table function (ATF) that you write yourself. Place these functions in a BEGIN-END section or after a logical-AND (&). Call the function only for the messages and MSUs that need it.
- Avoid calling a command procedure to process a message or MSU if you can process it with the automation table alone.

Use the AUTOCNT command to generate an automation table usage report. This report indicates how many times each statement was compared to a message or MSU, and how many times the statement comparisons resulted in a match. See "Automation-Table Usage Reports" on page 238 for more information.

Use the AUTOMAN command to manage one or more automation tables. See "Managing Multiple Automation Tables" on page 248 for more information.

Hardware Monitor Alerts

If you are using hardware monitor alerts to display information about the automated environment, try to limit the number of alerts sent to operators.

For every alert that it processes, the hardware monitor checks to see if operators are viewing the Alerts-Dynamic panel. For each operator viewing this panel, the hardware monitor determines which alerts are sent to the operator and updates the operator's screen. The processing cost for updating the operator screens can significantly increase the cost of processing the event, statistic, and alert workload. Therefore, if your arrival rate for alerts is high, consider controlling the use of the Alerts-Dynamic panel, either with operator command authorization checking or with viewing filters.

Chapter 34. Automation Table Testing

Test the automation table thoroughly before putting it into production to ensure that it functions properly. To test an automation table, generate the messages and MSUs that you expect the table to handle and determine that automation produces the correct response to each message and MSU.

To test the automation table:

- Use the AUTOTEST and AUTOCNT commands to test the effects of messages and MSUs on an automation table.
- Set up automation procedures in a test environment that duplicates the production environment.
- Introduce automation incrementally, on a production system, and ensure that automation procedures are running after each step.

Notes:

1. The AUTOTEST and AUTOCNT commands can test only one automation table at a time. This automation table can include additional members through the use of the %INCLUDE statement.
2. You can use the AUTOMAN command to manage multiple automation tables. It also provides additional diagnostic capabilities. For more information, see "Managing Multiple Automation Tables" on page 248.

The following topics describe these methods and diagnostic procedures for the automation table.

Automation Table Testing

Use the AUTOTEST command to test an automation table. Use either a recorded input stream of messages and MSUs or messages and MSUs that are being processed by the active automation table. Also, you can record messages and MSUs to test an automation table in the future.

Two approaches to automation table testing are:

- Compare the active automation table processing in parallel with the test automation table processing. You can do this by using the report generated with the AUTOCNT command.
- Prerecord AIFRs (messages and MSUs) as they are processed by the active automation table. Then use these AIFRs to repeat a test as necessary until the automation results are satisfactory.

Starting Parallel Testing

To test an automation table in parallel with the active automation table, do these steps:

1. Activate parallel testing using a command similar to:

```
AUTOTEST MEMBER=TESTTBL,LISTING=TESTLST,SOURCE=PARALLEL,
               REPORT=TESTRPT
```

   This command starts the testing of automation table TESTTBL. An automation testing report (TESTRPT) is generated. A new set of statistics for automation

table TESTTBL is kept. If you also want to record AIFRs for future testing, add the RECORD keyword to the AUTOTEST command. For example,

```
AUTOTEST MEMBER=TESTTBL,LISTING=TESTLST,SOURCE=PARALLEL,
               REPORT=TESTRPT,RECORD=TESTRECS
```

AIFRs are recorded in member TESTRECS.

2. Reset the active table counters to match the table being tested:

   ```
   AUTOCNT RESET
   ```

3. Allow the test to run for a period of time as messages and MSUs are processed by both automation tables.
4. Create a report of the automation table use of both the active and the test automation table:

   ```
   AUTOCNT REPORT=BOTH,FILE=PRODTBL,STATS=DETAIL
   AUTOCNT REPORT=BOTH,FILE=TESTTBL,TEST,STATS=DETAIL
   ```

5. Examine the results at any time. To reset the counters and continue automation table testing, enter:

   ```
   AUTOCNT RESET,STATS=DETAIL
   AUTOCNT RESET,TEST,STATS=DETAIL
   ```

6. Compare the output generated by the AUTOCNT and AUTOTEST commands to determine whether automation table processing was satisfactory.

   Note: Some messages and alerts can be included in the test table count that are not included in the active table count because of incoming traffic. This is because both tables not being closed at the same instant.

7. Examine the automation table testing report to verify that the table logic is correct. Verify that key messages and alerts that arrived during testing were processed correctly (matched on the correct automation table statement). For an example of the testing report, see "Sample Report for the AUTOTEST Command" on page 473.
8. If the test was satisfactory, activate the test automation table using the AUTOTBL command. If the test was not satisfactory, change the test automation table and rerun the test against the recorded AIFRs. This procedure is described in the following topic.

Testing an Automation Table Using Recorded AIFRs

You can test an automation table using message and MSU AIFRs recorded by a prior AUTOTEST command, change the automation table, and analyze their effect using a constant set of input data.

Record AIFRs as follows:

1. Activate AIFR recording:

   ```
   AUTOTEST RECORD=TESTRECS
   ```

2. After a period of time, stop AIFR recording:

   ```
   AUTOTEST RECORD=OFF
   ```

You can modify the file containing the recorded AIFRs by deleting AIFRs that are not to be used for testing. To delete an AIFR from the file, delete the line beginning `!!------------` and subsequent lines up to but not including the next `!!------------` line.

1. Locate the AIFR to be deleted in the file. Below the AIFR in the file is a separator line beginning

   ```
   !!------------
   ```

2. Delete the AIFR data lines and the separator line following the AIFR.

After saving the recorded AIFR file, you can change the security key on the first line of the recorded file from S> to <S. This prevents subsequent AUTOTEST commands from overwriting the file. The AUTOTEST command requires a security key of S> or S< to be present in the first record of a file to be used as a source file for the command.

To start automation table testing using recorded AIFRs:

1. Reset the test automation table counters and begin gathering statistics:

   ```
   AUTOCNT RESET,TEST
   ```

2. Start automation table testing:

   ```
   AUTOTEST SOURCE=TESTRECS,REPORT=TESTRPT
   ```

3. Discontinue gathering statistics for the test automation table when message BNH382I is displayed, indicating end-of-file on the source data set:

   ```
   AUTOCNT REPORT=BOTH,FILE=TESTTBL,TEST
   AUTOCNT RESET,TEST
   ```

4. Analyze the reports and statistics to determine whether the automation table processing was satisfactory. Also, examine the automation table testing report to verify that the table logic is correct. If satisfactory, activate the new automation table using the AUTOTBL command. If not satisfactory, change the automation table and repeat this process until the results are satisfactory.

Use recorded AIFRs to compare processing between two automation tables. For example, to compare processing between TESTTBL1 and TESTTBL2:

1. Test automation table TESTTBL1 using the recorded AIFRs in the previous example:

   ```
   AUTOTEST MEMBER=TESTTBL1,LISTING=LIST1,SOURCE=TESTRECS,
            REPORT=TESTRPT1
   ```

2. Test automation table TESTTBL2 when message BNH382I is displayed, indicating that testing of TESTTBL1 has completed, begin:

   ```
   AUTOTEST MEMBER=TESTTBL2,LISTING=LIST2,SOURCE=TESTRECS,
            REPORT=TESTRPT2
   ```

3. Compare the reports found in TESTRPT1 and TESTRPT2 when message BNH382I is displayed, indicating that testing of TESTTBL2 has completed.
4. Examine the automation table testing report to verify that the table logic is correct.

Sample Report for the AUTOTEST Command

The AUTOTEST command produces a report that shows the messages and MSUs that were processed by the automation table being tested. Use the report and the AUTOTEST listing file to understand the automation table statements identified in the report. Items in the report to notice include:

1 The AUTOTEST report contains a 2-character security key in the first record. The letter R indicates that the report was produced by the AUTOTEST command, and the character > indicates that the report can be overwritten by a subsequent AUTOTEST command. You can change the > to a < which prevents the AUTOTEST command from overwriting the report. Similarly, the listing file produced by the AUTOTEST command has the letter L in the first position, and the recorded AIFR file has the letter S in the first position. The > or < character in these files also indicates whether the AUTOTEST command allows overwriting of the files.

2 There are three matches for message DSI077A in member TESTTBL1. The

matches are identified by sequence number 00120020, statement 2, and the statement with a label of MYLABEL1.

A formatted example of a report follows:

```
R>  1
>> Automation table test of member DSIPARM.TESTTBL1     Listing: LIST1
>> Time: 04/06/10 08:54:46 Requesting operator: OPER1    Source: TESTRECS

-----------> Input number: 1. Type = Message  --------------

LIST ''

Matches: 0 Comparisons: 1

-----------> Input number: 2. Type = Message  --------------

STATION: OPER1      TERM: NTB4L702

Matches: 0 Comparisons: 1

-----------> Input number: 3. Type = Message  --------------

HCOPY: NOT ACTIVE   PROFILE: DSIPROFA

Matches: 0 Comparisons: 1

-----------> Input number: 4. Type = Message  --------------

STATUS: ACTIVE      IDLE MINUTES: 0

Matches: 0 Comparisons: 1

-----------> Input number: 5. Type = Message  --------------

ATTENDED: YES       CURRENT COMMAND: LIST

Matches: 0 Comparisons: 1

-----------> Input number: 6. Type = Message  --------------

AUTHRCVR: YES       CONTROL: GLOBAL

Matches: 0 Comparisons: 1

-----------> Input number: 7. Type = Message  --------------

NGMFADMN: NO        DEFAULT MVS CONSOLE NAME: NONE

Matches: 0 Comparisons: 1

-----------> Input number: 8. Type = Message  --------------

NGMFVSPN: NNNN (NO SPAN CHECKING ON NMC VIEWS)

Matches: 0 Comparisons: 1

-----------> Input number: 9. Type = Message  --------------

NGMFCMDS: YES       AUTOTASK: NO

Matches: 0 Comparisons: 1

-----------> Input number: 10. Type = Message  --------------

IP ADDRESS:  N/A
```

```
Matches: 0 Comparisons: 1

-----------> Input number: 11. Type = Message  --------------

OP CLASS LIST: NONE

Matches: 0 Comparisons: 1

-----------> Input number: 12. Type = Message  --------------

DOMAIN LIST: NTVB4 (I) CNM02 (I) CNM99 (I) B01NV (I)

Matches: 0 Comparisons: 1

-----------> Input number: 13. Type = Message  --------------

ACTIVE SPAN LIST: NONE

Matches: 0 Comparisons: 1

-----------> Input number: 14. Type = Message  --------------

END OF STATUS DISPLAY

Matches: 0 Comparisons: 1

-----------> Input number: 15. Type = Message  --------------

LIST  KKK

Matches: 0 Comparisons: 1

-----------> Input number: 16. Type = Message  --------------

DSI077A 'KKK' STATION NAME UNKNOWN

Matches: 3 Comparisons: 7  2
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1
 02.  2                Statement Number TESTTBL1
 03.  MYLABEL1         Label            TESTTBL1

-----------> Input number: 17. Type = Message  --------------

LIST ABND

Matches: 0 Comparisons: 1

-----------> Input number: 18. Type = Message  --------------

DSI077A 'ABND' STATION NAME UNKNOWN

Matches: 3 Comparisons: 7
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1
 02.  2                Statement Number TESTTBL1
 03.  MYLABEL1         Label            TESTTBL1

-----------> Input number: 19. Type = Message  --------------

MSG ALL HI

Matches: 0 Comparisons: 1

-----------> Input number: 20. Type = Message  --------------
```

```
DSI001I MESSAGE SENT TO ALL

Matches: 1 Comparisons: 7
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1

-----------> Input number: 21. Type = Message  --------------

DSI039I MSG FROM OPER1    : HI

Matches: 2 Comparisons: 3
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1
 02.  00160020         Sequence Number  TESTTBL1

-----------> Input number: 22. Type = Message  --------------

DSI039I MSG FROM OPER1    : HI

Matches: 2 Comparisons: 3
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1
 02.  00160020         Sequence Number  TESTTBL1

-----------> Input number: 23. Type = Message  --------------

DSI039I MSG FROM OPER1    : HI

Matches: 2 Comparisons: 3
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1
 02.  00160020         Sequence Number  TESTTBL1

-----------> Input number: 24. Type = Message  --------------

DSI039I MSG FROM OPER1    : HI

Matches: 2 Comparisons: 3
Match Location         Location Type    Member
----- ---------------- ---------------- --------
 01.  00120020         Sequence Number  TESTTBL1
 02.  00160020         Sequence Number  TESTTBL1

-----------> Input number: 25. Type = Message  --------------

DATE

Matches: 0 Comparisons: 1

-----------> Input number: 26. Type = Message  --------------

CNM359I DATE : TIME = 14:59     DATE = 03/31/10

Matches: 0 Comparisons: 1

-----------> Input number: 27. Type = Message  --------------

AUTOTEST OFF

Matches: 0 Comparisons: 1

-----------> Input number: 28. Type = Message  --------------
```

```
BNH344I AUTOMATION TABLE TESTING IS NOT ACTIVE

Matches: 0 Comparisons: 1

-----------> Input number: 29. Type = Message  --------------

BNH337I NO TEST AUTOMATION TABLE IS LOADED

Matches: 0 Comparisons: 1

-----------> Input number: 30. Type = Message  --------------

AUTOTEST RECORD=OFF

Matches: 0 Comparisons: 1

>> End of automation table test. Time: 04/06/10 08:54:46
```

Using a Test Environment

If you are using a test environment, you can set up your applications to generate the messages and MSUs to be automated, or you can write a program that simulates the messages to be automated.

Using Applications

If, for example, you want to automate a payroll application, you can install a test version of the payroll application and structure the input data to generate the messages and MSUs that you want to automate. Then run the application to generate the messages and MSUs. Observe and verify the results. If you do not get the expected results, make the necessary corrections and repeat the test.

An advantage of this method is that the messages and MSUs come from the actual application and, if you choose your test cases carefully, are very similar to the messages you receive in production. However, installing test versions of applications and creating test cases might be expensive because of the effort, machine time, and other resources required. The expense can be lower if your application developers already have test versions and test cases that you can use.

Using a Simulator

For simple automation, it might be easier to write a simulator program. A simulator can read a file of required messages and issue them. A simulator can also create MSUs and pass them to the automation table with the assembler DSIAUTO macro or the PL/I and C CNMAUTO service routine. You can then compare the result with the expected result (for example, was the proper command or reply given?) and, if required, make corrections.

Message Simulation

A message simulator can read a file specifying the messages you want to test with and can issue each one in turn. For simple automation-table statements that only check the text of the message, the simulator need not be complex. A command list can generate messages with the proper text by using SAY in REXX or &WRITE in the NetView command list language. For automation-table statements that use only MSGID, TEXT, and TOKEN compare items, this method is all that is required. Be careful to issue each message precisely, with all characters in the correct locations.

For statements that check information other than the text, you might need a more sophisticated simulator. The message simulator might need to generate messages

with the correct route codes, action codes, and job names. If your automation table checks a message's MVS job name, the simulator must generate and run a job to generate the messages under the correct name.

Your input message file can contain all the information that your automation table requires. For example, in MVS you might have the message ID, routing codes, descriptor codes, message text, and job name. You can then write a command procedure to take the message file as input, set the correct system variables, and issue the messages.

A shortcoming of using a message simulator is that the messages are based on what you think they should look like rather than the real messages generated by the product or application.

A variation of the message-simulation approach is to write the program so that instead of reading a file of messages, the program reads an existing message log file and regenerates the messages based on the content of the log. The log you use can be an actual system or network log, or you can edit it to change the mix of messages to suit your test or to meet the input requirements of your simulator. The messages you test with are then based on actual messages that you received from applications while running in a production environment. With your simulator, you can generate them any time you want for test purposes.

MSU Simulation

You can simulate MSUs by using the assembler, PL/I, and C interfaces to the automation table. In assembler, the DSIAUTO macro passes an MSU through automation table processing. The CNMAUTO service routine provides the equivalent function in PL/I and C.

One method of creating an MSU to pass to the automation table is to use an MSU that you have previously captured and stored. You can have an automation table entry that selects MSUs and passes them to a command procedure. The command procedure can save the buffer in a file for later use. You can write a second command procedure to retrieve the saved buffer, reconstitute the MSU, and pass it to automation.

Another method of creating an MSU for test purposes is to manually compose an MSU data field. The format of the MSU is available in *System Network Architecture Formats*, or by basing your MSU on one that you have already received. You can then pass the MSU data field to the automation table. Use your MSU to test MSUSEG-based statements in the automation table.

Implementing Automation Incrementally

To minimize the risk of disrupting your environment, incrementally implement your automation with checkpoints at each step to confirm correct processing.

One technique for preparing to automate is to have your routines send notifications to operators or to a log instead of taking actions. The routines state what actions they would take if you had activated automation. Operators still perform the actual actions manually, but the notifications help you determine whether the automation can correctly intercept and automate a message or MSU. The same technique can also help you identify additional actions that you can automate.

When you are first introducing automation, notify operators or keep logs of all automated actions to ensure that the correct actions are being taken. After operators know that the automation is functioning correctly, you can reduce the notification level, eventually providing only the information needed for debugging a problem with the automation if required.

To introduce automation incrementally:
- Verify automation table matches.
- Verify automated action parameters.
- Verify timed commands.
- Check the effect of the automation.
- Ensure that autotasks process command procedures correctly.

Verifying Automation Table Matches

Use automation table usage reports to verify that messages and MSUs are being matched against automation table statements correctly.

You can add statements to the automation table with the actions commented out. Generate a detailed usage report for a certain period of time and examine it to ensure that statements are being compared and matched the correct number of times. Figure 176 shows an example of an automation table statement with the actions commented out.

```
* Automate message DSI123I
  IF MSGID = 'DSI123I' THEN
*      EXEC(CMD('CLIST1') ROUTE(ONE AUTO1 *))
*      DISPLAY(N)
  ;
```

Figure 176. Automation Statement with Actions Commented Out

For this method to work correctly, it is important to know how many automated messages and MSUs should have matched these statements during the time that usage statistics were being taken. When the statements have been verified, you can uncomment the actions and reactivate the automation table.

Verifying Automated Action Parameters

When the automation table calls a command or command procedure, it can pass information to the procedure being called. You can test to verify the information passed without actually processing the procedures:

- If your automation extracts tokens from a message and passes them to an automation procedure, write a test procedure that displays all of the tokens in the message. Use the test procedure in place of the automation procedure and verify that the information you want to pass is in the tokens you expected.
- Write a test procedure to be called from the automation table in place of the actual automation procedure. Have the automation table pass the name of the actual automation procedure and the parameters with which it would have been started.

  Your test command procedure can then do such things as:
  - Put a record in the network log or a file showing the parameters.
  - Analyze the parameters to see if they are correct.
  - Keep data on different possible parameters that each automation procedure receives.

Verifying Scheduled Commands

After you add timer commands to your automation, you can periodically examine the network log to determine whether the scheduled commands are being processed correctly.

For AT, EVERY, and AFTER commands, message DSI208I is issued and can be written in the network log. That message contains the ID associated with the timer command and the name of the command or command procedure that is to be initiated.

For the CHRON command, message BNH549I is issued if NOTIFY RUN=*taskname* was specified on the CHRON command. To determine the timer ID and command name that was initiated, closely examine message BNH549

```
BNH549 CHRON NOTIFY=eventname BY=issueoper ID=timerid
               ROUTE=runoper COMMAND='CHRON text'
```

Note the following values in message BNH549:

- *timerid* contains the ID associated with the timer command.
- *text* field contains the name of the command or command procedure.

You can use the TIMER command to modify a CHRON command, including the NOTIFY parameter (which is required for message BNH549I to be issued).

You can write a command procedure to add the timer ID and scheduled command name to a file that is easier to examine than the network log. To do this, add an automation table statement that extracts the timer ID and the name of the scheduled command from message DSI208I or BNH549I. Pass the ID and command name to a command procedure that records them in a file. REXX command procedures can use EXECIO to write records to a member. PL/I and C can use CNMSMSG to write records to a sequential log file. Assembler command processors can use DSIWLS to write records to a sequential log file.

Checking the Effect of Automation

To prevent an unexpected message from interrupting automation while it is running in a test environment, you can include a test of the environment in the statements. For example, the automated operator AUTO1 has a common global variable called TEST that is set to YES if the command lists are to operate in test mode and is set to NO or null if the command lists are to function normally.

If you want to test the `$HASP098 ENTER TERMINATION OPTION` message, after which you want to dump and recycle JES2, use the following automation table entry:

```
IF JOBNAME = 'JES2' & MSGID = '$HASP098' THEN
   EXEC(CMD('$HASP098') ROUTE(ONE *)) DISPLAY(Y) NETLOG(Y);
```

The REXX command list $HASP098 is shown in Figure 177 on page 481. Note that, for test mode, a message is sent to the operator (and to the system log) indicating the action that would have occurred if the TEST common global variable were not set to YES. You can use a test-case analysis tool to extract the message from the log and compare your actual results to your expected results.

```
/* $HASP098 Command list                                          */
/*   - Responds to message $HASP098 ENTER TERMINATION OPTION      */
/*   - If in 'TEST' mode, doesn't send response                   */
'GLOBALV GETC TEST HASP098_LASTREPLY'    /* Get global variables    */
IF HASP098_LASTREPLY = '' |              /* If first reply or       */
   HASP098_LASTREPLY = 'PURGE' THEN      /*    did purge last time  */
      REPLY_TEXT = 'DUMP'                /* Do a dump this time     */
ELSE                                     /* Did dump last time      */
   REPLY_TEXT = 'PURGE'                  /* So do a purge this time */
CMD_TEXT = 'MVS R '||REPLYID()||','||REPLY_TEXT /* Build the command*/
HASP098_LASTREPLY = REPLY_TEXT           /* Set LASTREPLY for next  */
IF TEST = 'YES' THEN                     /* Are we in 'TEST' mode?  */
   'WTO TEST OF $HASP098 - 'CMD_TEXT     /* Write test results      */
ELSE                                     /* 'LIVE' mode...          */
   CMD_TEXT                              /* Issue the command       */
```

Figure 177. $HASP098 Command List

Ensuring That Autotasks Process Command Procedures Correctly

Autotasks are important in automation for running commands and command lists and for scheduling commands. Because autotasks are unattended, identifying problems with autotasks as quickly as possible is necessary to ensure that automation procedures are started correctly and promptly.

The following techniques can help verify that autotasks and automated command procedures are processing as intended:

- During automation testing, turn on command list tracing for command lists that run under an autotask. The tracing puts command list statements into the network log as they are processing. You can analyze the log to determine the cause of any problems. Examples of trace commands that can be used are TRACE I for REXX, and &CONTROL ALL for the NetView command list language. After you verify the automation command list, you can turn off the tracing to avoid cluttering the network log.
- Log on to an autotask's ID and watch the autotask's console to ensure that the correct actions are occurring. While you are logged on to the autotask's ID, you can issue an OVERRIDE DISPLAY=YES command to ensure that any messages sent to the autotask are displayed.
- Look in the network log for message CNM493I, which is written to the log whenever a command or command procedure is processed by an automation table statement. You can prevent logging of this message by using the DEFAULTS command, the CNM493I automation table action, or the OVERRIDE command. You can determine from the message when a command was scheduled and what task it was to run on, among other things.
- Look in the network log for message DWO032E, which is written to the log whenever a command or command procedure should have been processed but the task that it was to run on was not logged on. This message can also be automated in the NetView automation table. When testing automation, you can use this message to determine automated actions that were not correctly scheduled.

Include some sort of autotask checking to ensure that autotasks remain active and able to work. When an autotask becomes inactive because it is stuck in a loop, is waiting indefinitely, or has logged off, the problem can be difficult to recognize and resolve. The advanced automation sample set (described in Appendix I, “The

Sample Set for Automation," on page 577), uses a technique to automatically notify you if a task is inactive for longer than a defined period of time. The technique uses timer commands to periodically send a command to each autotask and set a global variable indicating that an acknowledgment is due. If the autotask is because of be checked again but notification from the last check has not been received, a message indicating a possible problem is sent to the system operator. All of the sample set's autotasks are checked by one master autotask, which in turn is checked by the PPT. The PPT is active if NetView is running.

Using Debugging Tools

Inherent in the operating system and NetView program are several audit trails and tools to help you in determining whether automation is doing what you expect it to do and to assist you if things are not going as you had planned. The following sections describe the system and network logs, evaluation of unautomated messages and MSUs, the NetView automation table listing, and NetView automation table tracing. For more information about logging, see Chapter 35, "Logging," on page 487.

Using Logs

The MVS system log is mapped by the IHAHCLOG macro in your SYS1.MACLIB data set. In the IHAHCLOG macro, an 8-byte field called HCLREQFL contains installation exit and message processing facility (MPF) request flags. Bit 10 of the suppression flag (bytes 7 and 8) indicates whether MVS requested automation for the message. You normally set this bit by having the MPF entry for that message specify AUTO(YES) or by having an MPF .DEFAULT or .NO_ENTRY statement that applies to that message specify AUTO(YES). You define MPF entries in the MPFLST*xx* member of SYS1.PARMLIB. The NetView program processes the message only if the automation-requested flag is on.

If a message is not being automated, one of the first places to look is in the system log to ensure that the automation-requested flag is on. Bit 1 indicates whether MPF suppressed a message from display. This flag is of interest when you are trying to determine how effective your message suppression is.

Every time the automation table generates a command, NetView places a CNM493I message in the network log, unless message logging has been prevented using the DEFAULTS command, the CNM493I automation table action, or the OVERRIDE command. A key parameter in the message is the sequence number, which is taken from positions 73–80 of the automation table statement. So that the CNM493I message has value to you, ensure that your automation table entries have sequence numbers, and maintain the numbering when you update the table.

CNM493I also shows the command that was generated as an operand of the EXEC portion of the automation table entry. It tells you what was processed and what parameters were passed to the command list or command processor. However, your automation command lists and command processors can be called from places other than the automation table. For example, other command lists and command processors might call them, or a NetView operator or system operator might start them.

Therefore, you might want to include an MSG LOG statement at the beginning of each automation procedure that records in the network log the name of the procedure being called and the parameters that were passed to it. The advanced automation sample set demonstrates this technique. You might want to have a

NetView global variable that your command procedures use to check whether they should be in debug mode. In this case, the command procedures provide additional information on their execution.

Another tool is the &CONTROL CMD statement in the command list. This tool causes all commands issued in the command list to be displayed. You might set &CONTROL ERR inside of command loops. It also causes the message in Figure 178 to be written to the network log.

```
DSI013I COMMAND LIST cmdlistname COMPLETE
```

Figure 178. DSI013I Message Written by the &CONTROL CMD Statement

Evaluating Unautomated Messages and MSUs

As part of your testing and debugging, you might want to create a list of messages and MSUs that are not automated by your automation table. You can use the list to find messages or MSUs that did not trigger automation. You can also use the list to determine additional messages and MSUs that you can automate.

For example, if you write a command processor named LOGSEQ that records what is passed to it in a sequential log data set, you can put the statement shown in Figure 179 as the last statement in your automation table to pass all messages to the LOGSEQ command processor.

```
IF MSGID = ANYID THEN
      EXEC(CMD('LOGSEQ ' ANYID ' NOT AUTOMATED'));
```

Figure 179. Statement that Passes Messages to LOGSEQ

For each message that is processed without finding a match in the automation table, LOGSEQ writes a record to the sequential log file indicating that the message was not automated. You can then analyze the sequential log to see if messages that should be automated are not being automated. You can also use this technique for MSUs. Note, however, that this technique assumes that you allow unautomated messages and MSUs, and no others, to reach the bottom of your table. This situation is not suitable for all tables.

You can also use the same technique in BEGIN-END sections to determine whether messages or MSUs are being automated as intended. This technique can help you determine whether you need to add additional statements to the BEGIN-END section.

You can use this technique with condition items other than MSGID to obtain other information about unautomated messages and MSUs.

Using NetView Automation Table Listings

You can create a listing of your automation table with the AUTOTEST and AUTOTBL commands. Syntax errors are indicated in the listing. The listing also shows all the included automation members and the synonym substitutions, making it ideal for determining the order of automation statements. It also helps you prevent of debug logic errors in your automation tables by showing the entire automation table in one place.

An example of an automation table and its listing is shown in “Example of an Automation-Table Listing” on page 236.

Using NetView Automation Table Tracing

You can trace the processing of a message or MSU through the automation table using the TRACE action. The TRACE action sets a trace tag for the AIFR as well as an indicator that the AIFR is to be traced as it is processed by the automation table. Detailed trace information is displayed by message BNH370I for each part of each automation table statement that is processed.

An example of an automation table with a TRACE action is shown in 484. In this example, the intent is to determine why operator OPER1 is not receiving the message that the command is longer than 8 characters and therefore not valid. Note that when a command is longer than 8 characters, the DSI002I notification message is displayed as an immediate message rather than as a regular message. Therefore, HDRMTYPE is checked to determine whether the command name was longer than 8 characters.

```
SYN %HDRMTYPE_IMMED% = '''!''';

IF (LABEL:LABEL1) MSGID = 'DSI002I' THEN
  TRACE('DSI002_IMMED_TRC');

IF (LABEL:LABEL2) MSGID = 'DSI002I'  &
   TOKEN(4) = CMDNAME                &
   HDRMTYPE = %HDRMTYPE_IMMED%  THEN
  EXEC(CMD('MSG OPER1 COMMAND' CMDNAME ' IS LONGER THAN 8 CHARS'));

IF (LABEL:LABEL3) MSGID = 'BNH370I' THEN
  COLOR(YEL);
```

When message DSI002I is issued for a command that is not valid (SHORTCMD) and is processed by the preceding automation table segment, the following messages are produced:

```
SHORTCMD                                                                  1
BNH370I PASS TRACE       MAINTABL INCLTABL  LABEL1   DSI002_IMMED_TRC     2
BNH370I PASS MSGID       MAINTABL INCLTABL  LABEL2   DSI002_IMMED_TRC     3
BNH370I PASS TOKEN       MAINTABL INCLTABL  LABEL2   DSI002_IMMED_TRC     4
BNH370I PASS AND         MAINTABL INCLTABL  LABEL2   DSI002_IMMED_TRC     5
BNH370I FAIL HDRMTYPE    MAINTABL INCLTABL  LABEL2   DSI002_IMMED_TRC     6
BNH370I FAIL AND         MAINTABL INCLTABL  LABEL2   DSI002_IMMED_TRC     7
BNH370I FAIL MSGID       MAINTABL INCLTABL  LABEL3   DSI002_IMMED_TRC     8
DSI002I INVALID COMMAND: 'SHORTCMD'                                       9
```

BNH370I messages indicate the tracing results for trace tag DSI002_IMMED_TRC, which was specified in the TRACE action in the preceding automation table segment. The individual messages are explained as follows:

Key	Explanation
1	The command SHORTCMD is entered.
2	The TRACE action in the LABEL1 statement ran successfully because the DSI002I message produced matched the conditions for this statement. This means tracing for this message is now in effect as it continues processing through the automation tables.
3	The MSGID conditional matches in the LABEL2 statement. This is indicated by `PASS MSGID`.
4	The TOKEN conditional (used to place the command name that is not valid into a variable) matches in the LABEL2 statement. This is indicated by `PASS TOKEN`.
5	The logical AND that joins the MSGID and TOKEN conditionals is

successful in the LABEL2 statement. This is indicated by `PASS AND`. The logical AND operator is successful because its two operands (MSGID and TOKEN) were successful.

6 The HDRMTYPE conditional fails in the LABEL2 statement. This is indicated by `FAIL HDRMTYPE`. The HDRMTYPE check fails because the command was not longer than 8 characters. Message DSI002I is output as a regular message, rather than an immediate message.

7 The logical AND operator (that joins HDRMTYPE with the preceding logical AND operator) fails in the LABEL2 statement. This is indicated by `FAIL AND`. The logical AND operator fails because its second operand (HDRMTYPE) failed.

8 Because the preceding statement did not result in a match, automation table processing continues. The MSGID conditional fails in the LABEL3 statement. This is indicated by the `FAIL MSGID`.

9 Message DSI002I (just processed through the automation table) is displayed on the console.

Message BNH370I is issued to the console. You can include automation logic in the automation table to direct it to the NetView log if desired. Do *not* specify a TRACE action for message ID BNH370I, as this causes a loop condition to occur. Note that BNH370I was automated in the preceding example in order to color the trace message yellow for easier recognition.

Chapter 35. Logging

This chapter describes logging, which you can do at the system, network, or user level. The topics are:

- Considerations for logging
- Different kinds of logs
- Logging capabilities provided by NetView
- Differences between data in the MVS system log and data in the network log

Logging Considerations

Data that is to be recorded is usually in message format. However, the data can be commands, programmed data, or other information, depending on the purpose of the log in your particular environment. Some common uses of logs are:

- To provide an audit trail of events that have occurred in the system. Audit trails can be useful for tracking the automation process or when an operational problem occurs. In those circumstances, all information must be relevant, readable, and stored in a usable format. An audit trail is not considered a trace. For more information about using logs to help with problem determination, see "Using Logs" on page 482.
- To report on the operational characteristics of the system. For example, management might want a report on the effectiveness of automation in your system. Such data can be compared to data collected when automation was not available.

In all cases, the log is only as good as the data put into it. You have considerable control over the data that is kept and what logs it is written to. Therefore, you must decide on the logging strategy best suited for your environment. Data that is not meaningful should not go into the log. Keeping only the data that is useful ensures that the log is readable and also minimizes the performance overhead of logging.

NetView writes message CNM493I to the network log each time a successful match in the automation table results in a command or command procedure being scheduled. You can prevent logging of this message with the DEFAULTS command, the CNM493I automation table action, or the OVERRIDE command.

Refer to the *IBM Tivoli NetView for z/OS Tuning Guide* for guidelines on when to prevent the logging of message CNM493I. Message CNM493I has the format shown in Figure 180.

```
CNM493I member : seqnum : commandtext
```

Figure 180. Message CNM493I Format

In Figure 180, *member* is the automation table member for the statement that scheduled the command or command procedure, *seqnum* is the sequence number of the statement (or `(NO SEQ)`, if the statement has no sequence number), and *commandtext* is the command or command procedure scheduled, including any parameters.

You can use message CNM493I to analyze automation use and as an audit trail to determine if your automation is processing as intended. Message CNM493I

indicates only that the command or command procedure was scheduled, not that it was processed. If the task to which the command or command procedure was routed is not active, the command or command procedure is not processed and message DWO032E is recorded in the log and sent through the automation table. Message DWO032E provides the name of the inactive task and the command or command procedure that was not processed.

Consider what data to record, including the CNM493I message. For example, consider logging the following types of data:

- Log all messages for which some processing is done, even if only to collect data as part of a monitoring application.
- Log the command list name and parameter string for each major automation procedure that runs. Message CNM493I is issued only for the first procedure that runs as the result of a match in the automation table. Message CNM493I is not issued for other command lists and command processors that are subsequently called by that procedure.
- Ensure that a record is logged each time an action is taken that directly automates an operation previously taken by an operator. Specifically, you might want to record it in a way that facilitates recognition as an "automated action" and thus aid the production of management statistics.

The operating system controls logging of system messages. Suppress any system-message logging at the system level if possible. Messages that are processed by NetView can be suppressed or directed to the system log, the network log, or various user-provided logs, in addition to the hardcopy log.

MVS System Log (SYSLOG)

All MVS write-to-operator (WTO) messages, including those suppressed in MPFLST*xx* entries, are recorded in the MVS system log (SYSLOG). Logging to the system log can be suppressed from a user-written MPF installation exit. NetView messages can be directed to the MVS system log (see "Network Log"). You can also use the Message Revision Table, described in "Message Revision Table" on page 25, to suppress messages.

JES must be active to log messages to the MVS system log.

Note: Messages suppressed by the Message Revision facilities are written to the MVS SYSLOG. Mentions deleted by the Message Revision facilities are not written to the MVS SYSLOG.

Network Log

NetView messages are written to the network log as a default. However, unsolicited messages received from the MVS subsystem interface that are not given to a task with ASSIGN and have no automation table entry are not written to the network log.

Use the VTAM start parameter PPOLOG=YES if you plan to record VTAM messages in the network log. This technique ensures that all VTAM commands, except START and HALT, entered at an extended multiple console support (EMCS) console and all VTAM responses are recorded in the network log, regardless of the DEFAULTS and OVERRIDE specifications for the task that started the subsystem interface router task.

The network log task (DSILOG) must be active before you can log messages to the network log. The NetView subsystem address space must be active to receive system messages before system messages can be recorded in the network log.

You can use the NetView BROWSE command to view the network log. The commands available through the NetView BROWSE command are similar to those of the ISPF BROWSE command. BROWSE cannot be used from an EMCS console associated with an autotask or by a NetView-NetView task (NNT).

The network log function buffers messages before actually writing them to the log. Using default processing, NetView writes to the network log when the buffer is full. With an initialization option, you can use deferred write (DFR) instead, but it makes no significant difference because network log I/O is sequential and there is no insert or delete processing.

User-Provided Logs

With the sequential access method log support, you can define one or more sequential log tasks to write variable length records to user-defined logs. You can send data to a sequential log task from command processors written in assembler (using DSIWLS), PL/I, or C (using CNMSMSG). Refer to *IBM Tivoli NetView for z/OS Installation: Getting Started* for more information.

PL/I and C I/O services are also available if you choose to write command processors in a high-level language. The services enable you to create logging schemes and, when used in conjunction with the ALLOCATE and FREE commands, enable you to easily allocate and free data sets from within NetView.

NetView Logging Capabilities

With NetView, you can specify whether messages go to the MVS system log, the network log, or various user-provided logs. User-provided logs can be accessed only from user-written code.

NetView logs messages according to the following rules:

- The NetView DEFAULTS command determines how messages are logged in the absence of other logging specifications. The DEFAULTS command can be entered by any NetView user and must therefore be authority-checked to avoid use by unauthorized operators. Specify values in the CNMSTYLE member to ensure that the preferred options take effect when NetView is initialized.
- An individual NetView operator can use the OVERRIDE command to set up defaults for messages directed to a particular operator station task (OST).
- You can specify SYSLOG(Y) and NETLOG(Y) in the automation table to determine which messages or sets of messages go to the system and network logs, respectively.

  If you specify SYSLOG(Y) for an MVS message that MVS has already written to the system log data set, NetView does not write an additional copy.

For messages written to the system log as a result of a SYSLOG(Y) specification, the message text is preceded by three NetView fields: the domain identifier, the operator identifier, and a message type symbol (for example, a dash designates a command-facility message). The actual message identifier is the fourth token of the message text.

Messages generated within NetView can also be logged with the NetView MSG command. Specifying a destination of LOG causes the message to be written to the network log, if the log is active, depending on the DEFAULTS and OVERRIDE settings. Specifying SYSOP writes the message to the system log as well as displaying it at an EMCS console.

A similar service (DSIWLS) is provided to those users coding command processors in assembler. The user can code command processors in a high-level language (specifically, PL/I or C). The high-level language service routine CNMSMSG allows the user to write messages to the network log, a user-provided sequential log, or an external log such as SMF. Specifying SYSOP writes a message to the system log as well as displaying it at an EMCS console.

MVS System Log and NetView Network Log Records

There are several differences between the data logged in the MVS system log and the data logged in the network log. Both carry the text of the message, but for MVS WTO messages, the system log also contains information pertaining to the origin (job number, console) and disposition (such as route codes and MPF actions) of the message.

The system log also indicates which type of message (such as multiline or command response) was recorded. The system log indicates the system that originated the message. The network log indicates the domain on which the message was first received. Usually the name of the domain and the name of the system are chosen so that it is easy to associate them.

For a JES3 global processor, the messages from different processors in the complex all arrive at the global processor. If you are obtaining subsystem interface messages only from the global processor, identifying the message source in the network log can be difficult because all of the messages have the same domain and the system name is not recorded.

For MVS messages, the system log indicates the time that the message was issued. The network log indicates the time at which automation table processing was done for that message, which can be much later. For example, if the NetView task that is processing the automation table is currently awaiting operator input as a result of an AUTOWRAP NO command, the message is not processed by that task until normal task processing resumes. This situation also occurs if the NetView task is in a full-screen application such as the session monitor or a help panel (but not if the NetView task is in session with the status monitor).

The message is later written to the network log. Similar delays occur for messages that NetView writes to the system log as a result of automation table processing. Therefore, messages can be written to the network log in a different chronological order from that of their causes, which can cause difficulties in problem determination. You can avoid these difficulties by careful design of the flow of messages in NetView and by use of ASSIGN commands and ROUTE operands in the automation table.

Chapter 36. Job Entry Subsystem 3 (JES3) Automation

A job entry subsystem 3 (JES3) complex can consist of several channel-to-channel (CTC) connected processors that appear to the operator as a single system. Operational control of the JES3 complex is performed by one processor, the global processor. The global processor is a central point for entry of jobs, control of resources needed by jobs, and distribution of work to processors in the complex. If the complex consists of more than one processor, the other processors are called local processors.

Because of the number of processors in a complex, operational control can be a demanding task. The complex is operated from consoles attached to the global processor, and the messages from all processors appear on those consoles. System logging is also done at the global processor. Operator commands that control the processors in the complex are issued from the consoles connected to the global processor.

The global processor must be the focal point of automation. The NetView program must run on that processor. Messages from all processors in the JES3 complex appear on the subsystem interface of the global processor. Commands can be sent to all processors from the global processor. Therefore, it is possible to automate the JES3 complex by installing NetView only on the global processor. However, also consider installing NetView on local processors to automate a recovery procedure.

If operational tasks are automated, it might not be necessary to maintain the same structure of console usage. Several functions can be consolidated on the same console if the message traffic is reduced by MPF and automated actions.

Message Flow in a JES3 Complex

The following sections describe the flow of messages that originate on the global processor and those that originate on the local processor.

Messages That Originate on the Global Processor

All WTO messages issued on the global processor pass through MPF processing on the global processor, and indicators for suppression, retention, and automation are set in its work queue element (WQE).

- If the message identifier is listed in MPFLST*xx*:
  - Specific AUTO, SUP, RET, and USEREXIT specifications are used if they exist.
  - If the entry for the message ID does not have specific definitions (that is, for AUTO and SUP), the .DEFAULT entry is used.
  - If no .DEFAULT entry exists, the defaults AUTO(NO) and SUP(YES) are used.
- If no entry exists in MPFLST*xx* for the message ID:
  - The definition from the .NO_ENTRY statement is used.
  - If no .NO_ENTRY statement exists, the system defaults AUTO(YES) and SUP(NO) are used.
- Next the message is presented to MPF exits. The exits can alter suppression or automation specifications.
  - If an MPF exit was specified for the message ID, the MPF exit routine is processed.

- If no MPF exit was specified but an IEAVMXIT exit routine exists, IEAVMXIT is processed.

After MPF processing is complete, MVS puts the message on the subsystem interface of the global processor. Each subsystem inspects the message in the order determined by the subsystem names table. JES3 must be the primary subsystem so it sees the message first.

JES3 performs MSGROUTE processing on the message and presents it to its exit 57 (IATUX57) and exit 69.

Note: JES3 can affect the display of the message on a multiple console support. That is, the message might not be suppressed in MPF, but it might be suppressed by JES3.

After NetView performs any Message Revision processing, if that processing specified a delete action, then there is no further processing of that message after the subsystem interface. If a NETVONLY action was specified, then the message is passed directly to CNMCSSIR for automation, but otherwise is processed as for a deleted message.

If automation was requested by either the MPF or Message Revision, then NetView copies the message from the subsystem interface for automation.

Finally, the message flows from the subsystem interface to the multiple console support for possible display, retention, and hardcopy logging, depending on routing codes, MPF specifications, Message Revision processing, and the subsystem interface return code.

Messages That Originate on the Local Processor

All WTO messages issued on a local processor pass through MPF processing on the local processor, and indicators for suppression, retention, and automation are set in its WQE.

- If an MPFLST*xx* entry exists for a message identifier:
  - Its AUTO, SUP, RET, and USEREXIT definitions are used.
  - If an entry exists but no definitions exist, the .DEFAULT entry is used.
  - If no .DEFAULT entry exists, the defaults AUTO(NO) and SUP(YES) are used.
- If no entry exists in MPFLST*xx* for the message ID:
  - The definition from the .NO_ENTRY statement is used.
  - If no .NO_ENTRY statement exists, the system defaults AUTO(YES) and SUP(NO) are used.
- Next the message is presented to MPF exits:
  - If an MPF exit was specified for this message ID, it is processed.
  - If no MPF exit was specified but an IEAVMXIT exit routine exists, it is processed.

Then the message is broadcast on the MVS subsystem interface of the local processor. Each subsystem inspects the message in the order determined by the subsystem names table. JES3 must be the primary subsystem so it sees the message first.

JES3 performs MSGROUTE processing and presents the message to its exit 57 (IATUX57) and exit 69 (IATUX69)..

NetView performs any Message Revision processing.

If a NetView subsystem is running on the local processor and AUTO(YES) was specified or defaulted in MPF for the message, NetView copies the message for automation.

After subsystem interface processing, the message goes to multiple console support on the local processor for possible display, retention, and hardcopy logging, depending on the current status of routing codes, suppression and retention indicators, the subsystem interface return code, and JES3 MSGROUTE specification.

Console support on the local processor copies the message and passes the copied message across an XCF connection to the global processor. JES3 does not alter the suppression, automation, or retention indicators of the original message in the local processor.

Console support on the global processor receives the message from an XCF connection on the local processor and processes it as follows:

- The message is presented to MPF exits if requested by the JES3 GLOBMPF parameter.
  - If an MPF exit was specified for the message ID, the MPF exit routine is processed.
  - If no MPF exit was specified but an IEAVMXIT exit routine exists, IEAVMXIT is processed.

After subsystem interface processing, the message goes to multiple support console for possible display, retention, and hardcopy logging at the global processor, depending on the routing codes, MPF specifications, and subsystem interface return code.

Commands in a JES3 Environment

This topic describes several ways of issuing commands in a JES3 environment.

Issuing JES3 Commands from NetView

JES3 commands can be issued from NetView on the global processor. The JES3 identifier (for example, `*`) is required when entering JES3 commands from NetView. JES3 commands can be prefixed with `MVS` when entered from the NetView program, or entries from NetView sample member CNMS6403 can be added to the CNMCMD member of DSIPARM to allow JES3 commands to be entered from the NetView program without the `MVS` prefix. This allows NetView operators to issue certain JES3 commands without being authorized to issue the MVS command. When a JES3 command is issued from NetView without the `MVS` prefix, the command must be followed by a space rather than a comma.

JES3 command verbs are subject to security, but keywords and values on the commands are not. For example, you can protect the *SEND command with command authorization using either the NetView command authorization table or a system authorization facility (SAF) product, such as RACF (Resource Access Control Facility). However, anything sent using this command is not subject to protection. Refer to the *IBM Tivoli NetView for z/OS Security Reference* for a description of command authorization.

JES3 commands issued from NetView on the global processor go onto the subsystem interface and then to JES3. Messages issued as a result of a JES3

command issued by NetView go on the subsystem interface where NetView can access them. The messages are not considered command responses but appear to NetView as unsolicited messages. JES3 does not issue multiline WTOs (MLWTOs), so what looks like a group of messages in response to a command is really several separate unsolicited messages and must be handled as such by NetView automation.

Most JES3 commands cannot be issued at a local processor. The only JES3 commands that can be issued by NetView on a local processor follow:
- *CALL,DSI
- *CALL,VARYL
- *START,DSI
- *START,VARYL
- *CANCEL,DSI
- *CANCEL,VARYL
- *DUMP
- *RETURN

NetView running on a local processor can send JES3 commands to the global processor and receive the response using the MVS ROUTE command.

Issuing MVS Commands from NetView in a JES3 Complex

In a JES3 environment, NetView on a local or global processor can issue MVS commands to be run on that processor.

NetView on the global processor can use the JES3 *SEND command to direct a system command (that is, an MVS command) to a local processor and receive the response to that command. The MVS ROUTE command can also be used to send MVS commands to a local processor and receive the response.

Issuing NetView Commands from Operating System Consoles in a JES3 Complex

NetView commands can be entered from an MVS console if they are prefixed with the NetView subsystem designator character and if an autotask is already associated with that MVS console (that is, the NetView AUTOTASK command was issued with the CONS operand identifying the number of the console where the NetView command is to be entered).

NetView in a JES3 Environment

When NetView automation command procedures are used in a JES3 environment, these are some additional items to consider:

- For a message on the subsystem interface, NetView Message Revision can alter:
  - JES3 logging
  - Routing/display of messages for multiple console support consoles
  - multiple console support hardcopy logging

  If AUTO(YES) is specified in the MPFLST*xx* member, the NetView program makes a copy of the message for automation.
- MPFLST*xx* must be set to suppress unnecessary messages on each processor and to specify automation for every message to be automated. Messages are propagated to the global processor for automation. The automation NetView program can issue commands to any processor in the complex.

- Because the automation NetView program can receive messages from several processors, command lists must check the SYSID() function (REXX) or the &SYSID control variable (NetView command list language) to check where the message is coming from and then send commands back to the specific processor using *SEND or the MVS ROUTE command.

  The value of SYSID() or &SYSID is the name defined by the SYSNAME parameter of the IEASYSxx member (GRS name), as long as JES3 is not active. When JES3 is active, SYSID() or &SYSID is the name of the JES3 processor from the MAINPROC statement. Therefore, consider setting them to be identical, or having an easily recognizable association.
- JES3 does not use multiline messages (MLWTO). JES3 can issue the same message several times as a response to a command. The command list receives all those messages as a response, and normally it should wait for all of them. However, it is difficult to know in advance how many messages a command list will receive. Possible methods for finding out are:
  - First issue a JES3 command to find out the number of elements, for example the number of jobs in a specific queue. Then issue a more specific command to know how many messages to expect.
  - Include the N operand in your *INQUIRY command to specify the maximum number of messages you want to receive.
- When you want a command procedure to wait for solicited messages in response to a JES3 command, the command must always be included:
  - On the &WAIT control statement in a command list written in the NetView command list language
  - After the TRAP instruction in a command list written in REXX
  - After the TRAP command in a command processor written in a high-level language (HLL)

  Doing so ensures getting the solicited message back. If you put the JES3 command before the &WAIT or TRAP, certain responses might come back so quickly that the &WAIT or TRAP might not receive them.
- Sometimes repeated JES3 messages trigger an automation command list twice. One message is an action message with the automation table's REPLYID properly set. A second message is not an action message but has the same message ID; the REPLYID condition item in this case is not set. The automation command procedure has to test whether the second message is a real action message. You can get the true reply ID by using:
  - The &REPLYID control variable in a command list written in the NetView command list language
  - The REPLYID function in a NetView REXX command list
  - The CNMGETA service routine in an HLL command processor
- The network log does not show which processor issued a message. You might want to have your customization exits include the system ID in messages that they write to the network log.

Chapter 37. SNMP Trap Automation

This chapter describes an architecture in which SNMP traps are turned into SNMP trap automation CP-MSUs that are then passed to NetView automation.

Whenever a complete SNMP trap is received, whether over Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), an SNMP trap automation task builds a CP-MSU and passes it to NetView automation. Within the CP-MSU are GDS variables containing data from the trap.

The SNMP trap automation task

The SNMP trap automation task is a NetView Data Services Task (DST) used for receiving and automating SNMP traps. The SNMP trap automation task can be set up as a concurrent server (for SNMP agents that are TCP clients), a datagram receiver (for SNMP agents that send traps via UDP), or both. An SNMP trap automation task can be used to receive and automate SNMPv1, SNMPv2c, and SNMPv3 traps in both IPv4 and IPv6 networks.

In order to distribute the SNMP trap automation workload, multiple SNMP trap automation tasks can be used. Each SNMP trap automation task within an instance of NetView must have a unique task name; however, the same DST initialization member (sample CNMTRAPI) is used for all of them.

An SNMP trap automation task automates an SNMP trap by converting the trap to a CP-MSU and passing that CP-MSU to NetView automation. The automation of MSUs is described in Chapter 22, "Automating Messages and Management Services Units (MSUs)," on page 317. The contents of the CP-MSU that an SNMP trap automation task builds are described in this chapter.

Configuring an SNMP trap automation task

An SNMP trap automation task is configured by setting common global variables that are unique to a specific task. These variables are read by an SNMP automation task. Additional detail on these variables can be found in the *IBM Tivoli NetView for z/OS Administration Reference*. These are the common global variables read by an SNMP trap automation task:

CNMTRAP.taskname.CONFIGFILE
This variable provides the name of a configuration file that contains SNMPv3 trap handling information.

CNMTRAP.taskname.MAXTCPCONN
This variable provides the maximum number of TCP connections supported by the SNMP trap automation task.

CNMTRAP.taskname.STACKNAME
This variable provides the name of the TCP/IP stack to which the SNMP automation task will obtain affinity.

CNMTRAP.taskname.TCPPORT
This variable provides a port number for a concurrent server (TCP) to which clients may connect and send SNMP traps.

CNMTRAP.taskname.TRACE
This variable enables or disables SNMP automation task tracing.

CNMTRAP.taskname.UDPPORT
This variable provides a port number to which entities can send SNMP trap datagrams (UDP).

As an example, consider these definitions that configure an SNMP trap automation task named CNMTRAPD:

```
COMMON.CNMTRAP.CNMTRAPD.STACKNAME = &CNMTCPN
COMMON.CNMTRAP.CNMTRAPD.TCPPORT = 162
COMMON.CNMTRAP.CNMTRAPD.UDPPORT = 162
COMMON.CNMTRAP.CNMTRAPD.CONFIGFILE =
                              /usr/lpp/netview/v5r3/cnmtrapd.conf
COMMON.CNMTRAP.CNMTRAPD.MAXTCPCONN = 50
```

CNMTRAPD would be expected to obtain affinity to a TCP/IP stack whose name was provided by the value of the &CNMTCPN symbol (typically the value of the TCPNAME keyword). CNMTRAPD would be expected to "listen" for traps on TCP port 162 and UDP port 162. UNIX System Services file /usr/lpp/netview/v5r3/cnmtrapd.conf (described in "SNMP trap automation task configuration file") would be expected to contain SNMPv3 trap processing information for the CNMTRAPD task. Because a CNMTRAP.CNMTRAPD.TRACE common global variable was not defined in this example, SNMP trap automation tracing would be disabled for CNMTRAPD.

SNMP trap automation task configuration file

If an SNMP trap automation task receives encrypted SNMPv3 traps, then it is necessary to provide the applicable pass phrase or key information in an SNMP trap automation task configuration file so that the SNMP trap automation task can decrypt and authenticate the trap data.

The SNMP trap automation task assumes an SNMPv3 trap is encrypted if the message flags field (msgFlags) in the SNMPv3 message header indicates both encryption (privFlag = 1) and authentication (authFlag = 1); otherwise, no decryption and no authentication procedures are performed.

The information for decrypting and authenticating an SNMPv3 trap is provided by a single statement in the SNMP trap automation task configuration file. Sample cnmtrapd.conf contains a description of the supported file characteristics, statement syntax, and examples. The file characteristics and line syntax are as follows:

- The file can have fixed or variable record format, and a logical record length of at most 2048 bytes.
- A file line is considered a comment if the first column contains an asterisk (*) or a number sign (#).
- A non-comment line has the following format (this is contained on a single line, but because of space constraints, must be displayed over two lines)

```
ipAddress port protocol snmpVer userName passPhrase
                    authAlg authKey privAlg privKey
```

where

ipAddress
Specify the IP address of a source of SNMPv3 traps.

port Specify a valid port number, in the range 1 - 65535. The port number is not currently used in the decryption and authentication process.

protocol
Specify a valid protocol (transport method), either **TCP** or **UDP**, The protocol is not currently used in the decryption and authentication process

snmpVer
Specify the SNMP version. You must specify a value of **snmpv3** for this parameter.

userName
Security name (user name in the SNMPv3 user-based security model being supported). The name may be from 1 to 32 characters, inclusive, in length.

passPhrase
A password used to generate authentication and decryption (privacy) keys for the username given above. A hyphen (–) can be specified if the key(s) are provided in subsequent parameters. Otherwise, the password can be from 8 to 64 characters, inclusive, in length. If a password is specified, then any keys provided in subsequent parameters are ignored.

authAlg
Algorithm used to generate a message digest for authenticating the SNMP trap PDU. Valid values are HMAC-SHA (Simple Hashing Algorithm 1, SHA-1) and HMAC-MD5 (Message Digest 5). This is also presumed to describe the algorithm that was used to produce any keys supplied on the statement,.

If a passPhrase is supplied, then this algorithm is used to generate the authentication and/or privacy keys for authenticating and decrypting, respectively, the SNMP trap PDU.

authKey
Character representation of hex digits representing an authentication key, presumably produced using the algorithm specified for authAlg. If authAlg is HMAC-SHA, this parameter should represent a 20-byte key. If authAlg is HMAC-MD5, this parameter should represent a 16-byte key

privAlg
Encryption/decryption (privacy) algorithm. The valid value is DES (data encryption standard with cipher block chaining).

privKey
Character representation of hex digits representing a privacy key, presumably produced using the algorithm specified for authAlg. If authAlg is HMAC-SHA, this parameter should represent a 20-byte key. If authAlg is HMAC-MD5, this parameter should represent a 16-byte key.

If you specify a value for privKey, you must also specify a value for authKey.

Note:

- Two items are used to find the applicable SNMP trap automation configuration file record containing decryption and authentication information: the origin IP address and the user name. The origin IP address is the connection peer, if the trap was received from a TCP client; otherwise, it will be the datagram sender, which is not necessarily the SNMP agent that sent the trap (particularly if an entity forwarded the

trap on the agent's behalf). The user name is extracted from the message security parameters (the user-based security model is presumed).
- There is no statement continuation. A statement must fit in one logical record.
- Symbols may be used within the statements. Symbols in a statement are substituted before the statement is processed.

These are examples:

- Example 1:

```
10.42.44.25 162 TCP snmpv3 adam adampassword HMAC-SHA - DES -
```

  This would be an entry for SNMPv3 traps originating from IP address 10.42.44.25 (currently applicable to both TCP and UDP and any origin port) and containing the user name **adam** in the message security parameters. The phrase **adampassword** is the pass phrase that the SNMP trap automation task uses to generate non-localized keys that are then used to authenticate and decrypt the trap.

- Example 2:

  Note: Because of space limitations in this text, this example extends over more than a single line; however, it should be construed as a single line.

```
10.42.44.25 162 UDP snmpv3 usermd5 - HMAC-MD5 67ef017ccb81111ba63b92e429338906
        DES 67ef017ccb81111ba63b92e429338906
```

  This would be an entry for SNMPv3 traps originating from IP address 10.42.44.25 (currently applicable to both TCP and UDP and any origin port) and containing the user name **usermd5** in the message security parameters. Here the keys required for authentication and decryption were provided, instead of a pass phrase. Notice that the keys are character representations of hexadecimal data, 16 bytes long, because HMAC-MD5 is chosen as the authentication algorithm.

SNMP Trap Automation CP-MSU

The CP-MSU contains GDS variables whose keys are in the range of context-dependent data to lessen the chance of overlapping with existing CP-MSU automation in the NetView program. For a description of context-dependent data, see *Systems Network Architecture Management Services Formats*, GC31-8302 , *Systems Network Architecture: Formats*, GA27-3136, or *Systems Network Architecture: Network Product Formats*, LY43-0081.

Each GDS variable within the CP-MSU SNMP trap automation begins with a 2-byte length followed by a 2-byte key. The length value includes the length of the length and key fields.

The first GDS variable within the CP-MSU contains information about the entity from which the SNMP trap originated. This GDS variable is always present.

Table 19. GDS variable within the CP-MSU

GDS variable key	Description
FFF0	SNMP trap origin

There are additional GDS variables within this GDS variable that describe the origin:

Table 20. GDS variables that describe the origin

GDS variable key	Description
FFF1	Origin IP address, standard text presentation form of an IPv4 address or IPv6 address, whichever applies.
FFF2	Origin port number, expressed as a character representation of the decimal port number.
FFF3	Protocol, IP transport over which the trap was received, may contain the characters TCP or UDP.

The next GDS variable within the CP-MSU includes all other GDS variables with data extracted from the trap. The GDS variable key depends on the type of SNMP trap:

Table 21. GDS variables that describe the type of trap

GDS variable key	Description
FFA4	SNMPv1 trap
FFA7	SNMPv2c or SNMPv3 trap

The GDS variables within the GDS variable contain the data as it occurred in the SNMP trap (without the tag and length). There are some exceptions to this, designed to assist CP-MSU automation and to allow the variety of data that might appear in a variable binding.

Agent IP address (SNMPv1 trap only)
The agent IP address is presented in the CP-MSU as a standard text presentation form of the IP (IPv4) address.

Data of type object ID
To make it easier to test for them and avoid having to perform a relatively complicated conversion of a BER-encoded object ID within automation, this is presented as a character representation of an ASN.1 object ID (decimal sub-identifiers separated by periods)

Data representing numeric quantities
For data types such as *integer*, *counter*, *gauge*, *timeticks*, and *counter64*, the value placed in the CP-MSU GDS variable always has a length large enough to represent the maximum value associated with the data type. That is, *integer*, *counter*, *gauge*, or *timeticks* data types are always be represented by a 4-byte value (the 32 bits required to hold the maximum value supported for the data type), while *counter64* data type is represented by an 8-byte value. If necessary, the value is padded on the left with zeros.

Note that SNMP trap automation does not do additional processing with constructor data types (for example, SEQUENCE). The value, without the tag and length, is simply placed as-is in the GDS variable.

The following table shows all of the GDS variables that can be created within an SNMP trap automation CP-MSU. It also notes the SNMP trap GDS variables in which the GDS variable might appear. GDS variables appear only once unless otherwise noted.

Table 22. GDS variables that can be created within an SNMP trap automation CP-MSU

Key	Description	Applicable SNMP trap GDS variables
FF00	SNMP version, integer data, always present	All
FF01	Community name, octet string data	FFA4, FFA7 only when an SNMPv2c trap
FF02	Enterprise object ID, object ID data	FFA4
FF03	Agent IP address	FFA4
FF04	Generic trap, integer data	FFA4
FF05	Specific trap, integer data	FFA4
FF06	Timestamp, timeticks data	FFA4
FF07	Request ID, integer data	FFA7
FF08	Error status, integer data	FFA7
FF09	Error index, integer data	FFA7
FF0A	Message ID, integer data	FFA7 only when an SNMPv3 trap
FF0B	Message maximum size, integer data	FFA7 only when an SNMPv3 trap
FF0C	Message flags, octet string data	FFA7 only when an SNMPv3 trap
FF0D	Message security model, integer data	FFA7 only when an SNMPv3 trap
FF0E	Message security parameters **Note:** See additional information about key FF0E in "GDS Variable Notes" on page 502.	FFA7 only when an SNMPv3 trap
FF0F	Context engine ID, octet string data	FFA7 only when an SNMPv3 trap
FF10	Context name, octet string data	FFA7 only when an SNMPv3 trap
FF11	Variable binding container, contains one *name* GDS variable X'FF12' followed by its corresponding *value* GDS variable X'FF13'	All, one for each variable binding in the original SNMP trap (if any)
FF12	Variable *name* from a variable binding, object ID data, occurs once for each variable binding in the trap	All, one for each variable binding in the original SNMP trap (if any)
FF13	Variable *value* from a variable binding, see note below about the format, occurs once for each variable binding in the trap and immediately follows the associated *name* GDS variable **Note:** See additional information about key FF13 in "GDS Variable Notes" on page 502.	All, one for each variable binding in the original SNMP trap (if any)

GDS Variable Notes:

1. The message security parameters GDS variable, key X'FF0E', contains 4 other GDS variables, as follows:

   FF1E Authoritative engine ID, octet string data

   FF2E Authoritative engine boots, integer data

FF3E Authoritative engine time, integer data

FF4E User name, octet string data

2. The *value* GDS variable, key X'FF13', has a format that must communicate the type of data contained within it. Its format is as follows:

   ```
   ....nnnnFF13ttttttttthhhhhhhhhhhh...
   ```

   where *ttttttt* is a 4-byte field at the beginning of the data in the GDS variable that communicates the type of data that follows. The value of this field is essentially the *tag* data taken from the value within the BER-encoded trap. The data is placed in the GDS variable subject to the guidelines described following Table 21 on page 501.

Notes:

1. The maximum size of an individual trap that the SNMP trap automation task accepts is 32500 bytes.
2. Because of the formatting conventions that are used and the origin information that is added, it is possible for an SNMP trap smaller than 32500 bytes to yield a CP-MSU that exceeds the maximum supported size (32600 bytes) of the CP-MSU. If the size is exceeded, the SNMP trap cannot be automated.
3. If an SNMPv2c or SNMPv3 trap follows the SNMP architecture, there are two variable bindings present, one for sysUpTime.0 and one for snmpTrapOID.0. The CP-MSU created by an SNMP trap automation task for such a trap would have an X'FFA7' GDS variable and two x'FF11' GDS variables, each one containing one X'FF12' GDS variable followed by one X'FF13' GDS variable.

For each of the different versions of SNMP traps, here are typical beginnings of SNMP trap automation CP-MSUs created from them. Note that each GDS variable within the SNMP trap automation CP-MSU begins with a 2-byte length followed by a 2-byte key. In these examples, the 2-byte length field is pointed to as the start of the GDS variable.

This is an example of an SNMPv1 trap. Note the *GDS variable* value X'FFA4' and the last byte of the *SNMP Version* value (X'00') indicates SNMP Version 1.

```
CP-MSU  Origin  Origin IP                     Origin          Protocol
|       |MV     |Addr GDS Variable            |Port           |GDS Variable
|       |       |                             |GDS Variable   |
LLLL12120022FFF0000FFFF1F1F04BF1F04BF1F6F34BF70008FFF2F1F0F2F40007FFF3E3C3D7

                  SNMPv1  SNMP            Community
                  |Trap   |Version        |Name GDS Variable
                  |MV     |               |
                  nnnnFFA4 0008FF0000000000 mmmmFF01hhhhhhhh...
```

This is an example of an SNMPv2c trap. Both SNMP Version SNMPv2c and SNMP Version SNMPv3 use a *GDS variable* value X'FFA7'; the last byte of the *SNMP Version* value (X'01') in this example indicates SNMP Version SNMPv2c.

```
CP-MSU  Origin  Origin IP                     Origin          Protocol
|       |MV     |Addr GDS Variable            |Port           |GDS Variable
|       |       |                             |GDS Variable   |
LLLL12120022FFF0000FFFF1F1F04BF1F04BF1F6F34BF70008FFF2F1F0F2F40007FFF3E4C4D7

                SNMPv2c SNMP            Community
                |Trap   |Version        |Name GDS Variable
                |MV     |               |
                nnnnFFA7 0008FF0000000001 mmmmFF01hhhhhhhh...
```

This is an example of an SNMPv3 trap. Both SNMP Version SNMPv2c and SNMP Version SNMPv3 use a *GDS variable* value X'FFA7'; the last byte of the *SNMP Version* value (X'03') in this example indicates SNMP Version SNMPv3.

```
CP-MSU  Origin  Origin IP                      Origin          Protocol
|       |MV     |Addr GDS Variable             |Port           |GDS Variable
|       |       |                              |GDS Variable   |
LLLL12120022FFF0000FFFF1F1F04BF1F04BF1F6F34BF70008FFF2F1F0F2F40007FFF3E3C3D7

                              SNMPv3  SNMP            Message ID
                              |Trap   |Version        |GDS Varible
                              |MV     |               |
                              nnnnFFA70008FF0000000003mmmmFF0Ahh...
```

Within the X'FFF0' GDS variable, note the trap origin information (IP address and port in character forms), as well as the name of a transport protocol, which can be TCP or UDP, over which the SNMP trap came.

In the SNMP version GDS variable, the value that appears for each SNMP version has been highlighted. Notice that the SNMP trap major GDS variable keys (X'FFA4' and X'FFA7') have the applicable SNMP trap constructor tag (X'A4' for SNMPv1 and X'A7' for SNMPv2c and SNMPv3) in the key.

For all types of SNMP traps, if no variable bindings exist in a trap, then no X'FF11' GDS variables are created in the SNMP trap automation CP-MSU. For each variable binding that does occur, an X'FF11' GDS variable is built as follows.

```
    VarBind Variable        Variable
    |Con-   |"Name" Variable|"Value"
    |tainer |GDS Variable   |GDS Variable
 ...LLLLFF11nnnnFF12objectIDmmmmFF13ttttttttthhhhhhhh...
```

With *nnnn* as the length of the *name* GDS variable (containing the character representation of the name; that is, object ID) and *mmmm* as the length of the *value* GDS variable (containing a 4-byte value representing the value's data type, followed by the value itself), *LLLL* is the total length of the variable container, and LLLL = nnnn + mmmm + 4.

Example of SNMP trap automation

To show a complete SNMP trap automation CP-MSU, consider this SNMP trap used in the detailed trap-to-alert conversion example in the *IBM Tivoli NetView for z/OS Customization Guide*.

```
*
* Outermost constructor for the trap (tag and length)
*
30820127
* SNMP version (00 = SNMPv1)
020100
* Community name (public)
04067075626C6963
* Trap PDU
A4820118
*   Enterprise object ID (1.3.6.1.4.1.12270)
06072B06010401DF6E
*   Agent address (10.71.225.20)
40040A47E114
*   Generic trap code (6 = enterprise specific)
020106
*   Specific trap code (32 in decimal)
```

```
020120
*   Timeticks
430402A2D49D
*   Variable bindings "container"
308200F9
*     Variable binding 1
3015
*       Variable 1 (1.3.6.1.4.1.12270.200.2.1.1.1)
060D2B06010401DF6E814802010101
*       Value 1 (octet string "1493")
040431343933
*     Variable binding 2
3019
*       Variable 2 (1.3.6.1.4.1.12270.200.2.1.1.2)
060D2B06010401DF6E814802010102
*       Value 2 (octet string "/L20/O50")
04082F4C32302F4F3530
*     Variable binding 3
3024
*       Variable 3 (1.3.6.1.4.1.12270.200.2.1.1.3)
060D2B06010401DF6E814802010103
*       Value 3 (octet string "2005-01-10T16:13:00")
0413323030352D30312D31305431363A31333A3030
*     Variable binding 4
3014
*       Variable 4 (1.3.6.1.4.1.12270.200.2.1.1.4)
060D2B06010401DF6E814802010104
*       Value 4 (octet string "I14")
0403493134
*     Variable binding 5
3025
*       Variable 5 (1.3.6.1.4.1.12270.200.2.1.1.5)
060D2B06010401DF6E814802010105
*       Value 5 (octet string "DIGIN ON    OCCURRED")
0414444947494E204F4E202020204F43435552524544
*     Variable binding 6
3015
*       Variable 6 (1.3.6.1.4.1.12270.200.2.1.1.6)
060D2B06010401DF6E814802010106
*       Value 6 (octet string "DI=1")
040444493D31
*     Variable binding 7
3025
*       Variable 7 (1.3.6.1.4.1.12270.200.2.1.1.7)
060D2B06010401DF6E814802010107
*       Value 7 (octet string "RC2 Gas Status Man. ")
0414524332204761732053746174757320 4D616E2E20
*     Variable binding 8
3011
*       Variable 8 (1.3.6.1.4.1.12270.200.2.1.1.8)
060D2B06010401DF6E814802010108
*       Value 8 (NULL)
0500
*     Variable binding 9
3011
*       Variable 9 (1.3.6.1.4.1.12270.200.2.1.1.9)
060D2B06010401DF6E814802010109
*       Value 9 (NULL)
0500
```

Suppose that this trap was sent on behalf of an entity from a TCP client at IP address 3A20:ABCD:70:1AB:10:10:173:7 and port 1038. The SNMP trap automation task that received this trap would produce an SNMP trap automation CP-MSU like this (GDS variables shown separately and annotated to show the data from the original trap to which the GDS variables correspond).

```
*
* Outermost CP-MSU container
*
026D1212
* Trap origin GDS variable
0033FFF0
* Origin IP address GDS variable ("3A20:ABCD:70:1AB:10:10:173:7")
0020FFF1F3C1F2F07AC1C2C3C47AF7F07AF1C1C27AF1F07AF1F07AF1F7F37AF7
* Origin port ("1038", character representation of the port number in decimal)
0008FFF2F1F0F3F8
* Transport ("TCP")
0007FFF3E3C3D7
* SNMP trap GDS variable (for the SNMPv1 trap)
0236FFA4
* SNMP version GDS variable (00 = SNMPv1)
0008FF0000000000
* Community GDS variable ("public")
000AFF017075626C6963
* Enterprise object ID GDS variable ("1.3.6.1.4.1.12270",
* notice EBCDIC character representation)
0015FF02F14BF34BF64BF14BF44BF14BF1F2F2F7F0
* Agent address GDS variable ("10.71.225.20")
0010FF03F1F04BF7F14BF2F2F54BF2F0
* Generic trap GDS variable (enterprise specific trap)
0008FF0400000006
* Specific trap GDS variable (32 in decimal)
0008FF05000000020
* Timestamp GDS variable
0008FF0602A2D49D
* Variable binding GDS variable 1
0031FF11
*  Variable name 1 (1.3.6.1.4.1.12270.200.2.1.1.1)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF1
*  Variable value 1 (00000004 = octet string, data = "1493",
*  notice unchanged ASCII data)
000CFF130000000431343933
* Variable binding GDS variable 2
0035FF11
*   Variable name 2 (1.3.6.1.4.1.12270.200.2.1.1.2)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF2
*   Variable value 2 (00000004 = octet string, data = "/L20/O50")
0010FF13000000042F4C32302F4F3530
* Variable binding GDS variable 3
0040FF11
*   Variable name 3 (1.3.6.1.4.1.12270.200.2.1.1.3)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF3
*   Variable value 3 (00000004 = octet string, data = "2005-01-10T16:13:00")
001BFF13000000043230303532D30312D31305431363A31333A3030
* Variable binding GDS variable 4
0030FF11
*   Variable name 4 (1.3.6.1.4.1.12270.200.2.1.1.4)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF4
*   Variable value 4 (00000004 = octet string, data = "I14")
000BFF1300000004493134
* Variable binding GDS variable 5
0041FF11
*   Variable name 5 (1.3.6.1.4.1.12270.200.2.1.1.5)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF5
*   Variable value 5 (00000004 = octet string, data = "DIGIN ON    OCCURRED")
001CFF13000000044444947494E204F4E202020204F43435552524544
* Variable binding GDS variable 6
0031FF11
*   Variable name 6 (1.3.6.1.4.1.12270.200.2.1.1.6)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF6
*   Variable value 6 (00000004 = octet string, data = "DI=1")
000CFF130000000444493D31
* Variable binding GDS variable 7
```

```
0041FF11
*   Variable name 7 (1.3.6.1.4.1.12270.200.2.1.1.7)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF7
*   Variable value 7 (00000004 = octet string, data = "RC2 Gas Status Man. ")
001CFF13000000045243322047617320537461747573204D616E2E20
* Variable binding GDS variable 8
002DFF11
*   Variable name 8 (1.3.6.1.4.1.12270.200.2.1.1.8)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF8
*   Variable value 8 (00000005 = NULL, no data following the data type)
0008FF1300000005
* Variable binding GDS variable 9
002DFF11
*   Variable name 9 (1.3.6.1.4.1.12270.200.2.1.1.9)
0021FF12F14BF34BF64BF14BF44BF14BF1F2F2F7F04BF2F0F04BF24BF14BF14BF9
*   Variable value 9 (00000005 = NULL, no data following the data type)
0008FF1300000005
```

Notice that object IDs are presented as EBCDIC character representations of their ASN.1 formats (decimal sub-identifiers separated by periods). This makes automation tests for them considerably easier.

Part 8. Appendixes

Appendix A. Planning for Migration to New Automation Capabilities in the NetView Program

This appendix includes general information in quick-reference form to help you plan for, and migrate to, the new automation capabilities provided by NetView. Some of these new capabilities result from automation improvements within NetView only, and some result from the NetView automation improvements working in conjunction with automation improvements in other products.

This appendix highlights the automation improvements for the various versions of the IBM Tivoli NetView for z/OS product.

NetView for z/OS V5R4 Program

Table 23 shows the items (components, functions, services, and support) that contribute to expanded and improved automation with the Tivoli NetView for z/OS V5R4 program.

Table 23. Automation Enhancements of the NetView for z/OS V5R4 Program

Enhancement	Description	For More Information, See
NetView MVS Command Revision	You can use the MVS Command Revision function to examine, modify, or delete (cancel) MVS commands. You can make complex changes, requiring a transfer to the NetView address space, that include getting a response to a WTOR, obtaining responses to other MVS commands, and reading files. The existing MVS Command Management function is still supported for migration purposes only, but is considered deprecated.	• Chapter 14, "The Command Revision Table," on page 135 • *IBM Tivoli NetView for z/OS Installation: Migration Guide*

NetView for OS/390 V1R4 Program

Table 24 shows the items (components, functions, services, and support) that contribute to expanded and improved automation with the Tivoli NetView for OS/390 V1R4 program.

Table 24. Automation Enhancements of the NetView for OS/390 V1R4 Program

Enhancement	Description	For More Information, See
Policy Services Overview	NetView Policy Services is a set of functions that enable dynamic policy-based management and automation of your resources. Policy Services includes: • Policy Engine APIs • Automation Policy Engine • Timer APIs	Chapter 16, "Policy Services Overview," on page 261

Table 24. Automation Enhancements of the NetView for OS/390 V1R4 Program (continued)

Enhancement	Description	For More Information, See
MVS Command Management	The MVS Command Management function enables you to examine, modify, or reject most MVS commands. You can specifically include or exclude commands from processing by command or by console names. The MVS Command Management function is still supported for migration purposes only, but is considered deprecated. The MVS Command Revision function replaces MVS Command Management.	• "Condition Items" on page 157 • *IBM Tivoli NetView for z/OS Installation: Migration Guide*

Appendix B. Sample Project Plan

This appendix provides a sample of an automation project plan for you to use as a model when you develop your own plan. The plan given here is only an example; it is not intended to apply to all environments.

The automation plan follows the four-phase approach used elsewhere in this book. Table 25 lists the phases and summarizes the activities involved in each phase. Although the phases are generally sequential, you might need to start one phase before you complete all parts of the preceding phase. That is, some tasks within a phase could depend on the completion of a task in a different phase. For example, you might need to complete the resource-definition task of the design phase (task 2.03) before you can perform the cost-justification task of the definition phase (task 1.13). Also, install automation on the test system (task 3.04) before you track and compare performance on the test system (task 2.05).

Table 25. Phases of the Project (Sample Project Plan)

Phase	Phase Name	Summary of Activities
1	Definition	Set up a planning team to manage and perform activities. Identify goals and objectives. Define short- and long-range goals for the automation project. Document the current operating environment and practices. Create a project plan for automation.
2	Design	Set up a design team to manage and perform activities. Design an implementation plan for automation and ensure that the plan meets your short-range and long-range goals and objectives.
3	Implementation	Set up an implementation team to manage and perform activities. Develop the procedures to automate operations in the areas identified, following the plan from the design phase. Code and test these procedures on a test system.
4	Production	Set up a production team to manage and perform activities. Install and test your automation procedures on production systems. Track system performance and revise procedures as necessary. Gather data to plan for the next stage of automation.

If you implement automation in stages, rather than all at once, you can perform the tasks in your automation plan repeatedly throughout the automation process. Experience and data gathered from one stage can help you improve your plan for the next stage.

An operating environment is dynamic. As you work on your automation plan, your organization might add new software and hardware to its systems and networks. Be prepared to accommodate the changes. Allow time in your schedules

to analyze changes, to evaluate automation with regard to new products, and to incorporate new products into the automation process.

If your organization has a thorough set of operating procedures, policies, and reports, you can accomplish many of the information-gathering tasks simply by collecting existing documentation.

Project Definition

Table 26 lists some tasks and subtasks for defining an automation project. For general information about the project-definition phase, see Chapter 3, "Defining an Automation Project," on page 41.

Table 26. Definition Phase

Task	Task Action	Subtask Activities
1.01	Hold an initial proposal session.	Conduct an initial proposal session for the project.
		Secure upper management commitment for the planning process.
		Identify who in management is responsible for automation project definition.
		Identify who in management is responsible for automation design, implementation, and production.
		Identify a project manager.
		Identify members of the project-definition team.
1.02	Hold an orientation session.	Conduct an orientation session with members of the project-definition team and the Tivoli branch-office systems engineer.
		Educate team members about automated operations.
		Establish a reporting process for project status.
		Investigate existing automation applications that can be purchased.
1.03	Analyze the organization's business environment.	Review business objectives.
		Review data processing objectives.
		Review the operations structure.
		Review operations-management disciplines.
		Identify tasks and objectives of each system-management discipline.
		Identify problem-resolution processes in operations areas.

Table 26. Definition Phase (continued)

Task	Task Action	Subtask Activities
1.04	Identify operating problems.	Interview operators.
		Analyze messages and MSUs.
		Analyze commands.
		Review service-level agreements.
		Analyze operator procedure books.
		Analyze problem-management reports and procedures.
		Analyze help-desk logs.
		Interview users.
		Interview management.
		Take measurements for tracking the success of automation.
1.05	Establish goals for automation.	Define measurable long-range and short-range goals.
		Quantify the benefits of automation to the organization.
		Set objectives.
1.06	Inventory operation resources.	Gather data about hardware.
		Gather data about software.
		Understand and document system and network configurations.
		Identify hardware and software to be installed.
1.07	Interview users.	Identify the number and operating requirements of users.
		Identify operator tasks that users could perform for themselves.
		Identify messages that could be distributed to users.
1.08	Establish a change-control group.	Establish a group to monitor change in the organization and in the operating environment.
		Establish a reporting procedure for this group.
1.09	Organize a design team.	Identify people to perform the design phase.
		Conduct education seminars that outline automated operations and the objectives of the design team.
1.10	Identify the roles of management in automation.	Identify the roles of management in the automated environment.
		Define the problem-resolution process for the automated environment.
1.11	Define the roles of personnel in automation.	Identify the roles of personnel in the automated environment.
		Identify job descriptions and major tasks for each role.
1.12	Establish education requirements for personnel.	Outline the education needs of personnel for the automated environment.
		Establish requirements for personnel training.
1.13	Justify automation.	Perform a cost justification for automation, using information gathered by both the planning team and the design team.

Table 26. Definition Phase (continued)

Task	Task Action	Subtask Activities
1.14	Develop a proposal.	Develop a proposal for automation. Prepare a final report that summarizes the design-team activities, findings, and recommendations. Present the proposal and the report to management.

Design

Table 27 lists some tasks and subtasks for designing an automation project. For general information about the project-design phase, see Chapter 4, "Designing an Automation Project," on page 51.

Table 27. Design Phase

Task	Task Action	Subtask Activities
2.01	Hold an initial design session.	Conduct a session with the planning team and the design team. Identify the roles of design-team members. Establish a reporting process for the design phase. Establish an approval process for automation design. Teach designers about automated operations.
2.02	Devise a high-level design for automation.	Review the findings of the project-definition team. Identify the future configuration of systems. Identify the operating procedures to automate. Identify the order in which systems and networks should be automated. Identify the order in which various programs should be automated.
2.03	Define resources for automation.	Define resources needed now and in the future: People Hardware and software products Facility support
2.04	Approve procedures before placing them on the test system.	Review and approve automated procedures before they are placed on a test system.
2.05	Track the performance of automation on the test system.	Analyze measurements of automation on the test system. Compare the results to measurements obtained in task 1.04.
2.06	Track the performance of automation on production systems.	Analyze measurements of automation on production systems. Compare the results to measurements obtained before automation.

Implementation

Table 28 lists some tasks and subtasks for implementing an automation project. For general information about the implementation phase, see Chapter 5, "Implementing an Automation Project," on page 61.

Table 28. Implementation Phase

Task	Task Action	Subtask Activities
3.01	Hold an initial implementation session.	Conduct a session with members of the other teams. Teach the implementation-team members about automated operations. Establish a reporting process for project status.
3.02	Attend training seminars.	Attend training seminars about automated operations. Learn established procedures for documenting automated procedures. Learn any new skills that are necessary to develop automated procedures, such as how to write command lists.
3.03	Install necessary products.	Install any products required to develop and test automation.
3.04	Perform the implementation plan.	Put the implementation plan created by the design team into use. Produce automation functions and procedures, following the guidelines established by the design team.
3.05	Install automation on the test system.	Install the automation on the test system.
3.06	Test the procedures.	Test the automation on the test system.
3.07	Measure and track performance.	Measure and track the performance of the test system with automation. Use the AUTOCNT command to generate an automation table usage report. Use the TASKUTIL command to monitor task performance and CPU utilization.
3.08	Review and tailor procedures.	Tailor procedures if necessary. Test the procedures again. Measure and track the performance of the test system again.
3.09	Approve the procedures.	Formally review and approve the automated procedures, based on testing and system performance. The group that approves the procedures should contain members from all of the automation teams.
3.10	Support the production team.	Assist in installing new products. Assist in tailoring automated procedures to the production systems. Monitor the results on production systems. Assist in testing any procedures that are tailored to production systems.

Production

Table 29 lists some tasks and subtasks for the production of an automation project. For general information about the production phase, see Chapter 5, "Implementing an Automation Project," on page 61.

Table 29. Production Phase

Task	Task Action	Subtask Activities
4.01	Hold an initial production session.	Conduct a session with members of the other teams. Identify the roles of production-team members. Teach members of the installation team about automated operations.
4.02	Install necessary products.	Install automation products on the appropriate systems. Conduct tests on the systems.
4.03	Migrate automation procedures.	Migrate automation functions and procedures, such as command lists and automation tables, to production systems.
4.04	Tailor the procedures.	Tailor and test all procedures to meet the requirements of each system. Use the AUTOCNT and AUTOTEST commands to generate an automation table usage report. Use the TASKMON and TASKUTIL commands to monitor task performance and CPU utilization.
4.05	Install program updates and enhancements.	Install and test any program updates and enhancements to automation products.
4.06	Tailor the enhancements.	Tailor and test any software or hardware enhancements to meet the requirements of the systems.
4.07	Review the procedures and train operators.	Periodically review automation and implement a plan for teaching operators about the automated environment. Compare measurement results to measurements taken before automation.

Planning Charts

You can use the following charts to calculate the time required to complete tasks in each phase of the plan.

Table 30. Planning Chart for Project Definition (Phase 1)

Task	Estimated Dates or Hours for Completion	Actual Dates or Hours for Completion	Person Responsible	Comments
1.01				
1.02				
1.03				
1.04				
1.05				
1.06				
1.07				
1.08				
1.09				
1.10				
1.11				
1.12				
1.13				
1.14				
Total				

Table 31. Planning Chart for Design (Phase 2)

Task	Estimated Dates or Hours for Completion	Actual Dates or Hours for Completion	Person Responsible	Comments
2.01				
2.02				
2.03				
2.04				
2.05				
2.06				
Total				

Table 32. Planning Chart for Implementation (Phase 3)

Task	Estimated Dates or Hours for Completion	Actual Dates or Hours for Completion	Person Responsible	Comments
3.01				
3.02				
3.03				
3.04				
3.05				
3.06				
3.07				
3.08				
3.09				
3.10				
Total				

Table 33. Planning Chart for Production (Phase 4)

Task	Estimated Dates or Hours for Completion	Actual Dates or Hours for Completion	Person Responsible	Comments
4.01				
4.02				
4.03				
4.04				
4.05				
4.06				
4.07				
Total				

Appendix C. Sample Progress Measurements

This appendix contains examples of objectives and indicators that you can measure. You can use these objectives and indicators to estimate the value of automation for comparison with the costs of automation. You can also use these objectives and indicators to measure the progress of your project.

Table 34 presents groups of indicators and measurements under several general objectives, such as the general objectives of "Improved operator productivity" and "Decreased complexity of operator tasks". The table includes columns for estimating costs, but you might not be able to assign costs to some of the indicators and measurements. Also, some of the indicators and measurements overlap; therefore, you should not try to total all of the costs that you might enter. You should choose objectives and indicators that relate to your organization's goals.

Note that all listed items may not apply to your system.

Table 34. Indicators and Measurements for Estimating the Value of Automation

Indicator or Measurement Related to a Goal	Before Automation	Cost	After Automation	Cost
Improved Operator Productivity Number of operators required Number of operators per system and subsystem Recovery time Number of console operators that monitor the system Number of system operations performed by operators				
Decreased Complexity of Operator Tasks Hours of manpower training required Number of consoles required Number of messages displayed Number of alerts displayed				
Decreased Human Error Number of procedure errors Number of console errors Number of outages				
Reduced Console Message Traffic Number of messages suppressed Number of messages automated Number of messages distributed Number of messages displayed				
Reduced Hardware Monitor Alert Traffic Number of alerts blocked (ESREC or AREC filters) Number of alerts automated Number of alerts displayed by the hardware monitor				

Table 34. Indicators and Measurements for Estimating the Value of Automation (continued)

Indicator or Measurement Related to a Goal	**Before Automation**	**Cost**	**After Automation**	**Cost**
Number of alerts displayed				
Reduced Problem Response Time Time from notification to corrective action				
Increased System Availability Number of outages Average outage time Average outage time by component Number of minutes recovery time Number of network restarts				
Centralized Operator Control Number of consoles required Response time for problem notifications Number of operators required for each system				
Centralized Reporting Process Number of logs Number of problem-management personnel Number of inconsistencies in logs Number of redundancies in logs				
Improved Management Control Number of procedure errors Number of incorrect responses Outage time in recovery Changes in service level agreements Number of errors in operation audit trails				
Fewer Constraints to Growth Number of console operators for a workload Number of consoles required Hours of manpower training required Number of operators required per system Time to install a new system Number of new procedures for a new system Time to connect new system to present environment				

Appendix D. MVS Message and Command Processing

This appendix documents information that is diagnostic, modification, and tuning information provided by NetView.

Attention: Do not use this diagnosis, modification, and tuning information as a programming interface.

Much of automated operations deals with the processing of messages issued by systems and their applications and subsystems such as IMS, CICS, JES2, and JES3. Similar considerations apply to the way commands are issued, whether they are issued from MVS, NetView, or other applications. This appendix helps you understand how messages and commands are handled by NetView using MVS.

Message Flow in MVS

In MVS, a normal message to the operator is created when a program issues one of the following requests:

WTO	Write to operator
WTOR	Write to operator with reply

These requests are processed by the WTO processor, which is part of the MVS supervisor. MVS creates a control block called a WQE (work queue element) for each message. The WQE contains the message text and all related information available for that message from the WTO parameter list and relevant information about the system at the time the WTO was issued.

Message Processing Facility

MPF allows an installation to influence how messages are to be handled. The important features of MPF are:

- Message suppression
- Exits
- Action message handling
- Automation

MPF carries out the message processing specifications expressed in the active MPF list (member MPFLST*xx* of SYS1.PARMLIB). The statements in that member specify two kinds of rules:

- The rules to be followed when MVS console support handles a message:
  - Display suppression
  - Action message retention
  - The display attributes for the message descriptor codes
- The rules by which automation facilities are driven by a message:
  - MPF installation exit selection
  - Automation subsystem selection

Processing done in an MPF installation exit occurs synchronously, in line with the WTO processor and the program that issued the WTO. Processing done in the automated subsystem in NetView occurs asynchronously to the WTO processor, and in parallel with the program that issued the WTO.

In MPF, the message is inspected to see if it should be marked for hardcopy log only by checking the SUP keyword for that message identifier. If the message is marked for hardcopy log only, it is not displayed at any multiple console support console. The marking is done by checking the AUTO and RETAIN® keywords for that message identifier.

Finally, the message is inspected to see if it should be passed to an installation exit routine. If so, the routine is loaded and control is passed to it. The WQE contains fields to indicate the results of MPF processing. The installation exit routine can update some fields.

Subsystems in Message Processing

Following MPF processing, the message is broadcast to all active subsystems. The message is presented to each subsystem in turn. Each subsystem can inspect the message and perform appropriate subsystem processing. The subsystem can alter the message text or other characteristics (WQE fields). The WQE contains fields to carry special job-related information supplied by the job entry system when it processes each message. Typical MVS subsystems include JES2, JES3, NetView, OPC/ESA, CICS, and IMS.

If a primary job entry subsystem (JES2 or JES3) is active, that subsystem is always the first to process a message, regardless of the subsystem's position in the IEFSSN*xx* member of SYS1.PARMLIB, which is referred to as the subsystems names table (SSNT). Following that, messages go to each subsystem according to the order specified in the SSNT. IMS is an exception to this rule. If IMS runs as an MVS subsystem, it has special code to ensure that it is the last subsystem on the subsystem interface.

Note: For JES2, if secondary job entry subsystems exist, their definitions should precede the definition for NetView in the list of subsystems in IEFSSN*xx*.

If you are using the subsystem interface for messages and an active NetView subsystem address space is present, it receives the WQE on the subsystem interface. The NetView subsystem selects messages to be passed to the NetView application and copies pertinent data from the WQE into its “message.” The WQE is not modified by NetView and passes on to the next subsystem, if any. The NetView subsystem queues the new messages in its own address space before they are dequeued by the NetView application. Messages are passed to the NetView application only while it is running and while an active subsystem interface router task is in that application.

If extended multiple console support (EMCS) consoles are being used by NetView, MVS facilities use cross-memory transfers to send messages directly to NetView tasks, which are defined as EMCS consoles.

The NetView subsystem checks to determine whether the message was marked for automation processing during MPF processing. If it was, the message text and selected attributes from the WQE are copied and queued into the NetView subsystem address space.

To direct a message for NetView automation, set the AUTO keyword for that message to AUTO(YES) or AUTO(*token*) in the MPF table. If you also want to suppress the message so it is not displayed for an operator, you can set the SUP keyword in the message processing facility (MPF) table to SUP(YES). However, if you do not direct a message for NetView automation and you also suppress the

message, the message is stopped at the MPF table and is effectively lost. Therefore, do not use both AUTO(NO) and SUP(YES) for the same message, unless you want to completely stop the message from passing through MPF.

EMCS consoles that are acquired by NetView cannot solicit messages by route code if the messages are marked with SUP(YES) in the MPF table.

Messages marked AUTO(YES) or AUTO(*token*) in the MPF table, or which are subject to NETVONLY or REVISE("1" AUTOMATE) revision table actions or similar, can be received on an EMCS console. By default, these messages are received by the CNMCSSIR task.

Note: JES2, JES3, and NetView allow you to have two or more copies of their subsystems active. Each copy requires a unique name in the SSNT, and each is called under its own subsystem name for a message. Under those circumstances, each NetView subsystem can select the message and queue it for processing by separate NetView applications.

Multiple Console Support

After a message is broadcast to all active subsystems, it is passed to multiple console support . If the message is not marked for the hardcopy log only, it is displayed at all multiple console support consoles with a matching routing code. Also, the RETAIN indicator is checked if it is considered by MVS as an action message or write-to-operator with reply (WTOR) message, and the MPF specifications for screen display colors are acted upon.

The message is then written to the hardcopy log, usually the system log data set. Writing the message to the system log data set can be prevented, within the MPF installation exits, but that is not usually done.

When using the subsystem interface for MVS message delivery in a sysplex environment, NetView avoids processing messages from other systems in the sysplex. This prevents duplicate automation if you run the NetView program on more than one system in the sysplex.

When using EMCS consoles for MVS message delivery in a sysplex environment, you can set the MSCOPE value for an EMCS console to enable a particular EMCS console to receive MVS system messages. These system messages can be from the console's own system, from all systems in the sysplex, or from a list of selected systems in the sysplex. By default, the EMCS consoles in use by NetView receive messages from all systems in the sysplex. The CNMCSSIR task is an exception to the default. This task receives messages only from its own system.

Command Flow

Commands issued from a multiple console support console are also broadcast on the subsystem interface. Commands are handled in a way that is similar to the way messages are handled. The command is inspected by each of the active subsystems, with the job entry subsystem first, followed by the others in the order they are specified in the SSNT.

Processing Determination

Each subsystem examines the command and determines whether to process it. For example, JES2 determines whether the first character is a dollar sign ($), assuming a typical use of JES2. If it is, JES2 accepts the command as one to be processed

within its address space and passes it on to its command processor. It also marks the command as having been processed by a subsystem, so that an error message for an unidentifiable command is not issued.

In a similar way, JES3 looks for an asterisk (*) or an eight (8), and the NetView program, by default, looks for a percent sign (%). NetView, like most subsystems, uses that identifying first character to distinguish its commands. If you run with two JES subsystems, you can direct commands to each individually by specifying a different character for each. If you specify the same character for each, both JES subsystems accept and process the command. The same is true of NetView. NetView also checks to see if an automation task (autotask) is in the NetView application address space associated with the console from which the command is issued. If that association exists, the command is copied and queued by the NetView subsystem address space. When it is subsequently dequeued by the NetView application, it is routed to the associated automation task.

Finally, if none of the subsystems have marked the command as having been processed, it is treated as an MVS command and the appropriate command processor is processed. If one does not exist, an error message is issued.

Commands Issued from a Console

When a command is issued from a console, the control block associated with the command (CSCB) indicates which console. That data is therefore available to the command processor and is usually used to issue a response message (using WTO) to the console that issued the command. NetView commands issued from a multiple console support console are processed by an autotask associated with that multiple console support console by reason of its console name.

A NetView autotask or operator can enter commands to MVS using the NetView MVS command, which causes an SVC 34 to be issued. To ensure that the responses are directed back to the issuer only, the NetView MVS command processor ensures that an MVS console has been obtained for the task that issues the MVS command.

Subsystem-allocatable consoles and EMCS consoles are not physical consoles. Instead, they are virtual consoles used by programs (such as JES2 and NetView) to selectively enter commands and retrieve output. When a NetView task enters an MVS command, an MVS console is obtained for the task unless the task already has an MVS console. The task retains the console as long as the task is active, unless you release the console using other NetView commands.

Note: Certain MVS or subsystem commands that do not issue responses using the console identifiers can produce unsolicited messages when solicited messages might be expected. These command responses are not delivered back to the issuer of the command because the command violates multiple console support designs.

NetView Interfaces with MVS

NetView is ideal as the focal point for automated operations because it has interfaces to all of the system, subsystem, and network components that create messages and to which commands are directed. NetView also provides programming capabilities to process messages through the automation table, a command list capability, and installation exits.

NetView consists of two address spaces. The NetView subsystem address space contains the program that interfaces to the MVS subsystem interface. It acts as a service address space for the NetView application address space. The NetView application address space contains the more familiar NetView programs, such as those that handle operators and process command lists.

Messages Issued as WTOs to Be Displayed or Processed by NetView

Messages issued as MVS WTOs can be processed using the subsystem interface or using EMCS consoles.

WTO Processing with the Subsystem Interface

Any number of NetView subsystems can be in an MVS system, each associated with a NetView application. The NetView subsystem receives messages from MVS WTO processing. If the message is marked AUTO(YES) or AUTO(*token*) in the MPF table, or the message is a command response, the NetView subsystem passes the message to the NetView application. Otherwise, message processing continues normally.

WTO Processing with EMCS Consoles

When EMCS consoles are used, MVS delivers system messages directly to EMCS consoles. The subsystem interface is not involved. Use the default EMCS console attributes. Note that solicited messages (command responses) come through the EMCS consoles and unsolicited messages come through the CNMCSSIR task.

You can change attributes for the EMCS console by using the RACF OPERPARM segment or the MVS VARY command. For example, you can specify that a specific EMCS console is to receive messages with certain route codes. However, changing the attributes of the EMCS consoles can result in duplicate automation of system messages.

MVS Commands Issued by NetView

NetView defines its commands using CMDDEF definition statements in its parameter library member named CNMCMD. Among the commands defined in the product sample CNMCMD is one called **MVS**. When the **MVS** command is issued by a NetView operator ID, the entire command operand string is sent to MVS as the text of an operator command from an MVS console assigned to that operator ID by the NetView MVS command processor.

If the subsystem interface is used, command response messages issued from MVS to the console that issued the command are selected by the NetView subsystem and routed back to the issuing operator ID. If EMCS consoles are used, command response messages are sent directly to the NetView operator task.

NetView Commands Issued as Subsystem Commands from an MVS Console

To issue NetView commands from an MVS console, three conditions must be met:

- The NetView subsystem must be active to recognize the command prefix character for NetView commands, which indicates that the remainder of the command text is to be passed to the NetView application for processing. The default command prefix character is the percent sign (%).
- The CNMCSSIR task in the NetView application must be running.

- An autotask must be running that was started with both a NetView operator ID and an associated MVS console.

NetView Commands Issued with MODIFY (F) Command from an MVS Console

To issue NetView commands from an MVS console using the MODIFY command, an autotask must be running that was started with both a NetView operator ID and an associated MVS console name.

Messages and Commands through VTAM Interfaces

The following sections describe how messages and commands are processed through various VTAM interfaces.

Terminal Access Facility

NetView uses the terminal access facility (TAF) operator control session to connect a NetView operator ID with VTAM applications that support LU1 sessions such as CICS and IMS. Messages from the application programs are packaged by TAF as NetView message buffers queued to the operator ID that owns the TAF operator control session. Similarly, the NetView SENDSESS command causes command text to be sent to the application program from the operator ID.

Interfaces

When the NetView primary program operator interface task (PPT) is active and defined as a VTAM application with AUTH=PPO, unsolicited VTAM messages are directed to NetView across the VTAM program operator interface (POI). If the POI is not active, unsolicited VTAM messages are issued by using WTO and are available to NetView across the MVS subsystem interface. When NetView submits commands to the VTAM program across the POI, the responses are correlated with the ID of the NetView operator that issued the command.

Communication Network Management Interface

Alerts can be received into NetView from VTAM as unsolicited data on the VTAM communication network management interface (CNMI). Also, MVS can send alert data to NetView with hardware monitor local-device records, whether VTAM is active or not. Finally, the GENALERT command processor in NetView can be issued to generate an alert for the hardware monitor to process, again regardless of whether VTAM is active.

Filters

When NetView receives alert data, the hardware monitor filters, set by NetView SRF commands, determine how that data is to be processed. One filter option can be used to format data from selected alerts into the text of NetView message BNJ146I. That message is then subject to all the usual NetView routing and automation processing.

Communication Network Management

VTAM communication network management (CNM) data can be received by NetView only when the NetView DSICRTR task is active and is defined as a VTAM application with an APPL name of DSICRTR and AUTH=CNM. VTAM's global routing table determines only one AUTH=CNM application to receive unsolicited data buffers of a given type. In that table, all alert data is routed to a single APPL named DSICRTR. Only a single NetView program per system can receive hardware monitor local-device records from the operating system. If your

system is running more than one NetView program, only one of the NetView programs can receive hardware monitor local-device records, and only one can open the CNM APPL named DSICRTR.

For more information about running two NetView programs in the same system, see Chapter 32, "Running Multiple NetView Programs Per System," on page 455.

To process the alerts from GENALERT in the NetView application that does not have the VTAM CNMI, you must:

1. Specify `DSTINIT FUNCT=OTHER` in the DSICRTTD initialization member for the DSICRTR task for handling NetView program-generated alerts.
2. Start the AUTH=CNM application in NetView by specifying a statement such as the one in Figure 181.

```
// EXEC PGM=BNJLINTB
```

Figure 181. Statement to Start the AUTH=CNM Application

3. Start the other NetView application programs by specifying the statement shown in Figure 182.

```
// EXEC PGM=DSIMNT
```

Figure 182. Statement to Start Other NetView Application Programs

Console Operations

MVS operator consoles are locally attached, system-allocatable devices, directly allocated to the system COMMTASK. They are usually 3270 display stations used to display WTO messages from operating system components, the job entry subsystem, the application subsystems, and the application programs that run in the system or local complex of systems.

Console operators enter system commands, subsystem commands, and program responses from those consoles. When the VTAM program is running under MVS, interfaces with the system MODIFY, REPLY, DISPLAY, VARY, and HALT commands allow system operators to issue those commands to VTAM from system consoles. VTAM messages are issued as WTO messages when no NetView PPT is available to receive them.

NetView consoles are 3270 display stations connected to NetView through VTAM logon processing. Using a NetView console, a network operator can log on using an assigned ID and password or password phrase. Through the NetView program's use of the VTAM POI, VTAM messages are displayed to network operators and VTAM commands are received from them. Messages issued from within the NetView program are displayed and NetView commands are received.

Using MVS Operator Consoles to Issue Commands and Command Lists as Subsystem Commands

From a system console, NetView looks similar to an MVS subsystem. Any NetView command or command list can be accepted by a NetView subsystem and passed to its associated NetView application if all of the following conditions are met:

- The subsystem name is defined to MVS.

In SYS1.PARMLIB, a subsystem names table member IEFSSN*xx* that is referenced in the system parameter SSN=(*xx*) contains a definition of valid subsystem names (refer to the MVS library).

- The NetView subsystem is active.

  A START command to activate the NetView subsystem is added to SYS1.PARMLIB member COMMND*xx* when NetView is installed.
- The command prefix character defined in the PARM field of the NetView start procedure is used as the first character of the command entered at the MVS console.

  The default command prefix character specified in the sample start procedure provided with NetView is the percent sign (%). To change the prefix character, follow the directions in *IBM Tivoli NetView for z/OS Installation: Getting Started*.
- The NetView application with a job name (used in the START command) that begins with the 4-character subsystem name is currently active with a fully started CNMCSSIR task.

  The TASK statement to define the CNMCSSIR task is supplied in the CNMSTYLE member and should be used without change.

  The NetView command STARTCNM ALL can be driven by specifications made in the CNMSTYLE member. Information about the CNMSTYLE member can be found in the *IBM Tivoli NetView for z/OS Installation: Getting Started*.
- An AUTOTASK command has been entered to associate a multiple console support console with the NetView program.

  If a NetView command is issued from a console that has no autotask, an error message is returned from the NetView subsystem, indicating that the console is not authorized to use the NetView subsystem. An AUTOTASK statement in the CNMSTYLE member starts an autotask for AUTO2 and an MVS console. Add a similar statement to the CNMSTYLE member to associate a valid OPID with each MVS console that you want to issue NetView commands.

  Refer to the NetView online help for more information about the AUTOTASK command.

Using MVS Operator Consoles to Issue Commands and Command Lists as MODIFY (F) Commands

From a system console, NetView appears as an MVS subsystem. Any NetView command or command list can be accepted by a NetView subsystem and passed to its associated NetView application if an AUTOTASK command has been entered to associate a multiple console support console with the NetView program.

If a NetView command is issued from a console that has no autotask, an error message is returned from the NetView subsystem, indicating that the console is not authorized to use the NetView subsystem. An AUTOTASK statement in the CNMSTYLE member starts an autotask for AUTO2 and an MVS console. Add a similar statement to the CNMSTYLE member to associate a valid OPID with each MVS console that you want to issue NetView commands.

Refer to the NetView online help for more information about the AUTOTASK command. Refer to the *IBM Tivoli NetView for z/OS Installation: Getting Started* for more information about the CNMSTYLE member.

Multiple Console Support Operator Use of Command Lists

One simple application of NetView automation is to provide the operators with command lists that they can run from a multiple console support console. To do

this, all you need is an autotask associated with the console and a set of command lists to be processed. For example, an operator wanting to bring a set of DASD online that requires specific mount attributes must enter a sequence of commands such as the ones in Figure 183.

```
M 350,VOL=(SL,VOL001),USE=STORAGE
M 351,VOL=(SL,VOL002)
⋮
```

Figure 183. Commands Used to Bring DASD Online

With a large number of volumes to mount, the sequence could be very long. However, you can provide a command list that issues all of the necessary MVS commands from NetView. The operator can then bring the DASD online just by entering the NetView designator character and the name of the command list.

Issuing an MVS Command from a NetView Operator ID

To issue an MVS operator command from any NetView operator ID, enter the command **MVS**, followed by a space, followed by the MVS operator console command, just as you enter it at an MVS operator console. Note that if NetView uses command authorization, it must permit the operator ID to issue the command. Refer to the *IBM Tivoli NetView for z/OS Administration Reference*.

Using EMCS Consoles

With EMCS consoles, you can define NetView consoles by operator name and give them various security and authority classes using MVS, the NetView program, and definitions in a system authorization facility (SAF) product, such as RACF (Resource Access Control Facility).

When you issue an MVS command, and do not already have an EMCS console, NetView attempts to obtain an EMCS console with your operator ID as the console name.

Use the GETCONID command to obtain an EMCS console if a naming conflict exists. There is no defined limit on the number of EMCS consoles.

Use the SETCONID command to assign a console name without actually allocating it.

Appendix E. VTAM Message and Command Processing

This appendix documents information that is diagnostic, modification, and tuning information provided by the NetView program.

Attention: Do not use this diagnosis, modification, and tuning information as a programming interface.

VTAM messages provide information to the VTAM operator. But message volumes can be high, adversely affecting system performance and possibly causing an operator to miss a vital piece of information. To help you control message rates, VTAM offers several message suppression mechanisms. This appendix describes the VTAM message and command flow, and the message suppression mechanisms.

Message and Command Flow in VTAM

VTAM messages can be either solicited or unsolicited. Solicited messages are issued in response to a command such as DISPLAY and are normally returned to the operator who entered the command. Unsolicited messages are issued by VTAM during the course of normal operations to give status information about system components.

When a NetView operator submits commands to VTAM across the program operator interface (POI), the solicited responses are correlated with the NetView operator ID that issued the command. When the NetView primary POI task (PPT) is defined as a VTAM application with AUTH=PPO, unsolicited VTAM messages are directed to NetView across the VTAM POI. If the PPT is not defined as a VTAM application with AUTH=PPO, NetView receives unsolicited VTAM messages across the MVS subsystem interface.

Because VTAM messages do not always get through the operating system to NetView, the message processing facility (MPF) is not always driven by VTAM messages. This means that VTAM messages identified in MPF as messages to be suppressed are suppressed only if the NetView-VTAM POI interface is not available. For that reason, VTAM message suppression should be accomplished with the VTAM message flooding prevention table (see “Message Flooding Prevention Table” on page 533) or the automation table.

Special considerations must be taken into account when two NetView programs exist in one system when defining the POI and service point operations (SPO). See Chapter 32, “Running Multiple NetView Programs Per System,” on page 455 for more information.

Message Flooding Prevention Table

The VTAM message flooding prevention table assists with situations in which a large number of messages are repeated frequently or are issued following an underlying event or condition. The table is a list of messages identified as potential sources of message flooding. Such messages are suppressed if they recur with variable fields unchanged within a certain time span (the default is 30 seconds). The following suppression rules for message flooding prevention are consistent with the VTAM MODIFY SUPP command:

- The message is recorded in the VTAM internal trace table.
- The message is constructed but not transmitted to the operator. It might be routed to other areas (for example, the network log).
- If the first line of a multiline write-to-operator (MLWTO) message group is suppressed, all lines in the group are also suppressed.
- Unformatted system services (USS) messages are not suppressed.

A message resulting from an operator command can be suppressed if the message is a member of the message flooding prevention table and the operator issues the same command within the designated time span.

If the header of an MLWTO message group is suppressed, all messages in the group are also suppressed. This is true even if the information is different from the last occurrence. For more information, refer to the z/OS Communications Server library.

Suppressing VTAM messages can affect AON/SNA automation. Do not code any VTAM message in the message suppression table that is also trapped in the automation table DSITBL01.

VTAM Message Suppression Criteria

One of the first tasks in deciding whether to implement any of the suppression techniques is to measure the rate of unsolicited VTAM messages and determine if that rate is too high. You can count the number of messages issued over a given time and compare that with an established threshold. Because different systems have different characteristics, you must consider those rates relative to your environment. If the rate is excessive, you should look at automatic message suppression. By means of automation tables and command lists, you can monitor the volume of message traffic and automatically perform selective suppression.

Identifying Events with the Automation Table

The primary way to recognize an event is through the NetView automation table, which is searched each time a message arrives. If the search argument (which can be anywhere in the message) is not found, the message is returned for normal processing by NetView; that is, it is displayed on the console. If the search is successful, the message can be held or deleted from the console. The message can also drive a command processor or command list to take further action.

Most VTAM messages are contained in USS tables, the main one being ISTINCNO, and are defined by means of the USSMSG macro. Messages can be modified by that macro, although it is generally preferable to create new tables rather than modify the ones that are supplied by IBM.

Understanding Suppression Levels

The SUPP parameter on the USSMSG macro is used to assign a message class that works in conjunction with the message suppression level to determine whether a message is displayed. The classes are, in increasing order of severity:

SUPP=INFO Informational
SUPP=WARN Warning
SUPP=NORM Normal
SUPP=SER Serious

In addition to these four classes, a message can be defined with SUPP=ALWAYS or SUPP=NEVER, which are independent of the suppression level in force at the time. For a description of all the standard VTAM operator messages and suppression classes, refer to the z/OS Communications Server library.

The message suppression level is set by means of the SUPP parameter on the VTAM START command or the MODIFY SUPP command. In either case, the SUPP parameter is one of the preceding four classes. If any one of these four levels of suppression classes is affixed to a message, that level and those above it are suppressed.

Finally, you can specify SUPP=NOSUP to negate any suppression. In that case, only those messages defined as SUPP=ALWAYS are suppressed.

Identifying Unsuppressable Messages

Messages defined as SUPP=NEVER are not suppressed, regardless of the suppression level set. They are known as unsuppressable messages and include:

- Error messages resulting from an abnormal end of a task
- Messages requiring operator response (suffix A action messages)
- Messages resulting from a DISPLAY or START command

You cannot suppress individual lines of multiline WTO messages. If the header line is suppressed, all lines in the group are suppressed.

To automate message processing, you must first initialize NetView for routing of messages to specific operators for processing. If you use the NetView ASSIGN command, you can route unsolicited messages directly to the specific operator station task (OST) that is to handle the messages. The automation table then runs under that task. If messages are not assigned, they must be routed through the automation table, and processing delays can occur because all the messages are queued to one task.

Appendix F. Detailed NetView Message and Command Flows

This appendix documents information that is diagnostic, modification, and tuning information provided by NetView.

Attention: Do not use this diagnosis, modification, and tuning information as a programming interface.

This appendix contains diagrams and descriptions that show the flow of messages and commands through NetView. The descriptions indicate where each numbered exit (DSIEX01, DSIEX02A, and so on) occurs and how it is processed in relation to commands. Information on the sequence and context of message processing is particularly useful when you automate messages using the automation table, ASSIGN command, and other message processing facilities.

Flow Diagrams

The diagrams in this section illustrate the flow of messages and commands within the NetView product. The numbered tags in the diagrams correspond to the numbered topics in "Flow Descriptions" on page 546.

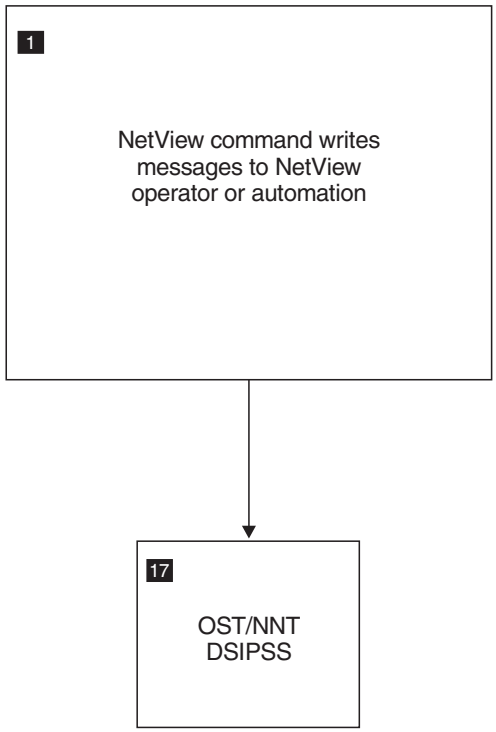


Figure 184. Flow Diagram for NetView Command Entry (VTAM Terminal)

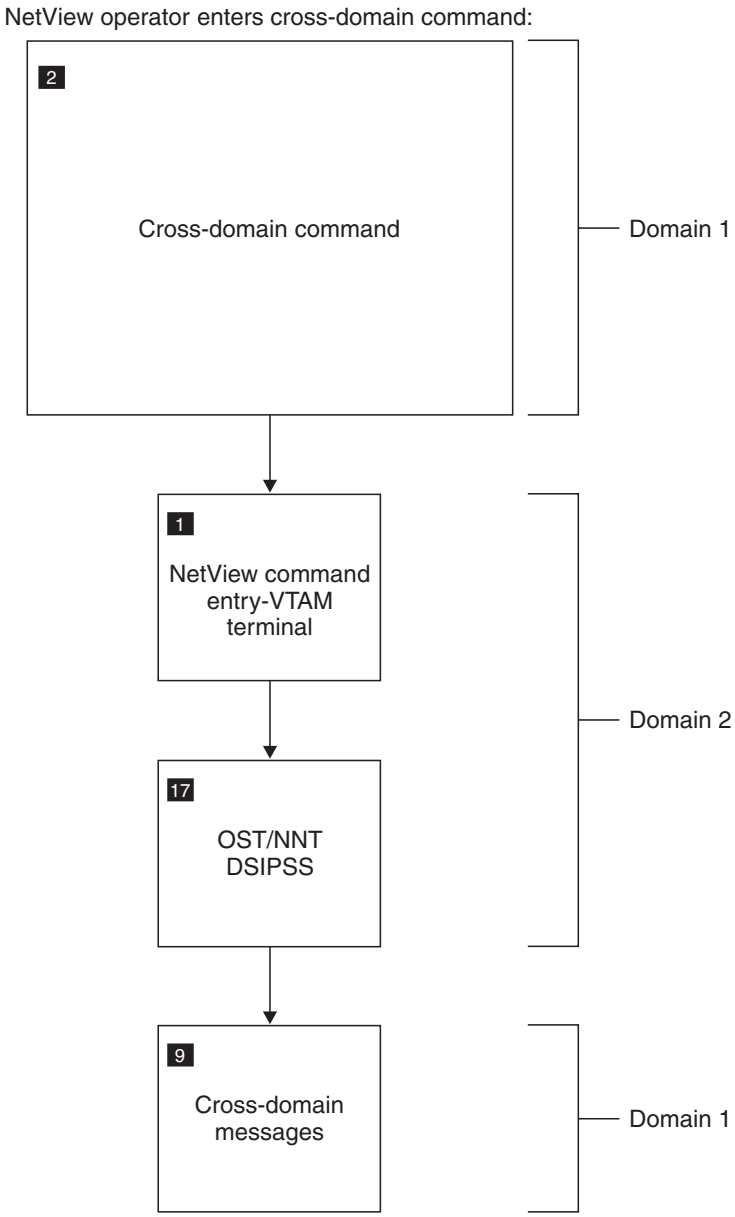


Figure 185. Flow Diagram for Cross-Domain Commands

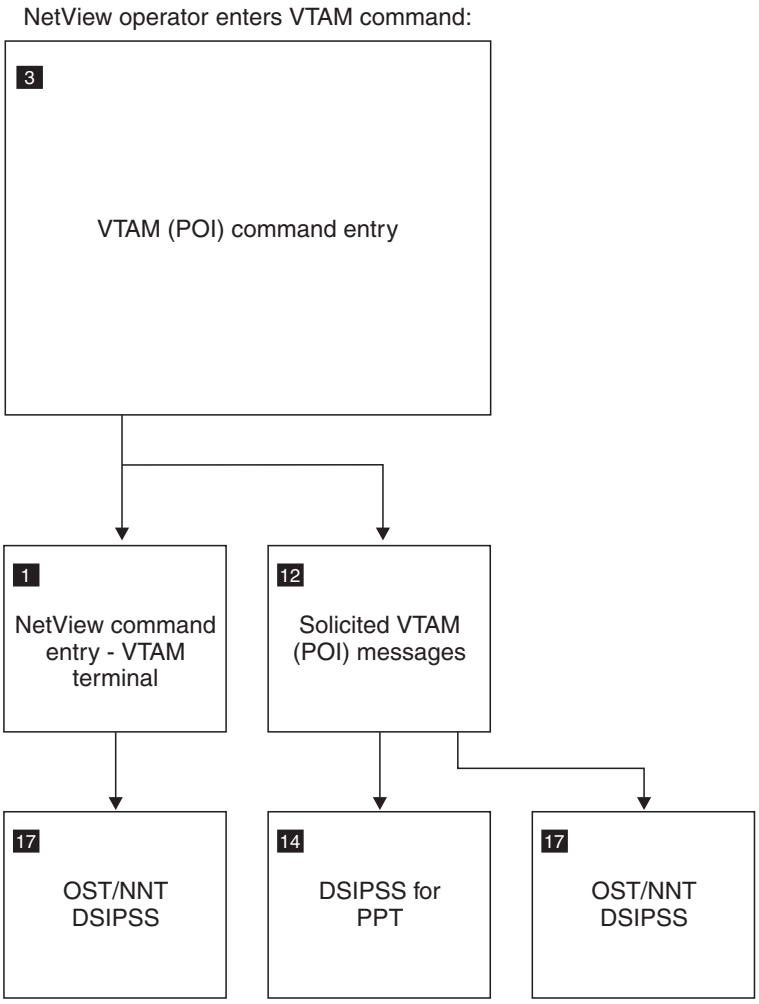


Figure 186. Flow Diagram for VTAM (POI) Command Entry

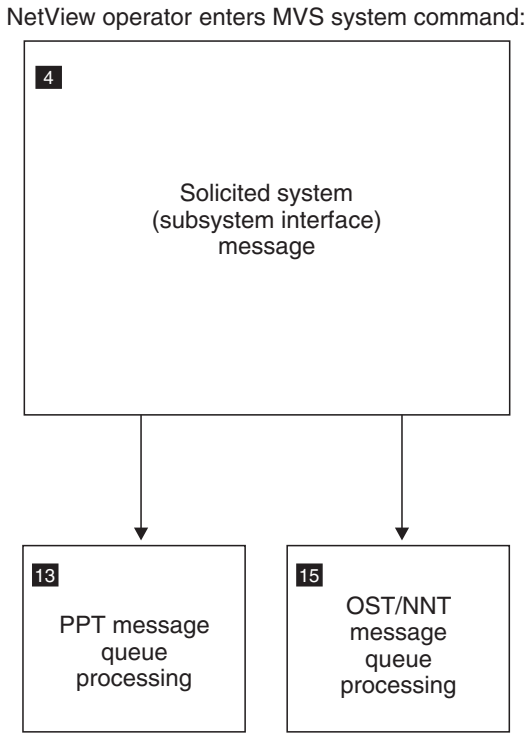


Figure 187. Flow Diagram for Solicited System (Subsystem Interface) Messages

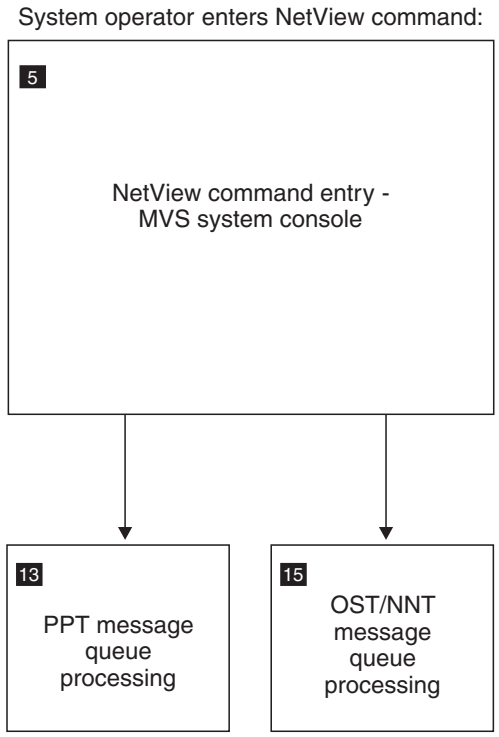


Figure 188. Flow Diagram for NetView Command Entry (MVS)

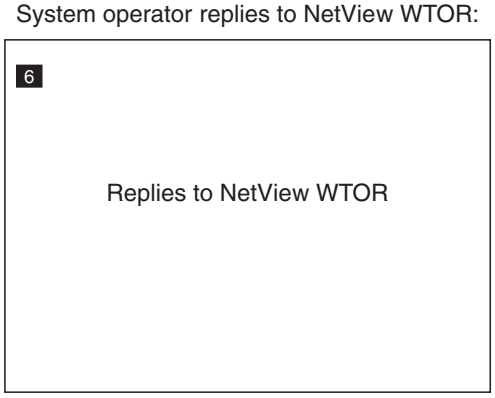


Figure 189. Flow Diagram for Replies to NetView WTOR

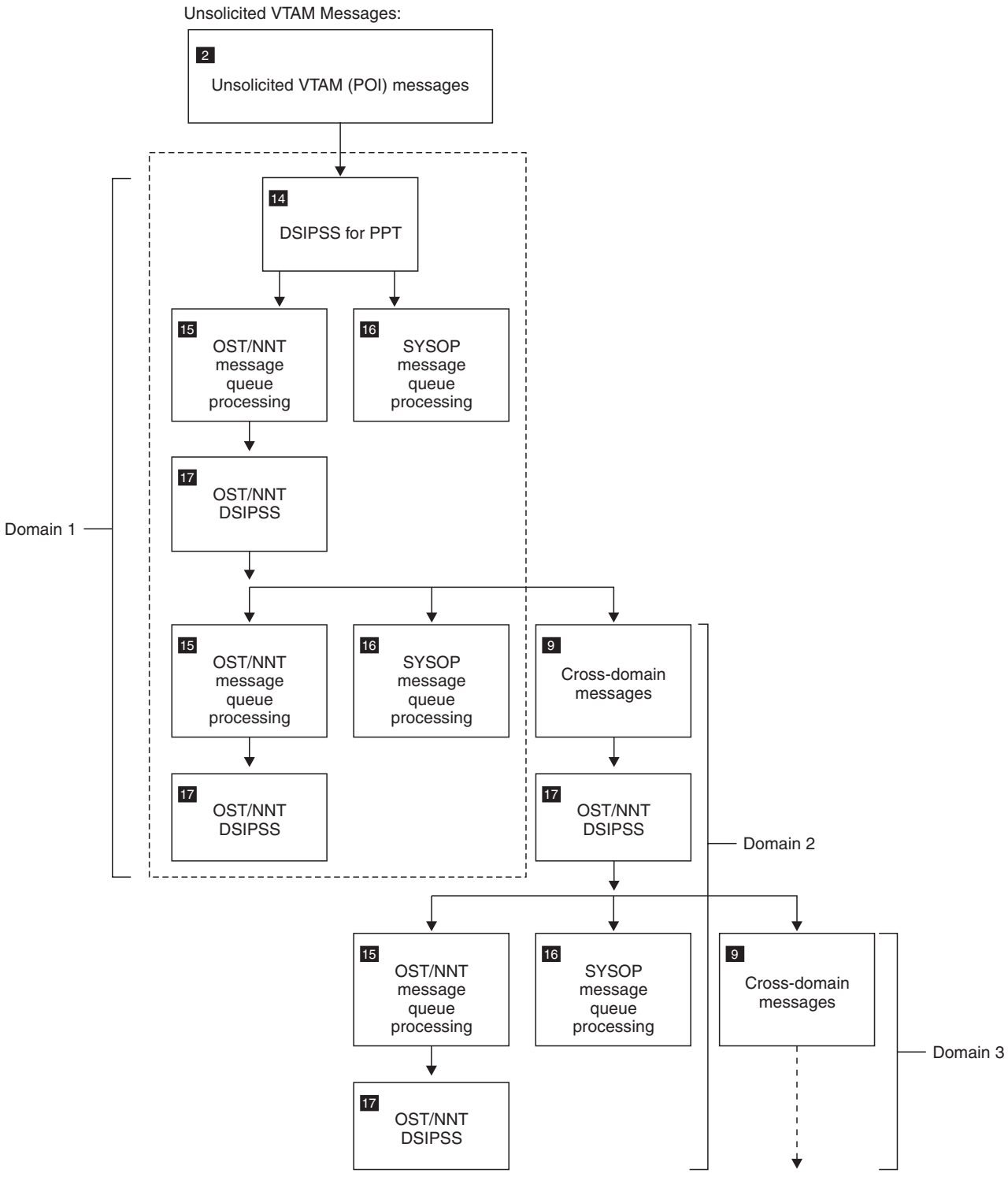


Figure 190. Flow Diagram for Unsolicited VTAM (POI) Messages

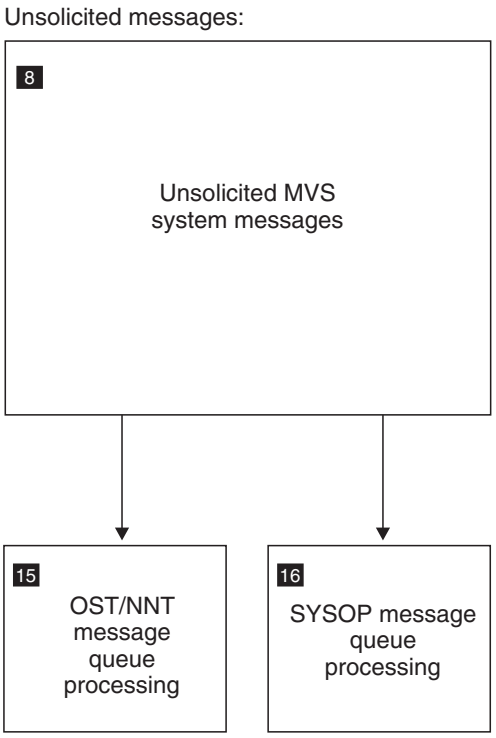


Figure 191. Flow Diagram for Unsolicited System (SSI or MVS Extended Console) Messages (CNMCSSIR)

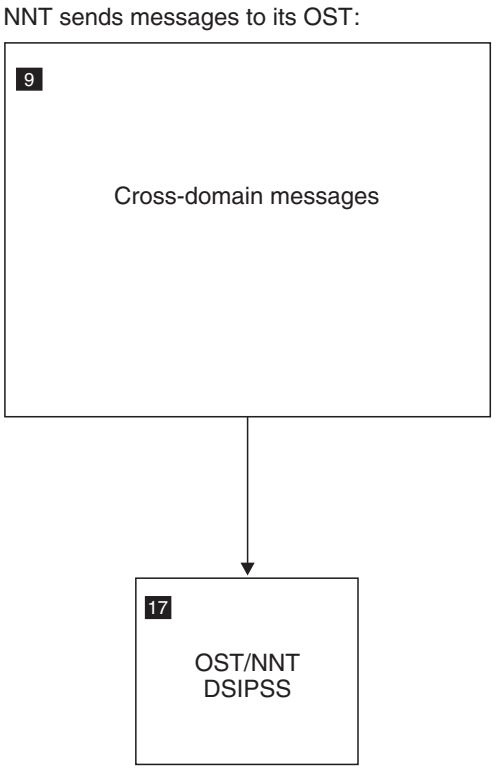


Figure 192. Flow Diagram for Cross-Domain Messages (NNT to OST)

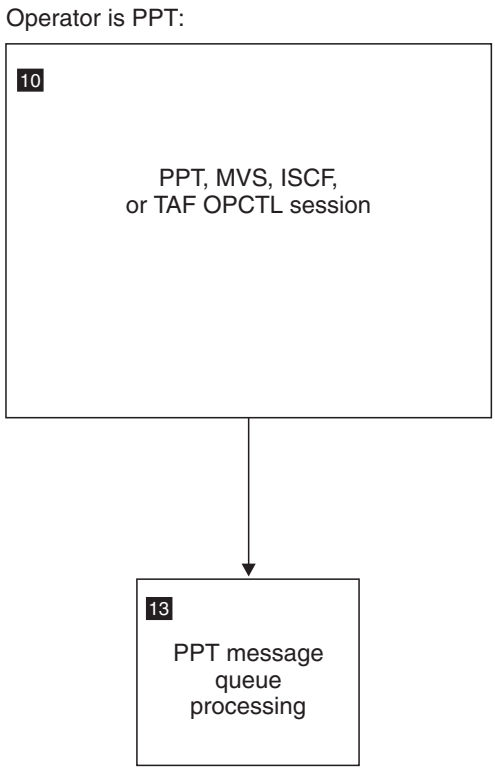


Figure 193. Flow Diagram for Messages (Operator is PPT)

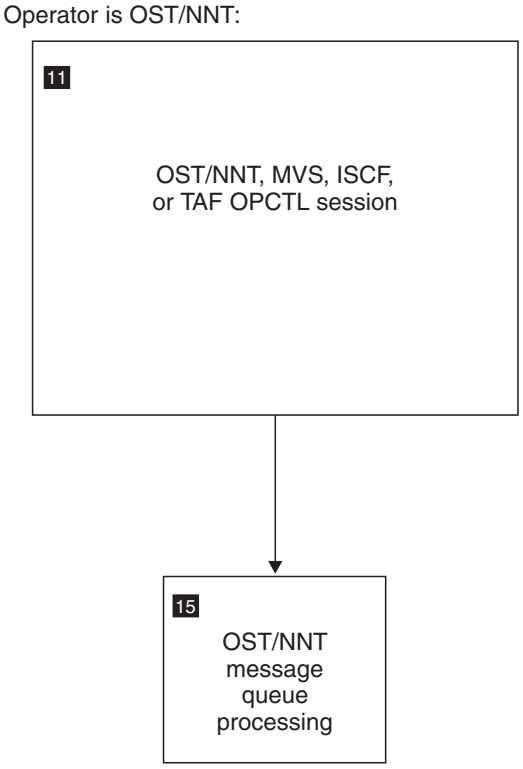


Figure 194. Flow Diagram for Messages (Operator is OST/NNT)

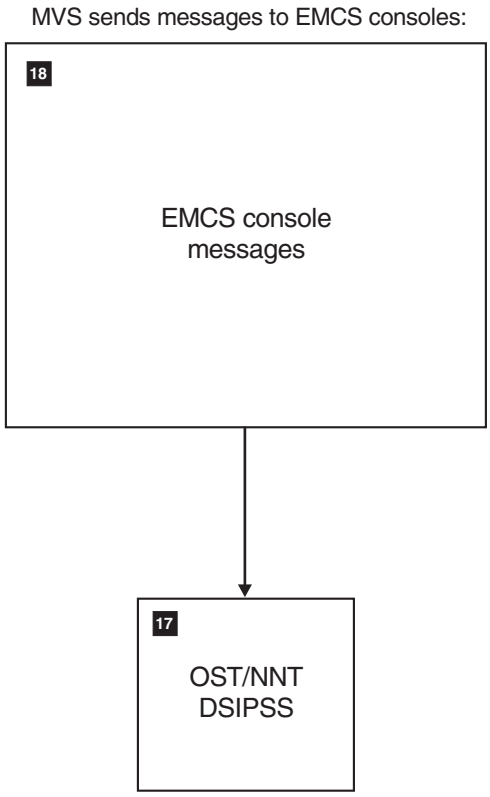


Figure 195. Flow Diagram for Solicited and Unsolicited System MVS Extended Console Messages for OST, NNT, or Autotask

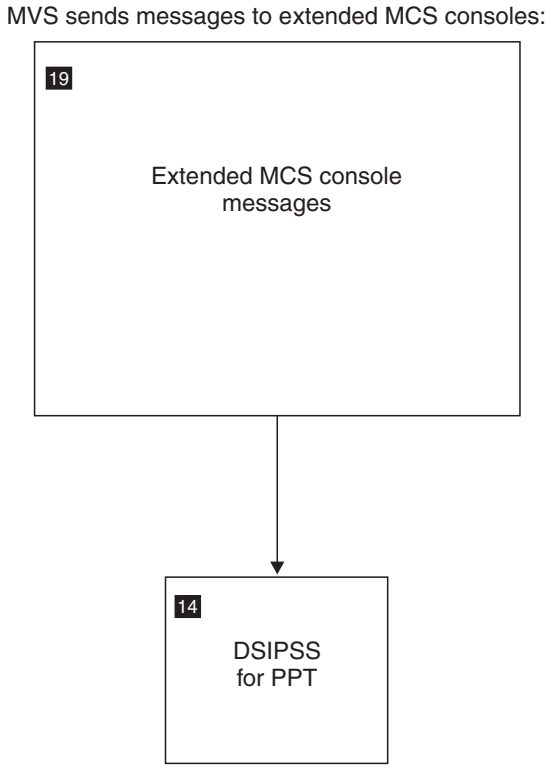


Figure 196. Flow Diagram for Solicited and Unsolicited System MVS Extended Console Messages for PPT

Flow Descriptions

This section describes the flow of messages and commands within the NetView program. Each flow description contains the following information:

Cause	The condition or event that initiates a particular flow
Originating Task	The task in which the condition or event occurred
Process Flow	The sequence of message processing

1. NetView Command Entry (VTAM Terminal)

Cause: A NetView operator enters a line-mode command, or a NetView-NetView task (NNT) receives a command from an OST. The command could be routed by MVS or TAF.

Originating Task: OST or NNT

Process Flow:

1. DSIEX01 is called (TVBINXIT=ON).

   If the exit deletes the command, it is not processed further.
2. The command processor is called and processing continues as follows:
   - Immediate commands are run from the asynchronous input exit and have TVBINXIT=ON.
   - Regular commands are run from the normal task process, with TVBINXIT=OFF.

- Commands can issue DSIPSS. See "17. OST or NNT DSIPSS" on page 555.

2. Cross-Domain Commands (OST to NNT)

Causes:

- NetView ROUTE command (DSIRTP)
- NetView VTAM command (DSIVTP) with implicit or explicit cross-domain routing

Originating Task: OST or NNT

Process Flow:

1. DSIEX07 is called.

   If the exit deletes the command, it is not processed further.
2. The command is sent to the NNT.

   See "1. NetView Command Entry (VTAM Terminal)" on page 546.

3. VTAM (POI) Command Entry

Cause: An operator or command list enters a VTAM command using a NetView command processor whose associated CMDDEF definition specifies DSIVTP.

Originating Task: OST, NNT, or primary program operator interface (POI) task (PPT)

Process Flow:

1. If it is an OST or NNT and is an implicit or explicit cross-domain VTAM command:
   a. DSIEX07 is called. If the exit deletes the command, it is not processed further.
   b. The command is sent to the NNT and runs under the NNT in the other domain, with one level of explicit routing removed, as needed. See "1. NetView Command Entry (VTAM Terminal)" on page 546.
2. Otherwise:
   a. DSIEX05 is called. If the exit deletes the command, it is not processed further.
   b. The command is sent to VTAM.
   c. Messages are returned asynchronously. See "12. Solicited VTAM (POI) Messages" on page 551.

4. Solicited System Messages

Cause: An MVS command is entered from the NetView program, and MVS issues a WTO using the console name. If the console name is not used, the messages are unsolicited and appear unsolicited to NetView.

Note: If an MVS command is issued from a console owned by NetView and the response is marked AUTO(YES) and SUP(YES), the message is automated under the CNMCSSIR task. The message is treated as an unsolicited MVS system message.

Originating Task: OST, NNT, or PPT. NetView ignores any subsystem interface messages from any NetView program if the messages were checked against an automation table, whether the messages were actually automated or not.

Process Flow: DSIEX17 is called after the message is converted into an automation internal function request (AIFR).

DSIEX17 is called for both messages and delete operator message (DOM) commands. The message is sent to the associated task. See "13. PPT Message Queue Processing" on page 552 and "15. OST or NNT Message Queue Processing" on page 553.

5. NetView Command Entry (MVS System Console)

Causes: The system operator enters commands with the NetView subsystem designator, and the following are true:

- The NetView subsystem is active.
- The CNMCSSIR task is active.
- One of the commands shown in Figure 197 was issued in NetView to associate a NetView autotask with this system console.

```
AUTOTASK OPID=operid,CONSOLE=name
```

Figure 197. Commands to Associate an Autotask with a System Console

Originating Task: CNMCSSIR

Process Flow: The command is sent with HDRMTYPE=HDRTYPET to an OST, NNT, or PPT.

See "13. PPT Message Queue Processing" on page 552 and "15. OST or NNT Message Queue Processing" on page 553.

6. Replies to NetView WTOR

Cause: The operator enters a system reply command for a NetView WTOR (message numbers DSI802A and DSI803A).

Originating Task: DSIWTOMT

Process Flow:

1. The operator replies to the WTOR.
2. The NetView main task calls DSIEX10.

   If the exit deletes the input, it is not processed further.
3. The main task processes CLOSE, REPLY, or MSG commands.
4. REPLY commands (commands with IFRCODCR on) go on to the PPT for processing.

   The NetView PPT calls DSIEX03. If the exit deletes the command, it is not processed further. Otherwise, the command runs under the PPT.

7. Unsolicited VTAM (POI) Messages

Cause: VTAM sends unsolicited messages to the NetView PPT, the application that has AUTH=(PPO).

Originating Task: PPT

Process Flow:

1. If the VTAM MSGMOD option is active:

- The PPT logs the message with MSGMOD (for diagnostics).

  The DSIEX04 exit is called during log processing. DSIEX04 specifies whether the message is sent to the hardcopy, network, or system logs, or is deleted or replaced.
- The PPT removes the MSGMOD identifier to make the message consistent with automation.
- The PPT continues as if MSGMOD were not active.

2. If the message is used by the status monitor to update network status, it is processed by the status monitor.
3. The PPT calls DSIEX11. If the exit deletes the message, it is not processed further.
4. If the message is a PPOLOG message, PPT logs the message.

   DSIEX04 is called during log processing. By placing a return code in register 15, DSIEX04 determines whether the message is sent to the hardcopy, network, or system logs.

   No other processing is done for a PPOLOG message.
5. Otherwise, an authorized receiver is the destination for PPT messages. PPT issues DSIPSS TYPE=OUTPUT to send messages to an authorized receiver. See "14. DSIPSS for PPT or NetView Authorized-Receiver Messages" on page 552.

8. Unsolicited MVS System Messages

Cause: MVS WTOs are routed through the NetView subsystem to the CNMCSSIR task.

Originating Task: Any task in any address space that issues WTO for a console that is not assigned to a NetView operator.

Process Flow: DSIEX17 is called after the message is converted into an automation internal function request (AIFR). DSIEX17 is called for both messages and delete operator message (DOM) commands. The processing is parallel to OST or NNT DSIPSS and PPT DSIPSS processing. Compared to "14. DSIPSS for PPT or NetView Authorized-Receiver Messages" on page 552, CNMCSSIR does not search for the authorized receiver. If it did, all system messages would be routed indiscriminately.

1. If ASSIGN PRI was specified for the message, the message is sent to the specified operator.

   See "15. OST or NNT Message Queue Processing" on page 553 and "16. NetView Console Output or SYSOP Message Queue Processing" on page 554.

   ASSIGN SEC messages are never processed by the automation table in this NetView domain. When sent to another NetView domain, they are eligible for processing by DSIEX02A, the automation table, and DSIEX16.
2. For all other messages:
   a. DSIEX02A is called.

      If the message is deleted by the exit, no further processing occurs. DSIEX02A is called only once for each unique message in a NetView domain.

      After that, any copies of the message made by the ASSIGN command or the automation table do not result in a call to DSIEX02A in this NetView domain. Sending a copy of the message to another NetView domain can result in a call to DSIEX02A in that domain.

b. The automation table is checked. Command and display actions can be selected by the table.

   The automation table is called only once for each unique message in a NetView domain. After that, any copies of the message made by the ASSIGN command or the automation table do not result in a call to the automation table in this NetView domain.

   Sending a copy of the message to another NetView domain can result in a call to the automation table in that domain.

c. DSIEX16 is called at this point.

d. The actions indicated up through DSIEX16 are performed. Buffer structure determines the actions that occur.

   DSIEX16 is called only once for each unique message in a NetView domain. After that, any copies of the message made by the ASSIGN command or the automation table do not result in a call to DSIEX16 in this NetView domain. Sending a copy of the message to another NetView domain can result in a call to DSIEX16 in that domain.

   Note: Compared to "14. DSIPSS for PPT or NetView Authorized-Receiver Messages" on page 552, CNMCSSIR discards the message if no action is specified up to this point. In that case, the message is not displayed and its Canzog entry is not marked as being germane.

   See also "15. OST or NNT Message Queue Processing" on page 553 for automation table routed messages.

   Automation table entries can be specified without a ROUTE keyword. In this case, the CNMCSSIR task marks the Canzlog entry as germane, but will not display the message anywhere. If a command was specified, the command will be routed to the Primary Autotask.

   Having an autotask start CNMCSSIR is a good way to allow an autotask to monitor the status of CNMCSSIR. Doing so also provides the default destination for automation processing (for when the ROUTE option is omitted from the automation statements).

9. Cross-Domain Messages and Commands (NNT to OST)

Cause: An NNT sends all messages it receives to the OST that started the OST-NNT session (using the START DOMAIN command). A command can be sent from an NNT to its associated OST by a DSIPSS TYPE=OUTPUT of a HDRTYPEX buffer. Refer to the DSIPSS macro in *IBM Tivoli NetView for z/OS Programming: Assembler* for more information.

Originating Task: NNT

On the originating NNT, any information delivered to EMCS consoles is not sent to the OST.

Process Flow:

At the receiving OST:

1. If the received buffer is a command and has a HDRMTYPE of HDRTYPEI and an IFRCODE of IFRCODAI (an AIFR) and if the buffer pointed to by IFRAUTBA has a HDRMTYPE of HDRTYPEX, bits IFRAUPHI and IFRAUPLO are used to set the priority at which the received command is requeued to the OST message queues.

If neither IFRAUPHI nor IFRAUPLO is on, the defaults of HIGH (for TYPE=IMMED and TYPE=BOTH commands) or LOW (for other types of commands) are used. The value in field IFRAUTBA is moved to the IFRAUCMB field and the IFRAUCMD bit (indicating a command is pointed to by IFRAUCMB) is turned on. All of the automation flags in the received buffer are reset except BEEP, DISPLAY, and HOLD.

2. If the buffer is a message (that is, if IFRAUTBA is not a HDRTYPEX buffer), the priority is set to HIGH.
3. The AIFR is queued to the OST message queue corresponding to the priority previously determined, with the exception of some TYPE=IMMED messages related to cross-domain logon.

   These messages are not queued. They are displayed through the DSIPSS macro immediately upon receipt.

See "15. OST or NNT Message Queue Processing" on page 553 for further processing.

10. PPT as the MVS or TAF OPCTL Operator

Causes:
- An MVS command was issued by the PPT.
- A BGNSESS OPCTL command was run on the PPT.

Originating Task: PPT

Process Flow: All solicited or unsolicited messages are received from MVS or TAF OPCTL sessions on the PPT's message queue. See "13. PPT Message Queue Processing" on page 552.

11. OST or NNT as MVS or TAF OPCTL Operator

Causes:
- An MVS command was issued by an OST or NNT.
- A BGNSESS OPCTL command was run on the OST or NNT.

Originating Task: OST or NNT

Process Flow: All solicited or unsolicited messages are received from MVS or TAF OPCTL sessions on the OST or NNT's message queue. See "15. OST or NNT Message Queue Processing" on page 553.

12. Solicited VTAM (POI) Messages

Cause: The CMDDEF definition of a VTAM command specifies module DSIVTP.

Originating Task: OST, NNT, or PPT

Process Flow:

1. If the VTAM MSGMOD option is active:
   - The message is logged with MSGMOD (for diagnostics). DSIEX04 is called during log processing. By placing a return code in register 15, DSIEX04 determines whether the message is sent to the hardcopy, network, or system logs.
   - The MSGMOD identifier is removed to make the message consistent with automation.
   - Processing continues as if MSGMOD were not active.

2. If the message is used by the status monitor to update network status, it is processed by the status monitor.
3. DSIEX06 is called. If the message is deleted, no further processing takes place.
4. DSIPSS TYPE=OUTPUT is issued to send the message.

   See "14. DSIPSS for PPT or NetView Authorized-Receiver Messages" and "17. OST or NNT DSIPSS" on page 555.

13. PPT Message Queue Processing

Causes:

- Messages from the MSG command
- Automation-table directed messages and commands
- General cross-task messages through DSIMQS macro
- Terminal access facility (TAF) operator control

Originating Task: Any

Process Flow:

1. If HDRMTYPE=HDRTYPEI or HDRMTYPE=HDRTYPET (message is an internal function request), the requested functions are performed.
   - If HDRMTYPE=HDRTYPEI and IFRCODE=IFRCODUS (user internal function request), the PPT calls DSIEX13 to process the message buffer and frees the buffer (with DSIFRE) upon return.
   - Otherwise, if HDRMTYPE=HDRTYPET or HDRMTYPE=HDRTYPEI and IFRCODE=IFRCODCR, the PPT calls DSIEX03 to process command input. If DSIEX03 deletes the command, processing of the command is ended. Otherwise, the command processor runs.
2. Otherwise, the PPT does message processing:
   - The message is processed by DSIPSS TYPE=OUTPUT. See "14. DSIPSS for PPT or NetView Authorized-Receiver Messages."

14. DSIPSS for PPT or NetView Authorized-Receiver Messages

Causes:

- DSIPSS issued in PPT
- Message sent to PPT
- DSIMQS to authorized receiver
- Unsolicited VTAM (POI) messages
- Messages received from a TAF OPCTL session started by the PPT using the BGNSESS command

Originating Task: Any

Process Flow:

1. If ASSIGN PRI was specified for the message, the message is sent to the operator specified by the ASSIGN command.

   See "15. OST or NNT Message Queue Processing" on page 553 and "16. NetView Console Output or SYSOP Message Queue Processing" on page 554.

   The following types of messages cannot be assigned in this step:
   - Messages previously routed using the ASSIGN command
   - WTOs from an NetView address space in this system

ASSIGN SEC messages are never processed by the automation table in this NetView domain. When sent to another NetView domain, they are eligible for processing by DSIEX02A, the automation table, and DSIEX16.

2. Otherwise, the message is sent to one of the following authorized receivers, if active:
   - An operator logged on to a POS terminal

     If more than one POS terminal is defined, the first one defined has first priority.
   - An operator that is not defined as a POS

     If more than one such operator is defined, the first one defined has first priority.
   - A cross-domain operator

     If more than one cross-domain operator is defined, the first one defined has first priority.
   - An autotask operator

     If more than one autotask was started, the first one started has first priority. Use the ASSIGN command if an autotask is to be the receiver of unsolicited messages.
3. If neither 1 nor 2 preceding is true:
   a. DSIEX02A is called. If the message is deleted by the exit, no further processing occurs.
   b. The automation table is checked. Command and display actions can be selected by the table.
   c. DSIEX16 is called with the results to this point, even if the table deletes the message.
   d. DSIEX02, the automation table, and DSIEX16 are called only once for each unique message in a NetView domain.

      After that, any copies of the message made by the ASSIGN command or the automation table do not result in a call to DSIEX02, the automation table, or DSIEX16 in this NetView domain. When a message is sent to another NetView domain, a call can result in that domain.
   e. PPT logs the message if logging was not suppressed.

      If the message is one that the status monitor uses to update network status, it is processed by the status monitor.
   f. The message is sent to the system console (SYSOP) if no other action was indicated or the actions indicated up through DSIEX16 are carried out.

      Messages originating from the subsystem interface in NetView are not written to any system console. See also "15. OST or NNT Message Queue Processing" for automation table routed messages.

15. OST or NNT Message Queue Processing

Causes:

- Messages from the MSG command
- Authorized-receiver routed messages
- ASSIGN PRI, SEC, COPY messages
- Automation-table directed messages and commands
- General cross-task messages through DSIMQS macro
- MVS messages if this operator is an MVS operator
- Messages from terminal access facility (TAF) operator control (console)

Originating Task: Any

Process Flow:

1. If HDRMTYPE=HDRTYPEI or HDRMTYPE=HDRTYPET (message is an internal function request), the requested functions are performed.
   - If HDRMTYPE=HDRTYPEI and IFRCODE=IFRCODUS (user internal function request), DSIEX13 is called to process the message buffer.
   - If HDRMTYPE=HDRTYPET or HDRMTYPE=HDRTYPEI and IFRCODE=IFRCODCR, the OST or NNT calls DSIEX03 to process command input. If DSIEX03 deletes the command, processing of the command is ended. Otherwise, the command processor runs.
2. Otherwise, the OST or NNT does one of the following:
   - If HDRMTYPE=HDRTYPEM, the OST or NNT calls DSIEX13 to process the message buffer and frees the buffer (with DSIFRE) upon return.
   - Otherwise, the OST or NNT processes the message with DSIPSS TYPE=OUTPUT. See "17. OST or NNT DSIPSS" on page 555.

16. NetView Console Output or SYSOP Message Queue Processing

Causes:

- NetView issues DSIWCS to send the message to the system console.
- The NetView authorized receiver routes the message to the system console if no other destination is specified or available.
- The NetView PPT routes the message to the system console if no other destination is specified or available.
- Messages are queued to SYSOP.
- ASSIGN PRI, SEC, COPY messages are routed to SYSOP.

  Messages originating from the subsystem interface which are processed by NetView can be redisplayed on a system console. Descriptor code 13 is turned on for these messages to prevent a potential looping condition.

Originating Task: Any

Process Flow:

1. DSIWCS writes the buffer to the logs, and both of the following occur:
   - DSIWLS macro processing calls DSIEX04 if DSIEX02A was not called. By placing a return code in register 4, DSIEX04 determines whether the message is sent to the hardcopy, CanzLog log, network, or system logs.
   - DSIWLS writes the message to the logs according to the DEFAULTS and OVERRIDE commands.

   Note: Messages originating from the subsystem interface in this NetView program are not written to the system log.
2. DSIWCS macro processing calls DSIEX09 if DSIEX02A was not called. If DSIEX09 deletes the message, it is not written to the console.
3. DSIWCS writes the message to the system console.

   Note: ASSIGN PRI=SYSOP bypasses DSIPSS processing, and such messages are not subject to automation.

17. OST or NNT DSIPSS

Cause: DSIPSS

Originating Task: OST, NNT

Process Flow:

1. DSIEX02A is called. If the exit deletes the message, the processing ends.
2. &WAIT and WAIT search is called.

   If the message is suppressed by &WAIT and WAIT processing, it is marked with force flags to not be displayed or logged. However, processing continues to allow exit DSIEX16 to account for such messages.
3. Automation table processing begins.

   Table actions are reflected in the buffer structure given to DSIEX16.
4. DSIEX16 is called.
5. DSIEX16, DSIEX02A, and the automation table are called only once for each unique message in a NetView domain.

   After that, any copies of the message made by the ASSIGN command or the automation table do not result in a call to DSIEX16 in this NetView domain. When a message is sent to another NetView domain, it can result in a call to DSIEX16 in that domain.
6. Logging, display, routing, and command actions are processed as specified in the automation internal function request (AIFR) buffer in combination with the current DEFAULTS and OVERRIDE command settings.

   Buffer structure determines the actions that occur.

   See also "15. OST or NNT Message Queue Processing" on page 553 for automation-table routed messages.
7. If the message is displayed, the ASSIGN COPY service is performed.

   ASSIGN COPY messages are never processed by the automation table in this NetView domain. When sent to another NetView domain they are eligible for processing by DSIEX02A, the automation table, and DSIEX16.

   Messages are sent to the OST or NNT message queue. See "15. OST or NNT Message Queue Processing" on page 553 and "16. NetView Console Output or SYSOP Message Queue Processing" on page 554.
8. If this is an OST, the message is displayed to the NetView terminal (VTAM) or to the system console (AUTOTASK OPID=*name*). Messages originating from the subsystem interface in NetView are not written to any system console.
9. If this is an NNT, the message is sent cross-domain to the OST. See "9. Cross-Domain Messages and Commands (NNT to OST)" on page 550. When received in the next domain, DSIEX02A, automation table, and DSIEX16 processing are permitted, even for ASSIGN SEC and ASSIGN COPY messages.

18 Solicited and Unsolicited System MVS Extended Console Messages for an OST, NNT, or Autotask

Cause: NetView is configured to use EMCS consoles for MVS/ESA system messages. The OST, NNT, or autotask is configured to receive EMCS console messages.

Originating Task: OST, NNT, or autotask

Process Flow:

1. DSIEX17 is called for both messages and DOMs. If the message or DOM is deleted, no further processing occurs.
2. DSIPSS is issued for the message or DOM. See "17. OST or NNT DSIPSS" on page 555.

MVS system messages that are received on EMCS consoles in use by NetView tasks (except the CNMCSSIR task) are considered solicited by NetView. Therefore, these messages are not subject to the ASSIGN command PRI and SEC processing.

MVS/ESA Version 4 Release 3 and subsequent releases support command response suppression. If an MVS command is issued from a NetView console and the response is marked AUTO(YES) and SUP(YES), the message is automated under the CNMCSSIR task. The message is treated as an unsolicited MVS system message.

19 Solicited and Unsolicited System MVS Extended Console Messages for the PPT

Cause: NetView is configured to use EMCS consoles for MVS/ESA system messages. The PPT is configured to receive EMCS console messages.

Originating Task: PPT

Process Flow:

1. DSIEX17 is called for both messages and DOMs.

   If the message or DOM is deleted, no further processing occurs.
2. DSIPSS is issued for the message or DOM.

   See "14. DSIPSS for PPT or NetView Authorized-Receiver Messages" on page 552.

MVS system messages that are received on an EMCS console in use by the PPT are considered solicited by NetView. However, unlike other NetView tasks that receive solicited messages, the PPT enables ASSIGN command PRI and SEC processing.

Appendix G. NetView Message Type (HDRMTYPE) Descriptions

This appendix documents information that is diagnostic, modification, and tuning information provided by the NetView program.

Attention: Do not use this diagnosis, modification, and tuning information as a programming interface.

This appendix lists the NetView message types, which are arranged in alphabetical order. Message types apply to commands, messages, and MSUs. To examine a message type in the automation table, use the HDRMTYPE keyword. Message type is stored in the HDRMTYPE field of the BUFHDR control block.

HDRTYPAC (A)
Is not used in NetView for MVS V1R2 and later releases. This message type is replaced by the automation IFR (HDRMTYPE=HDRTYPEI, IFRCODE=IFRCODAI). You can receive this message type during a cross-domain session with an earlier release of NetView.

HDRTYPDT (D)
Indicates a non-message data type.

HDRTYPEA (T)
Is not used in NetView for MVS V1R2 and later releases. Indicates a solicited message from TCAM in the network communications control facility (NCCF). You might receive this message type on a cross-domain session with a TCAM NCCF.

HDRTYPEB (?)
Indicates a command or command list buffer that has display and logging suppressed. Used to suppress display and logging of commands entered with a suppression character as defined in the CNMSTYLE member. Information about the CNMSTYLE member can be found in *IBM Tivoli NetView for z/OS Installation: Getting Started*. HDRTYPEB is also used to suppress display and logging of command list statements that are preceded by this same suppression character.

HDRTYPEC (C)
Indicates a command or message from a command list. Becomes HDRTYPEB for suppressed command list statements.

HDRTYPED (!)
Indicates a message from an immediate command processor. Usually sent to the screen using DSIPSS TYPE=IMMED. When this type of message is displayed in the immediate message area on the screen, the HDRMTYPE and DOMAIN name are not displayed. When received cross-domain, this type of message is in the normal output area, along with its domain name and type prefix. DSIPSS TYPE=IMMED does not enforce or set HDRTYPED.

HDRTYPEE (E)
Indicates a message from the operating system. This type is not used for title-line mode multiline write-to-operator (MLWTO), system action, or WTOR messages. See also HDRTYPEK and HDRTYPEY for other forms of operating system messages.

HDRTYPEF (F)
Indicates a VSAM record. Not displayed on the operator's screen. Used within the data services task (DST).

HDRTYPEG (G)
Indicates a CNMI record. Not displayed on the operator's screen. Used within the DST.

HDRTYPEI (I)
Indicates an internal function request. This buffer is a formatted interface within and between tasks. The IFR contains a function number (IFRCODE) that determines the format and function of the buffer. For more information, refer to *IBM Tivoli NetView for z/OS Programming: Assembler*.

HDRTYPEJ (')
Indicates a title-line multiline write-to-operator (MLWTO) message originating from NetView itself. These buffers must be in a sequence and include a description of control, label, data, and end designators. NetView for MVS V1R2 and later releases treat these sequences of buffers as a single message for presentation and automation.

HDRTYPEK (")
Has the same meaning as HDRTYPEJ, but for messages originating in IBM routines that are not supplied with the NetView program.

HDRTYPEL (=)
Has the same meaning as HDRTYPEJ, but for messages originating in non-Tivoli routines.

HDRTYPEM (M)
Indicates a message from the NetView message command processor.

HDRTYPEN (-)
Indicates a regular single-buffer message from NetView.

HDRTYPEP (P)
Indicates a message from the PPI.

HDRTYPEQ (Q)
Indicates a message from the VTAM POI that is a single-buffer unsolicited message. See also HDRTYPEV, HDRTYPEY, and HDRTYPEK for other VTAM POI messages. This message type is not set for messages from VTAM received on the operating system interface.

HDRTYPER (R)
Indicates that an operator entered the VTAM REPLY command in response to NetView WTOR number DSI802A. This message type is logged but does not appear on NetView consoles.

HDRTYPES (S)
Is used in some installation-exit interfaces to indicate a swapped buffer.

HDRTYPET (*)
Indicates a command issued to NetView from a NetView terminal. This message type indicates that the buffer is a command rather than a message.

HDRMTYPE=HDRTYPEI with IFRCODE=IFRCODCR is similar in that the buffer represents a command to be processed. Notice that IFRCODCR generally implies an internally formatted command, such as between operator station tasks (OSTs) and DSTs. HDRTYPET generally implies a

command buffer as if an operator had typed the command. IFRCODCR buffers can contain non-printable data. HDRTYPET buffers should contain no non-printable text.

HDRTYPEU (U)
Is reserved for non-Tivoli users. Cannot be used for action messages, WTOR, or title-line (MLWTO) messages.

HDRTYPEV (HEX('40'))
Indicates a message from the VTAM POI that is a single-buffer solicited message. See also HDRTYPEQ, HDRTYPEY, and HDRTYPEK for other VTAM POI messages. This message type is not set for messages from VTAM received on the operating system interface.

Note: This message type is the value X'40', a character space.

HDRTYPEW (+)
Indicates a non-NetView, Tivoli-written single-line message. This message type is similar to HDRTYPEN and HDRTYPEU.

HDRTYPEX (X)
Indicates a cross-domain (NNT to OST) command. Allows reverse-direction commands, because commands are normally routed from the OST to the NNT, for example, with the ROUTE command.

Code running in an NNT can issue DSIPSS TYPE=OUTPUT for an HDRTYPEX buffer, and the corresponding command is processed in the OST that started the session with that NNT. This is useful for sending non-formatted (hexadecimal) data from the NNT to an OST for full-screen or other formatting. The data is limited to 256 bytes and not displayed on the operator's screen.

HDRTYPEY (>)
Indicates a single-buffer action or WTOR. For NetView for MVS V1R2 and later releases, it can be a message from the operating system interface as well as from the VTAM POI. For NetView for MVS V1R2 and later releases, these messages remain on the NetView command facility screen until an action is taken or the reply is entered. The operator can delete these messages by overstriking the greater-than (>) character and pressing ENTER. The message disappears the next time the screen wraps over the text. Installation exits can set this message type to force a message to be held.

When the HDRTYPEY flag is set and the IFRAUWQE flag is not set, NetView looks for a 3-character reply ID immediately preceding the message number in the message text. If the reply ID exists, the message is a VTAM WTOR. Otherwise, the message is treated as a held message (if IFRAUWQE is zero). If IFRAUWQE is set to 1, the IFRAUWQD data is checked to see if the work queue element (WQE) data indicates a WTOR or action message. If a WTOR is indicated, a reply ID (consisting of 2–4 characters) immediately precedes the message ID. If a reply ID exists, it is delimited from the message ID by one space.

HDRTYPEZ (Z)
Is similar to HDRTYPEN, but specifically indicates a message from a data services task (DST).

HDRTYPE$ ($)
Indicates a message that contains data that cannot be printed.

HDRTYPE1 (V)
Is similar to HDRTYPEV, but indicates a PPOLOG message.

HDRTYPE2 (Y)
Is similar to HDRTYPEY, but indicates a PPOLOG message.

HDRTYPFB (HEX'FB')
Indicates a message to flush the buffer.

HDRTYPLS (s)
Indicates a user-substituted command.

HDRTYPLT (L)
Indicates a trace record, not a message buffer. This message type is used exclusively for NetView Internal Trace and must not be used in any message buffer.

HDRTYPOR (HEX '4F')
Indicates a pipeline-generated message.

HDRTYPQC (HEX'50')
Indicates that a command was specified with a double suppression command. Both the command echo at the operator console and any synchronous messages output as a result of the command are suppressed. Asynchronous messages are not suppressed.

HDRTYPWT (W)
Indicates a message that matched a WAIT condition and was displayed. The W appears in the message type field on the screen and in the logs but is not in the HDRMTYPE field in the buffer. The HDRMTYPE field in the buffer contains the original message type.

HDRTYP10 (HEX'10')
Indicates a management services unit (MSU) buffer. The MSU buffer might have an associated MSU HIER buffer.

HDRTYP11 (HEX'0B')
Indicates a remote data transfer message.

Appendix H. MVS Command Management (Deprecated)

The MVS Command Management function is deprecated and is replaced by the MVS Command Revision function. For additional information, see Chapter 14, "The Command Revision Table," on page 135. The MVS Command Management function is only supported for migration purposes. For information on migrating to the MVS Command Revision function, see the *IBM Tivoli NetView for z/OS Installation: Migration Guide*.

With MVS Command Management you can examine, modify, or reject most MVS commands. You can specifically include or exclude commands from processing by command or by console names.

After MVS command management is activated, all MVS commands are passed to the NetView MVS command exit. Most MVS commands are sent to the NetView program for processing unless they are not included or specifically excluded. In the NetView program, the CNMEMCXY REXX command list is called with the MVS command under the DSIMCAOP autotask. You can add logic to this command list to examine, modify, or reject MVS commands. If an MVS command is not rejected, it is returned to MVS for processing. RACF checking is performed after the command is processed by the NetView MVS command exit.

Figure 198 on page 562 shows the logic flow of MVS command management. To enable this command management requires changes to the MVS and NetView environments.

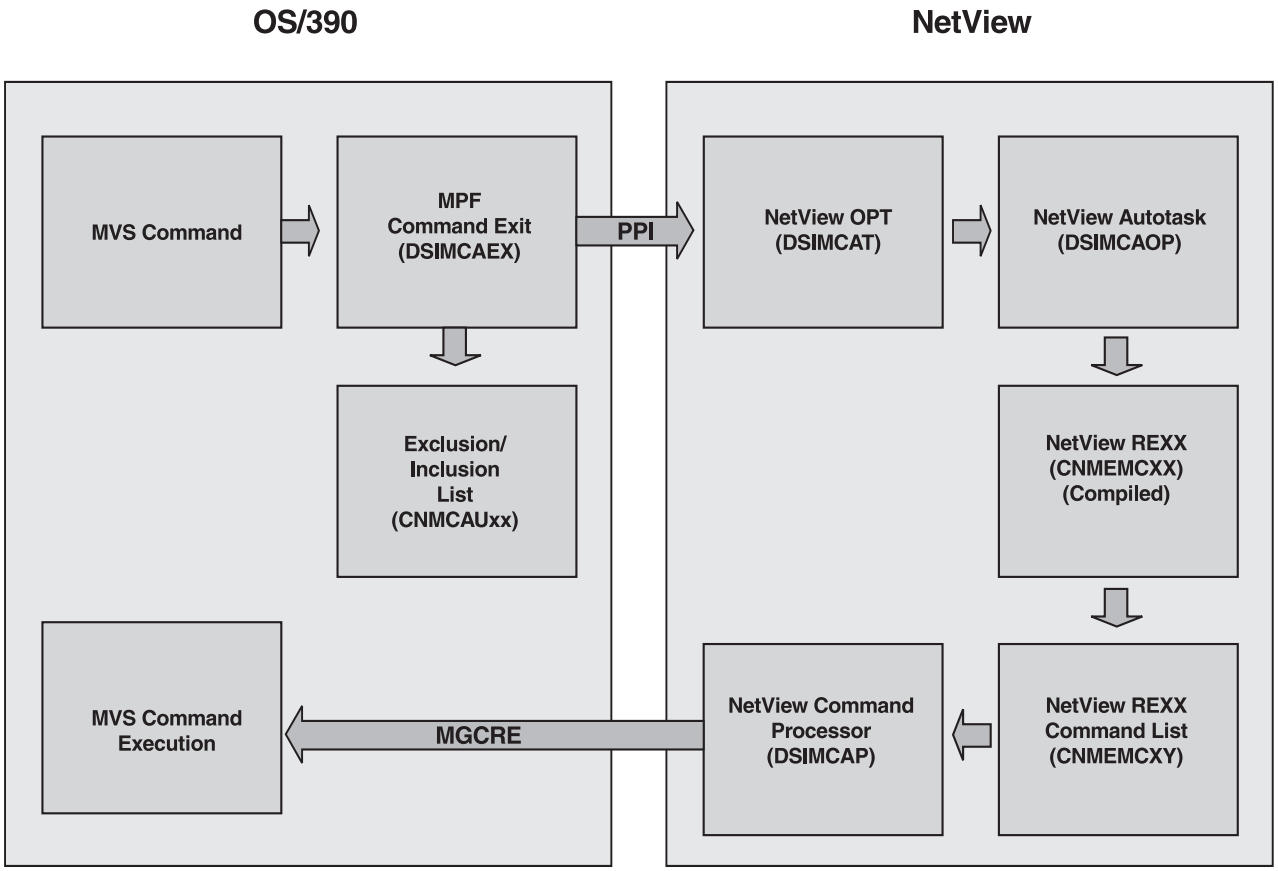


Figure 198. MVS Command Management Flow

Enabling MVS Command Management in the NetView Environment

To enable MVS command management in the NetView environment:

1. Define a new NetView operator DSIMCAOP in DSIOPF or an SAF product. If you use an operator name other than DSIMCAOP, use the following statement in the CNMSTUSR or C*xx*STGEN member, and add your operator ID. For information about changing CNMSTYLE statements, see *IBM Tivoli NetView for z/OS Installation: Getting Started*.

   ```
   function.autotask.mvsCmdMgt=operid
   ```

   If this statement is not included in the CNMSTYLE member, DSIMCAOP is the default operator ID.
2. Protect DSIMCAP, CNMEMCXX, and CNMEMCXY from unauthorized use.

   Note: If you are using an SAF product such as RACF for operator definitions and command authorization, make the equivalent updates to these definitions.
3. Verify that the tower statement in the CNMSTYLE member does not specify an asterisk (*) preceding the MVScmdMgt tower.

Enabling the MVS Command Exit on MVS

The MVS command exit uses the NetView program-to-program interface (PPI). Ensure that the NetView subsystem address space program (SSI) is started before enabling the exit.

To enable the MVS command exit for processing on MVS:

1. Ensure the load module DSIMCAEX is in a load library in the MVS LINKLST concatenation. If required, issue the following command to enable it:

   F LLA,REFRESH

2. Update an MPFLST*xx* member in PARMLIB by adding the following statement:

   ```
   .CMD USEREXIT(DSIMCAEX)
   ```

   To activate the change, issue the following command:

   SET MPF=*xx*

   where *xx* is the suffix of the MPFLST member.

3. Unless a command inclusion/exclusion list is provided, most commands are sent to the NetView program. To restrict commands from being sent to the NetView program, use a command inclusion/exclusion list. The NetView program provides a sample list CNMCAU00. You can use this sample or create your own and place it in the logical PARMLIB.

   To activate the change, issue the following command:

   SET CNMCAUT=*yy*

   where *yy* is the suffix of the CNMCAU member in PARMLIB. This also enables the inclusion/exclusion list in normal mode. If no inclusion/exclusion list is to be used, specify a value of **ON** for *yy*.

   You can then set the inclusion/exclusion list to test mode by issuing the following command:

   ```
   SET CNMCAUT=TEST
   ```

   When your test is successful, issue either of the following commands to reset the test mode:

   SET CNMCAUT=*yy*
   SET CNMCAUT=ON

4. After testing, you can add an entry to the MPFLST*xx* member to suppress message IEE295I, which is issued every time a command is modified. Otherwise, you receive the following messages for every command that is processed by the exit:

   ```
   IEE295I COMMAND CHANGED BY EXIT 043
   ORIGINAL: command ' '
   MODIFIED: command
   ```

Suppressing additional command echoes and IEE295I messages

When all of these conditions are in place, a command is echoed in the system log multiple times:

- The NetView MVS Command Management function is active
- The command is sent to NetView for processing
- The command is not rejected by CNMEMCXY for execution

IEE295I messages are logged in the system log. You can suppress the additional command echoes by specifying the `TRACKING.ECHO` statement and you can suppress the IEE295I message by specifying the `ISSUE.IEE295I` statement in the logical PARMLIB member CNMCAUaa.

The syntax of the TRACKING.ECHO statement is

```
TRACKING.ECHO = Y | N
```

The default of the TRACKING ECHO command is

```
TRACKING.ECHO = Y
```

The syntax of the ISSUE.IEE295I statement is

```
 ISSUE.IEE295I = Y | N
```

The default of the ISSUE.IEE295I is

```
ISSUE.IEE295I = Y
```

These statements can be coded anywhere in the logical PARMLIB member CNMCAUaa.

- A specification of TRACKING.ECHO = Y indicates that additional command echoes are to be logged in the system log.
- A specification of TRACKING.ECHO = N indicates that additional command echoes are not to be logged to the system log. Note that if TRACKING.ECHO = N is coded, there is not be an indication in either the system log or in the NetView log if the command is changed by CNMEMCXY. The command response can be for an entirely different command than the one that is entered originally. If you want to know when the command is changed by CNMEMCXY, specify TRACKING.ECHO = Y or else add code to CNMEMCXY to log the command before and after it is changed.
- A specification of ISSUE.IEE295I = Y indicates that the IEE295I messages are to be logged in the system log.
- A specification of TRACKING.ECHO = N indicates that the IEE295I messages are not to be logged to the system log.

If multiple TRACKING.ECHO or ISSUE.IEE295I statements are specified in the CNMCAUaa member, the last valid value is used. If no statement is coded or if a statement is not valid, the default is used.

Example: Assume that the MVS command D T is entered. Assume further that the command is not excluded and neither TRACKING.ECHO or ISSUE.IEE295I is coded (or that these statements have been specifically coded with a value of Y). These statements are logged in the system log:

```
1.  D T
2.  D T ' '
3.  IEE295I COMMAND CHANGED BY EXIT 314
4.  ORIGINAL: D T ' '
5.  MODIFIED: D T
6   D T
7.  IEE136I LOCAL: TIME=14.34.45 DATE=2009.103 UTC: TIME=18.34.45 DATE=2009.103
```

A specification of TRACKING.ECHO = N prevents lines 2 and 6 from being logged. A specification of ISSUE.IEE295I = N prevents lines 3, 4, and 6 from being logged.

If the statements were specified in CNMCAUaa as

```
TRACKING.ECHO = N
ISSUE.IEE295I = Y
```

then these statements are logged in the system log:

```
1.  D T
2.  IEE295I COMMAND CHANGED BY EXIT 423
3.  ORIGINAL: D T ' '
4.  MODIFIED: D T
5.  IEE136I LOCAL: TIME=14.39.44 DATE=2009.193 UTC: TIME = 18.34.45 DATE=2009.103
```

If the statements in CNMCAUaa are coded as

```
TRACKING.ECHO = Y
ISSUE.IEE295I = N
```

then these statements are logged in the system log:

```
1.  D T
2.  IEE295I COMMAND CHANGED BY EXIT 423
3.  ORIGINAL: D T ' '
4.  MODIFIED: D T
5.  IEE136I LOCAL: TIME=14.39.44 DATE=2009.193 UTC: TIME = 18.34.45 DATE=2009.103
```

If the statements in CNMCAUaa are coded as

```
TRACKING.ECHO = N
ISSUE.IEE295I = N
```

then these statements are logged in the system log:

```
1.  D T
2.  IEE136I LOCAL: TIME=14.39.44 DATE=2009.193 UTC: TIME = 18.34.45 DATE=2009.103
```

The following command can change the value of TRACKING.ECHO or ISSUE.IEE295I:

```
SET CNMCAUT=aa
```

If CNMCAUaa is loaded and processed successfully, and if TRACKING.ECHO and ISSUE.IEE295I statements are coded, the values specified are in effect. If the statements are not coded, the default values are used. If CNMCAUaa is not loaded or is not processed successfully, the values specified for the statements are not changed. The values for the statements default to

```
TRACKING.ECHO = Y
ISSUE.IEE295I = Y
```

The following commands do not change the value of TRACKING.ECHO or ISSUE.IEE295I:

```
SET CNMCAUT=OFF
SET CNMCAUT=TEST
```

Exclusion or Inclusion Lists

You can use MVC Command Management to control

- Whether input from a specific console is included or excluded by specifying a CONSOLE EXCLUSION LIST or a CONSOLE INCLUSION LIST, both of which are described in “Console Exclusion List and Console Inclusion List” on page 566.
- Whether a specific command is included or excluded by specifying a COMMAND EXCLUSION LIST or a COMMAND INCLUSION LIST, both of which are described in “Command Exclusion List and Command Inclusion List” on page 567.
- Whether certain other types of commands that do not fit typical formats are included or excluded by specifying a CMDTEXT EXCLUSION LIST or a

CMDTEXT INCLUSION LIST, both of which are described in "CMDTEXT Exclusion List and CMDTEXT Inclusion List" on page 568

Each of these lists is defined in a Logical Parmlib member CNMCAU*aa*, where *aa* must be alphanumeric. You can have more than one CNMCAU*aa* member in the logical PARMLIB, but only one can be active at any time.

Logical PARMLIB Member - CNMCAUaa

EXCLUSION and INCLUSION lists for consoles and commands are defined in the PARMLIB member CNMCAU*aa*.

Exclude these commands from automation processing.

```
COMMAND EXCLUSION LIST
The following are internally-issued DB2 commands:
S DSNAMSTR
S DSNAIRLM
S DSNADBM1
S DSNADIST
S DSNASPAS

 The following are internally-issued MQ Series commands:
S MQM1MSTR
S MQM1CHIN
```

Syntax for CNMCAUaa Statements

These syntax rules apply for CNMCAU*aa* statements:

- A forward slash (/) in column 1 followed by an asterisk (*) in column 2 indicates comments.
- Columns 73–80 are ignored.
- Only comments, CONSOLE EXCLUSION LIST, CONSOLE INCLUSION LIST, COMMAND EXCLUSION LIST, COMMAND INCLUSION LIST, console names, and commands are recognized.
- For non-comment statements, column 1 must be a blank or an asterisk (*).
- The wildcard character (*) is supported in console name and command text.
- An asterisk (*) in column 1 is an indicator that there is a wildcard match (0 to *n* characters match) at the beginning of the console name or the command text.

Console Exclusion List and Console Inclusion List

These rules apply to either a Console Exclusion List or a Console Inclusion List

- The CONSOLE EXCLUSION LIST (or the CONSOLE INCLUSION LIST) must start in column 2.

  Note: Do *not* add extra blanks between words.
- All excluded console names must follow the line CONSOLE EXCLUSION LIST (or CONSOLE INCLUSION LIST).
- Column 1 is reserved for the wildcard character (*).

  If no wildcard character (*) is specified, column 1 must be blank.
- There must be only one console name per line, starting at column 2.
- Each console name is assumed to be 8 characters long (including blanks).
- A wildcard character (*) can be specified at the beginning or at the end of the console name, but not both.

- An asterisk (*) in column 1 indicates that wildcard matching is selected at the beginning of the console name.
- An asterisk (*) at the end of the console name indicates that wildcard matching is selected at the end of the console name.
- If an asterisk (*) is the only character entered for the console name, it is treated as a regular character, not as a wildcard character.

Command Exclusion List and Command Inclusion List

These Command Exclusion List rules apply:

- The COMMAND EXCLUSION LIST (or the COMMAND INCLUSION LIST) starts in column 2.

  Note: Do **not** add extra blanks between words.
- All excluded commands must follow the line COMMAND EXCLUSION LIST (or COMMAND INCLUSION LIST).
- Each command can be from 1 to 122 characters long.
- Column 1 is reserved for the wildcard character (*).

  If no wildcard character (*) is specified, column 1 must be blank.
- The command must start at column 2 and can run to column 71 (if wildcard matching does not occur at the beginning of the command).
- Column 72 is the continuation column.

  Column 72 must be blank if no continuation is desired. The continuation character is not included in the string.
- Columns 73 through 80 are ignored.
- Extra blanks must not be entered in the command.
- The continuation line must start in column 2.
- Trailing blanks are deleted if column 72 is blank.
- A wildcard character (*) can be specified at the beginning or at the end of the command text, but not both.
  - An asterisk (*) in column 1 indicates that wildcard matching is selected at the beginning of the command text.
  - An asterisk (*) at the end of the command text indicates that wildcard matching is selected at the end of the command text.
  - If an asterisk (*) is the only character entered for the command text, it is treated as a regular character, not as a wildcard character.

Usage Notes for COMMAND EXCLUSION LIST:

- If you are using a COMMAND EXCLUSION LIST, add all internally issued START DB2 commands to this list.

  To see how these commands look in your environment, look in your SYSLOG right after the START DB2 command has been issued. DB2 internally issues a number of subsequent START commands to start-up its subordinate address spaces. It is these commands that you want to add to the command exclusion list.

  At the present time, a DB2 subsystem consists of at least five address spaces with names such as:

```
DSNAMSTR
DSNAIRLM
DSNADBM1
DSNADIST
DSNASPAS
```

- If you are using a COMMAND EXCLUSION LIST, add all internally issued MQ START commands to this list. To see what these commands look like in your environment, look in your SYSLOG right after the MQM1 START OMGF command has been issued. MQ internally issues a number of subsequent START commands to start-up its subordinate address spaces. It is these commands that you want to add to the command exclusion list. At the present time, an MQ subsystem consists of at least two address spaces with names like:

  ```
  MQM1MSTR
  MQM1CHIN
  ```

- For additional information see "General Processing of CONSOLE and COMMAND Inclusion and Exclusion Lists" on page 572.

Usage Notes for COMMAND INCLUSION LIST:

- If you are using a COMMAND INCLUSION LIST, do not add internally issued DB2 START commands to this list.
- If you are using a COMMAND INCLUSION LIST, do not add internally issued MQ START commands to this list.
- For additional information see "General Processing of CONSOLE and COMMAND Inclusion and Exclusion Lists" on page 572.

CMDTEXT Exclusion List and CMDTEXT Inclusion List

Some commands might not have the same command format described in "Command Exclusion List and Command Inclusion List" on page 567. For example, some NetView commands can have a blank instead of a comma as a keyword or value delimiter. For those commands that do not conform to the previously described command format, the CMDTEXT EXCLUSION LIST and the CMDTEXT INCLUSION LIST provide similar function. These rules apply to the CMDTEXT EXCLUSION LIST and CMDTEXT INCLUSION LIST:

- Both the CMDTEXT EXCLUSION LIST and the CMDTEXT INCLUSION LIST must start in column 2.
- All excluded command strings must follow the line CMDTEXT EXCLUSION LIST and all included command strings must follow the line CMDTEXT INCLUSION LIST.
- All the rules listed under COMMAND EXCLUSION LIST and COMMAND INCLUSION LIST regarding wildcard characters and continuation characters are applicable to the CMDTEXT EXCLUSION LIST and CMDTEXT INCLUSION LIST.
- CMDTEXT EXCLUSION LIST and CMDTEXT INCLUSION LIST are mutually exclusive; that is, only one can be defined.

A distinction must be made between the COMMAND INCLUSION (or EXCLUSION) LIST and the CMDTEXT INCLUSION (or EXCLUSION) list. If a command is contained in the COMMAND INCLUSION (or EXCLUSION) LIST, a blank character separates the command from any additional keywords or values. If a second blank character occurs, all information following the second blank character is discarded. However, for the CMDTEXT INCLUSION (or EXCLUSION) LIST, all information contained in the command entered is compared to the detail in the CMDTEXT INCLUSION (or EXCLUSION) list. In creating any of these lists -- the CONSOLE INCLUSION LIST, the CONSOLE EXCLUSION LIST, the

COMMAND INCLUSION LIST, the COMMAND EXCLUSION LIST, the CMDTEXT INCLUSION LIST, or the CMDTEXT EXCLUSION LIST -- you can be as specific as necessary, or, you can use wildcard characters to describe consoles or commands. For example, a list can contain a specific command

```
START MYPROC,INT1=USER,PARM='ANY PARM
```

in which case the entire command and its keywords and values must be contained in the list to be included (or excluded). Or, a list can contain only a command such as

```
START *
```

in which case the START command, regardless of additional keywords and values is included (or excluded).

Order of matching

This is the sequence in which consoles and commands are matched to any of the lists described. Once a match is found in any of these sequential steps, the matching process completes and further matching steps end.

1. CONSOLE EXCLUSION LIST (or CONSOLE INCLUSION LIST)
2. COMMAND EXCLUSION LIST (or COMMAND INCLUSION LIST)
3. CMDTEXT EXCLUSION LIST (or CMDTEXT INCLUSION LIST)

However, within each logical grouping, follow these guidelines:

- Do not specify both a CONSOLE EXCLUSION LIST and a CONSOLE INCLUSION LIST.
- Do not specify both a COMMAND EXCLUSION LIST and a COMMAND INCLUSION LIST.
- Do not specify both a CMDTEXT EXCLUSION LIST and a CMDTEXT INCLUSION LIST.

Starting MVS Command Management

After the NetView command exit is defined in the MPF member and the NetView autotask and Optional Task are defined to NetView, you can start MVS Command Processing by:

1. Activating MVS command exit
2. Starting MVS Command Processing

MVS Command Management is deactivated whenever SET MPF=xx is entered and processed, even if the MPFLSTxx member defines DSIMCAEX as an MVS command exit. If MVS Command Management has been deactivated by the SET MPF=xx command, you must restart it by entering either SET CNMCAUT=ON or SET CNMCAUT=xx after the NetView MVS Command Exit DSIMCAEX has been activated again.

Activating the MVS Command Exit

To activate the MVS command exit, issue this MVS command:

```
SET MPF=xx
```

where *xx* is the suffix of MPFLST*xx*. MPFLSTxx must have the statement:

```
.CMD USEREXIT(DSIMCAEX)
```

Starting MVS Command Processing

To start MVS Command Processing, issue this command:

```
SET CNMCAUT=ON or SET CNMCAUT=aa
```

Where aa is the suffix for CNMCAU*aa* member (except ON).

Displaying the MVS Command Management Setting

To find out the name of the active CNMCAU*aa* member, the TRACKING ECHO statement, the ISSUE.IEE295I, and the CNMCAUT setting, issue this command:

```
DISPLAY CNMCAUT or
D CNMCAUT
```

To find out the contents of CNMCAU*aa*, issue:

```
D CNMCAUT=TABLE or
DISPLAY CNMCAUT=TABLE
```

Stopping MVS Command Management

Stopping MVS Command Management and Keeping the CNMCAUaa Member

To stop MVS command management and keep the active CNMCAU*xx* PARMLIB member in storage, issue this command from any MVS console:

```
SET CNMCAUT=OFF
```

This does not change the TRACKING.ECHO or the ISSUE.IEE295I setting. When the command CNMCAUT=ON is issued, the CNMCAU*xx* PARMLIB member is active again.

Stopping MVS Command Management and Deleting the CNMCAUaa Member

To delete the CNMCAU*aa* PARMLIB member and stop MVS Command Management, issue this command from any MVS console:

```
SET CNMCAUT=DELETE
```

This stops MVS commands from being sent to NetView. NetView MVS Command Exit still gets control for every MVS command. Deleting the CNMCAUaa member also changes the setting of TRACKING.ECHO to Y and also changes the setting of ISSUE.IEE295I to Y. These are the default values.

Stopping the MVS Command Exit from Being Invoked

To stop NetView MVS Command Exit from being invoked, issue this command from any MVS console:

```
SET MPF=yy
```

The *yy* is a MPFLSTyy member which does *not* have the .CMD USEREXIT(DSIMCAEX) statement. Or you can enter a SET MPF=NO command to stop MPF processing.

Note: Use SET MPF=NO only as a last resort because it stops all MPF processing.

Deactivating the MVS Command Exit

To deactivate the MVS command exit, issue one of these commands from an MVS console:

MVS Command Management is deactivated when SET MPF=xx is entered, event if the MPFLSTxx member defines DSIMCAEX as an MVS command exit. You can restart MVS Command Processing by entering either SET CNMCAUT=ON or SET CNMCAUT=xx.

Note: Use SET MPF=NO only as a last resort because it stops all MPF processing.
or

```
SET MPF=NO
```

or

```
SET MPF=yy
```

where *yy* is the suffix of a MPFLST*yy* member that contains the same statements as MPFLST*xx*, except the CMD USEREXIT (DSIMCAEX) statement.

MVS Command Management is made unavailable whenever a SET MPF=xx is entered, even though the MPFLISTxx member defines DSIMCAEX as an MVS command exit. In order to restart MVS Command Management, enter SET=CNMCAUT=ON or SET CNMCAUT=xx.

If you want information about...	Refer to...
MPFLST*nn*	*z/OS* library

Testing MVS Command Management

To test MVS command management, issue this command from an MVS console:

SET CNMCAUT=TEST

In TEST mode, each command is processed as if CNMCAUT=ON. The MVS command is processed by MVS immediately after the command exit processing is completed, before it is processed by NetView. NetView does not send the command back to MVS.

To turn off TEST mode, enter:

SET CNMCAUT=OFF

The TEST mode is turned off when a SET CNMCAUT=ON or SET CNMCAUT=*xx* command is completed successfully. Issuing SET CNMCAUT=TEST and SET CNMCAUT=OFF does not change the setting of either the TRACKING.ECHO statement or the ISSUE.IEE295I statement.

Starting the Exclusion or Inclusion List

To start the EXCLUSION or INCLUSION list issue this command from any MVS console:

SET CNMCAUT=*aa*

Where *aa* is the suffix of PARMLIB member CNMCAU*aa*.

Changing the Exclusion or Inclusion List

To change the EXCLUSION or INCLUSION list, you can either start a new CNMCAU*aa* member, or change the existing member and re-enter the SET CNMCAUT=*aa* command.

General Processing of CONSOLE and COMMAND Inclusion and Exclusion Lists

You can use console and command inclusion and exclusion lists in any combination. This Command List Chart describes the logic used:

		COMMAND LIST			
		EXCLUSION		INCLUSION	
		match	no match	match	no match
Console List	exclusion match	IGNORE	IGNORE	IGNORE	IGNORE
	exclusion no match	IGNORE	-->NetView	-->NetView	IGNORE
	inclusion match	IGNORE	-->NetView	-->NetView	IGNORE
	inclusion no match	IGNORE	IGNORE	IGNORE	IGNORE

- A match in a CONSOLE EXCLUSION list results in the command being IGNORED.

  A non-match in the list results in the command being tested against the COMMAND list.
- A match in a COMMAND EXCLUSION list results in the command being IGNORED.

  A non-match in the list results in the command being sent to NetView for processing.
- A match in a COMMAND EXCLUSION list results in the command being IGNORED.

  A non-match in the list results in the command being sent to NetView for processing.
- A non-match in a COMMAND INCLUSION list results in the command being IGNORED.

  A match in the list results in the command being sent to NetView for processing.

Commands Excluded by NetView Command Exit

These commands are not sent to NetView by the MVS command exit (DSIMCAEX) even if they are included in the CNMCAU*aa* member:

Any command that has a single quotation mark as the first character of the command

CONTROL E,SEG

CONTROL E,PFK

CONTROL E,N

CONTROL E,F
CONTROL E
CONTROL V
CONTROL T
CONTROL S
CONTROL D
CONTROL C
CONTROL A
CONTROL E,nn
CONTROL
K E,SEG
K E,PFK
K E,N
K E,F
K E
K V
K T
K S
K D
K C
K A
K
MOUNT
LOGON
RO T=nnn,dest,mvscmd or ROUTE T=nnn,dest,mvscmd except when *dest* starts with an asterisk (*) as in ***ALL** or ***OTHER**, or when *dest* starts with a left parenthesis as in **(sysnamelist)**.
S INIT.INIT
SET MPF=NO
START INIT.INIT
T MPF=NO

Restrictions

- Using MVS Command Management to pass START commands to NetView for processing might cause problems for some uses of the START command because the return codes (R15) and ASID (R0) that are returned by MGCRE are not accurate as the result of using this function.

  Note: This is known to cause a problem for START commands that are internally issued by DB2 and MQ Series.

  Exclude internally issued DB2 and MQ START commands from any command automation processing, either by adding these commands to a command exclusion list or by specifically keeping them out of a command inclusion list.
- Whenever a TSO end-user logs on, MVS Command Management specifically excludes LOGON commands that are issued internally.

  The LOGON command also returns an ASID and return code (like the START command).

- MVS Command Management specifically excludes MOUNT commands that are issued by operators to manually request a tape mount.

  The MOUNT command also returns an ASID and return code (like the START command).
- MVS Command Management specifically excludes various K (CONTROL) commands that are issued by operators to control real extended multiple console support (EMCS) consoles.

  The K command processor makes assumptions that are not compatible with the command automation code.
- Strings to the right of an equal sign (=) in REXX cannot exceed 250 characters.

  Command text is passed into the REXX exec CNMEMCXY in printable hexadecimal form (to prevent REXX from parsing the command). Only commands of 123 characters or less can be processed (4 characters are used to convey the command length in printable hexadecimal form).

  Note: Do *not* code wait processing in CNMEMCXY because that can delay the handling of MVS commands, which remain queued until the wait ends.

  Wait processing, in this case, includes REXX and PIPE waits, WTORs, and Parse Pull types of commands.
- Because of the current mechanism that is used to "tag" commands so that they are processed only once, the maximum command length that can be handled is further reduced to 122 characters.

  The only commands that are known to approach these limits are internally issued SEND commands that are used to notify TSO end users when jobs have complete or NJE file transmissions have occurred. These commands are currently exempted from processing by use of a console exclusion list specifying a console name of INTERNAL and INSTREAM.

MVS Command Management Processing on NetView

The line CONSOLE EXCLUSION LIST starts in column 2.

Note: Do *not* add extra blanks between words.

After MVS command management processing is activated, every MVS command that is not excluded is sent to NetView for further processing. On the NetView side, the optional task DSIMCAT receives the MVS command from the PPI and invokes a REXX CLIST CNMEMCXY. When CNMEMCXY runs, it receives these parameters:

Note: Do *not* code wait processing in CNMEMCXY because that can delay the handling of MVS commands, which remain queued until the wait ends.

Wait processing, in this case, includes REXX and PIPE waits, WTORs, and Parse Pull types of commands.

MODE=mode	*mode* is T (test), or O (on)
ISYN=isyn	*isyn* is the issuing system name in hexadecimal. The isyn is 16 hexadecimal digits long.
CNNM=consname	*consname* is the issuing console name in hexadecimal. The consname is 16 hexadecimal digits long.

C4ID=consid — *consid* is the issuing console ID in hexadecimal. The consid is 8 hexadecimal digits long.

TOKN=token — *token* is the users's command token in hexadecimal. The token is 8 hexadecimal digits long.

AUTH=auth — *auth* is the user's command authorization in hexadecimal. The auth is 4 hexadecimal digits long.

ASID=asid — *asid* is the user's ASID in hexadecimal. The asid is 4 hexadecimal digits long.

TRNM=termname — *termname* is the user's terminal name in hexadecimal. The termname is 16 hexadecimal digits long.

CLNM=conclass — *conclass* is the console class name in hexadecimal. The conclass is 16 hexadecimal digits long.

CART=cart — *cart* is the command and response token in hexadecimal. The cart is 16 hexadecimal digits long.

OCID=ocid — *ocid* is the originating console ID in hexadecimal. The ocid is 16 hexadecimal digits long.

UTKN=utoken — *utoken* is the user token in hexadecimal. The utoken is 160 hexadecimal digits long and cannot be modified. It must be returned to MVS unchanged.

CTXT=cmdtext — *cmdtext* is the command text in hexadecimal. The cmdtext is up to 250 hexadecimal digits long. The first 4 digits are the length of the command.

Note: All input to CNMEMCXY is in hexadecimal, except MODE. To examine input, convert it to character format by using the REXX function X2C.

After examining the command, you send a return code to the invoking REXX CLIST to indicate you want the command returned to z/OS for further processing.

These are return codes that you can return and their meaning:

0 Continue processing. The command is not changed. The MVS command is sent back to MVS.

4 Command text changed. The changed command must be sent to MVS.

If a return code of 4 is returned, the modified MVS command must be saved in a SAFE named MVSCMD.

8 The command must not be returned to MVS.

If MODE=T is specified, the MVS command is not returned to MVS, regardless of the return code.

If the length of a command is changed, update the Length field (the first 4 bytes of the command text) accordingly. (The maximum length of the command text is 122 characters.)

Note: Do *not* code wait processing in CNMEMCXY because that can delay the handling of MVS commands, which remain queued until the wait ends.

Wait processing, in this case, includes REXX and PIPE waits, WTORs, and Parse Pull types of commands.

Protecting MVS Command Management Processing

To prevent an operator from executing an unauthorized MVS command, the NetView command DSIMCAP must be protected from all NetView operators, except DSIMCAOP or the NetView operator defined in the following CNMSTYLE statement:

```
Function.autotask.mvsAuto=opid
```

Ensure that this is the only task that can be permitted to issue DSIMCAP, CNMEMCXX, or CNMEMCXY.

DSIMCAOP or the defined autotask used by MVS Command Management Processing must be protected so that other NetView operators cannot send commands to the autotask for execution if AUTHCHK=TARGETID is used.

Appendix I. The Sample Set for Automation

NetView includes a set of samples to help you get started with automated operations. These samples are referred to as the ***sample set for automation***. The sample set for automation is designed to show examples of automation techniques; it is not intended to be a drop-in solution to automation.

The sample set for automation consists of the following sample sets:

Message suppression
: This set contains two MVS message-suppression lists (one conservative and one aggressive).

Basic automation
: This set contains automation table entries, command lists, and start-up procedures that demonstrate how routine operator actions can be automated.

Advanced automation
: This set includes automation table entries and command lists that demonstrate how to initialize, recover, and shut down subsystems and applications using automation techniques discussed in this manual.

Log analysis program
: This set contains a log analysis program for JES2 and JES3 that can be modified for use in analyzing other logs. The analysis program helps you identify frequently issued messages that you might want to suppress or automate.

Setup samples
: This set contains samples to help you rename the samples in the other parts of the sample set for automation.

You can identify the samples in the sample set for automation by their names. Samples with names beginning in CNMS62, CNMS64, CNME62, or CNME64 belong to the sample set for automation. Descriptions of all of the samples included in the sample set for automation are contained in "Cross-Reference Listing of Command Lists and Samples" on page 609.

Using the Sample Set for Automation

To use the sample set for automation as examples of automated operations, you can follow these steps:

1. Rename all of the samples in the sample set for automation.
2. Begin using parts of the message suppression sample set for examples of message suppression. Message suppression is described in Chapter 19, "Suppressing Messages and Filtering Alerts," on page 299.
3. Use the log analysis program to help suppress messages and automate NetView. The log analysis program identifies common messages that are likely candidates for automation.
4. Prepare for communication between NetView and MVS. This step is a prerequisite for activating the basic and advanced sample sets.

5. Use the basic automation sample set for examples of message automation, a technique discussed in Chapter 22, "Automating Messages and Management Services Units (MSUs)," on page 317. Activate parts of this sample set that are appropriate to your environment.
6. Use the advanced automation sample set for example of coordinated automation, a technique discussed in Chapter 23, "Establishing Coordinated Automation," on page 355. Activate parts of the advanced automation sample set that are appropriate to your environment.

Locating and Renaming the Sample Set for Automation

The following sections describe how to locate and rename the sample set for automation on an MVS system.

The following table shows the names and locations of the samples after you have installed them as part of the NetView installation process.

Table 35. Locations of the Sample Set for Automation on MVS Systems

Sample Set	Sample Type	Library	Member Names
Message Suppression	Sample MPFLSTxx Members	SYS1.CNMSAMP	CNMS6201-CNMS6202
Basic Automation	Miscellaneous NetView Samples	SYS1.CNMSAMP	CNMS6205-CNMS6206
	Sample Procedures	SYS1.CNMSAMP	CNMS6211-CNMS6214
	Sample Parameters	SYS1.CNMSAMP	CNMS6221-CNMS6224
	Sample Command Lists	SYS1.CNMCLST	CNME6201-CNME6205
Advanced Automation	Miscellaneous NetView Samples	SYS1.CNMSAMP	CNMS6401-CNMS6410
	Sample Panels	SYS1.CNMSAMP	CNMS64P0-CNMS64P5
	Sample Command Lists	SYS1.CNMCLST	CNME6400-CNME6440
Log Analysis	Sample Analysis Program	SYS1.CNMSAMP	CNMS6207
	Sample JCL	SYS1.CNMSAMP	CNMS62J2
Setup	Renaming JCL	SYS1.CNMSAMP	CNMS62J1

Use sample CNMS62J1 to rename the samples supplied in the sample set for automation. Rename the samples before you can run them, because they refer to each other by the new names. Take the following steps to rename the samples:

- Examine CNMS62J1 to ensure the symbolics set in the PROC statement are appropriate to your environment and to ensure that the OUTDD DD statement for each step contains the data set where you want to store the samples. The JCL supplied in CNMS62J1 copies the renamed samples into new data sets. If you prefer to use existing data sets, update the sample JCL contained in CNMS62J1 accordingly.
- If you change the library where the PARMLIB members are stored, update samples CLRSMF (CNMS6212) and LGPRNT (CNMS6213) to point to the new library.

- Because the data set that contains the PROCLIB samples must be a valid PROCLIB data set, include that data set as one of the DD statements on the PROC00 DD in your JES procedure.
- Include the data sets containing the renamed DSIPARM and DSIPRF samples in the appropriate DD statements in the NetView start procedure. You must stop and restart NetView for the changes made to the NetView start procedure to take effect. If you are using the samples provided with the NetView program, that NetView start procedure is CNMPROC (CNMSJ009).
- Submit CNMS62J1, which runs as a batch job, into the system input stream for processing using either the TSO SUBMIT command or the NetView SUBMIT command.
- If desired, create a second copy of the renamed samples. Then, if you change some of the samples, you have a backup copy already renamed.

Using the Message Suppression Sample Set

Use the message suppression sample set to begin suppressing messages, or you can incorporate parts of the sample set into your existing message suppression. See Chapter 19, "Suppressing Messages and Filtering Alerts," on page 299 for information.

Using the Log Analysis Program

The log analysis program does not perform automation but is a tool to help you create automation of your own. See "Log Analysis Program" on page 463 for information.

Setting Up Communication between NetView and MVS

Before you can activate the basic or advanced sample set on the MVS system, NetView must be defined to send commands to the operating system and receive MVS system messages. Define NetView to MVS as a subsystem (with IEFSSNxx) and ensure that messages are forwarded to NetView. Check your MPFLSTxx PARMLIB member and ensure that you are not suppressing a message that one of the sample sets requires.

"Preparing the z/OS System for System Automation" on page 291 describes defining NetView as a subsystem and forwarding messages from MVS to NetView.

Using the Basic Automation Sample Set

The following sections describe the functions provided by the basic automation sample set, how the sample automation table provides those functions, and how the basic automation sample set is activated.

Functions Performed by the Basic Automation Sample Set

The samples in the basic automation sample set demonstrate how routine operator actions can be automated using the automation facilities described in Part 4, "NetView Program Automation Facilities," on page 105.

The samples include automation table entries, command lists, and start-up procedures. The command lists are written in both the NetView command list language and REXX. If they are run on a system that has NetView REXX capability, the command lists run in REXX. If they are run on a system that has no NetView REXX capability, they run in the NetView command list language.

The samples in the basic automation sample set contain a small subset of the automated actions that can be performed. The samples demonstrate how simple, routine responses to events occurring in the environment can be automated using the NetView automation facilities. The samples included in the basic automation sample set provide the following automated actions:

- Clear the SYS1.LOGREC data set
- Print the SYS1.LOGREC data set
- Vary a channel online
- Reply to a GTFTRACE parameter request
- Issue the MVS START command for the SMF memory dump task
- Print the network log after the DSILOG task switches from the primary to the secondary or vice versa
- Monitor JES2 spool utilization and purge held files older than 24 hours when utilization exceeds 70%

Automation Table Used in the Basic Automation Sample Set

This section describes the automation table used in the basic automation sample set and the ways in which the functions are provided. Figure 199 shows the automation table.

```
*************************************************************************
*  (C) COPYRIGHT IBM CORP. 2010                                         *
*    IEBCOPY    SELECT MEMBER=((CNMS6205,ACOTABLE,R))                  *
*  LAST CHANGE:    08/25/10                                             *
*                                                                       *
*  DESCRIPTION:                                                         *
*    DESCRIPTION: SAMPLE DSIPARM - MSG AUTOMATION DEFS FOR BASIC        *
*                                  AUTOMATION SAMPLE SET                *
*                                                                       *
*  CNMS6205 CHANGED ACTIVITY:                                           *
*  CHANGE CODE  DATE      DESCRIPTION                                   *
*  -----------  --------  ------------------------------------------*
*************************************************************************
*
 IF MSGID = 'IFB040I' 1
  THEN EXEC(CMD('MVS S CLRLOG') ROUTE(ONE AUTO1));
*
 IF MSGID = 'IFB060E' 2
  THEN EXEC(CMD('MVS S LGPRNT') ROUTE(ONE AUTO1));
*
 IF MSGID = 'IOS150I' & TOKEN(3) = DEVICE 3
  THEN EXEC(CMD('MVS VARY ' DEVICE ',ONLINE') ROUTE(ONE AUTO1));
*
 IF MSGID = 'AHL125A' & TEXT(1) = REPLYID . 4
  THEN EXEC(CMD('MVS REPLY ' REPLYID ',U')
  ROUTE(ONE AUTO1));
*
IF MSGID='DSI556I' & TOKEN(2) ='DSILOG' & TOKEN(6) = 'CLOSE'' & 5
          TEXT = . 'DDNAME = '' LOGID ''' RETURN CODE =' .
   THEN EXEC(CMD('MVS S DSIPRT,NAME='LOGID) ROUTE(ONE AUTO1));
*
 IF (MSGID = 'IEE362A' | MSGID = 'IEE362I') & TEXT = STRNG 6
  THEN EXEC(CMD('IEE362A ' STRNG) ROUTE(ONE AUTO1));
*
 IF MSGID = '$HASP646' & TEXT = STRNG 7
  THEN EXEC(CMD('$HASP646 ' STRNG) ROUTE(ONE AUTO1));
```

Figure 199. Basic Automation Sample Set Automation Table Entries

Figure 200 lists the messages automated by the Basic Automation Sample Set Automation Table.

```
IFB040I  SYS1.LOGREC AREA IS FULL,hh.mm.ss  A
IFB060E  SYS1.LOGREC NEAR FULL  B
IOS150I   DEVICE ddd NOW AVAILABLE FOR USE  C
xx AHL125A   RESPECIFY TRACE OPTIONS OR REPLY U  D
DSI556I   DSILOG  :  VSAM DATASET 'CLOSE' COMPLETED,DDNAME='ddname'  E
          RETURN CODE=X'code',ACB ERROR FIELD=X'code'.
IEE362n   SMF ENTER DUMP FOR SYS1.MANx ON ser  F
$HASP646  xx PERCENT SPOOL UTILIZATION  G
```

Figure 200. Messages Automated by the Basic Automation Sample Set Automation Table

Issuing Commands: Statement 1 in Figure 199 on page 580 issues a system command directly from the automation table. The statement automates message A in Figure 200.

When NetView receives message A, it issues the MVS START command for the sample procedure CLRLOG, which clears the SYS1.LOGREC data set. CLRLOG goes to the autotask AUTO1 for processing. MVS is a NetView command processor that enables the entry of the MVS system, subsystem, and application commands from NetView. The command you issue can also be a NetView or VTAM command, which you can issue directly from the automation table without any preceding command processor.

In statement 1, the command to be processed is routed to only one operator and goes to the autotask AUTO1. Because no action is taken if AUTO1 is not active, you can specify the statement in Figure 201.

```
ROUTE(ONE AUTO1 opid1 opid2 opid3)
```

Figure 201. Specifying Multiple Autotasks and Operators on the ROUTE Command

In the ROUTE command in Figure 201, `opid1`, `opid2`, and `opid3` are other autotasks or operator IDs. In that case, the command is routed to the first one on the list that is active or logged on.

Statement 2 in Figure 199 on page 580, uses the same techniques to print the SYS1.LOGREC data set upon receipt of message ID IFB060E, message B in Figure 200.

Assigning a Value to a Variable: Statement 3 in Figure 199 on page 580, varies a channel online upon receipt of the message for message ID IOS150I, message C in Figure 200.

Statement 3 captures the information in token 3 (*ddd*, the device number), and passes it to the command as a variable called DEVICE. The fact that DEVICE has no single quotation marks around it indicates that it is a variable rather than a comparison item. The variable DEVICE is then used in the action to be processed. Again, MVS is the NetView command processor allowing the VARY command to be issued from NetView, and the command is to be routed to the autotask AUTO1 for processing.

Statement 4 in Figure 199 on page 580, passes a variable to a command or command procedure to reply to a WTOR. Statement 4 responds to message AHL125A, message D in Figure 200. Message AHL125A requests that you respecify trace options for the Generalized Trace Facility (GTF), or reply U to continue initialization. In message AHL125A, `xx` is the reply ID. Statement 4

captures the reply ID as the variable REPLYID. TEXT(1) indicates the text beginning in position 1. The period (.) is a placeholder that means that only the text before the next blank should be used for REPLYID. Thus, the variable REPLYID obtains the text from position 1 to the first blank. The automation table can then pass the REPLYID variable to the command specified in the EXEC statement. If you left out the period, all the text to the end of the message would go into the REPLYID variable.

The rest of statement **4** replies U to the request for GTFTRACE parameters. This causes the MVS command REPLY to be issued for the appropriate reply ID, under the autotask AUTO1.

Statement **5** in Figure 199 on page 580, for message ID DSI556I, message **E** in Figure 200 on page 581, parses part of the message text and uses that in the MVS START command that is issued from the automation table. That command prints the primary or secondary network log upon receipt of the DSI556I message.

Statement **5** has three comparison items plus the assignment of the variable LOGID. The automation table initiates action only if all of the following comparison conditions are met:

- The message ID must be DSI556I.
- The second token must be DSILOG.
- The sixth token must be 'CLOSE'.

If these conditions are met, then:

- The message text is parsed, assigning the text to the variable named LOGID, which follows the string DDNAME = ' and precedes the string ' RETURN CODE =.
- The MVS START command is issued for the sample procedure DSIPRT, which prints the log for either the primary or secondary log, whichever has just been closed.

Wherever a single quotation mark (') appears in the text of a message and must be indicated as part of a comparison condition, it is represented as two consecutive single quotation marks ('). Of the three single quotation marks that surround CLOSE in the automation table statement, the outside quotes indicate the text that must be contained in TOKEN(6) for the comparison condition to be met. Without the single quotation marks, the text would be assigned to a variable. The remaining single quotation marks reduce to the ones that enclose CLOSE in the message itself.

Invoking Command Lists and Command Processors: Statements **1** through **5** in Figure 199 on page 580 have issued commands directly from the automation table. All the commands are MVS system or subsystem commands, but they can be application commands, or NetView or VTAM commands. Statement **6** in Figure 199 on page 580 (for message ID IEE362A or IEE362I, message **F** in Figure 200 on page 581) illustrates the use of NetView automation facilities when more complex actions than a single command are required. In those cases, the automation table statement can invoke a NetView command list (written in the NetView command list language or REXX) or command processor (written in PL/I, C, or assembler).

Statement **6** looks for the message ID IEE362A or IEE362I, both of which have the text shown in message **F** .

Statement 6 specifies that, if either of the message IDs is received, the entire text of the message is captured as a variable named STRNG and passed to the command list IEE362A for automation processing. The command list parses the message string to determine the value of x in the message and uses that value when issuing the MVS START command to start the sample procedure CLRSMF.

You generally use a command list or command processor when the action you are automating involves issuing more than one command or when the process of extracting information from the message is too complex for the automation table. Also, multiline write-to-operator (MLWTO) messages require special message processing that must be done in a command list or command processor.

Statement 7 in Figure 199 on page 580, for message ID $HASP646, message G in Figure 200 on page 581, is part of a monitoring sample for JES2 spool utilization. To monitor the spool utilization, set a NetView timer to issue the $D SPOOL command at certain intervals. In the samples provided, the NetView command EVERY is issued in the initial command list whenever the autotask AUTO1 is initialized, and the $D SPOOL command is scheduled to be issued every 24 hours.

When the $D SPOOL command runs, the message $HASP646 results. When the automation table receives message $HASP646, it passes the message text to the NetView command list $HASP646. That command list checks the spool utilization percentage. If the spool utilization is greater than 70%, the command list cancels all held jobs more than 24 hours old. In addition, the command list sets a second timer that drives the $DSPOOL2 command list every hour. That command list monitors the spool space until spool utilization goes below 20%.

The technique of issuing query commands at regular intervals and taking actions based on the status is called ***proactive monitoring***. The basic automation sample set uses a very basic example of proactive monitoring. See “Proactive Monitoring” on page 587.

Activating the Basic Automation Sample Set

If you elect to pattern your automation after the sample set for automation, you can activate the basic automation sample set. To activate samples from the basic automation sample set, you must:

- Ensure that the necessary messages are forwarded to NetView
- Define command synonyms for the command lists
- Prepare and activate the sample automation table
- Activate the AUTO1 autotask
- Test the basic automation sample set

The following sections describe the steps in activating the basic automation sample set.

Defining Command List Synonyms: For the basic automation sample set to work, you must define the necessary command synonyms by combining the command definitions contained in CNMS6206 with the existing CNMCMD.

CNMS6206 contains one entry for each basic automation sample set command list. The command synonyms are used throughout the basic automation sample set. If any of those synonyms conflicts with a synonym already defined in your system, you might have to change the basic automation sample set command synonym. If so, be sure to change it in all of the samples that refer to that command synonym.

Preparing and Activating the Sample Automation Table: The sample automation table for the basic automation sample set is ACOTABLE (CNMS6205). To run ACOTABLE:

- Prepare the NetView automation table for use.
- Test the syntax of the table.
- Activate the table.

Preparing the Sample Automation Table: You should become familiar with what is in the basic automation sample set automation table, ACOTABLE (CNMS6205) and decide which samples to use. If you decide not to activate all of the basic automation sample set samples, you must remove the ACOTABLE entries that drive the samples you do not want to use.

The automation table supplied with the basic automation sample set, ACOTABLE (CNMS6205), can be used as a standalone NetView automation table or can be combined with an existing table, perhaps one that you are already using in production. If you are already using an automation table, you can copy the entries that drive the samples you want to use from ACOTABLE into your existing automation table. Alternatively, you can leave the entries in a separate file and include them in your table with a %INCLUDE statement.

Ensure that the ACOTABLE entries do not conflict with existing entries in your automation table. For example, if you have duplicate message IDs without CONTINUE(Y), only the first statement in the table is driven when the message is received, because the automation table is processed sequentially. If there are conflicts, edit the table to resolve the conflicts. After including the ACOTABLE entries, reorganize your table for processing efficiency. See "Design Guidelines for Automation Tables" on page 231 for a list of principles to follow.

Testing the Syntax of the Sample Automation Table: When you have prepared an automation table by combining the one from the basic automation sample set with your existing automation table, test the syntax by issuing command **1** in Figure 202, in which `automem` is the name of the member containing the automation table. If the syntax is correct, you see the message CNM501I, message **2** in Figure 202.
Otherwise, correct the syntax and perform the test again.

```
AUTOTBL MEMBER=automem,TEST 1
CNM501I  TEST OF MESSAGE AUTOMATION FILE "automem" WAS SUCCESSFUL. 2
```

Figure 202. Testing Your Automation Table

Activating the Sample Automation Table: After a successful test of the automation table, you can activate it. It is not necessary to stop and restart NetView to change which automation table is active. Activate your automation table using the AUTOTBL command in Figure 203.

```
AUTOTBL MEMBER=automem
```

Figure 203. Activating Your Automation Table

The table you specify is the active automation table until you stop and restart NetView, deactivate that automation table, or activate another automation table.

To activate the table automatically every time NetView comes up, specify the table in the CNMSTYLE member.

Activating the Autotask AUTO1: Ensure that your initial command list starts the autotask AUTO1, which is used by the basic automation sample set. If you are

using CNME1034, the initial command list shipped with NetView, the AUTO1 autotask is already started for you. If you are using another initial command list, ensure it contains command **1** in Figure 204.

```
AUTOTASK OPID=AUTO1  1
EXCMD AUTO1 AUTO1IC  2
```

Figure 204. Activating Autotask AUTO1

If you want to use the JES2 spool utilization monitoring and recovery samples, schedule the $D SPOOL command for processing at regular intervals. Do that by running the AUTO1IC command list under AUTO1. You can run the AUTO1IC command list under AUTO1 by entering command **2** in Figure 204 from an authorized operator's console.

To automatically schedule the $D SPOOL command for repeated processing whenever AUTO1 logs on and runs its initial command list, call the AUTO1IC command list from the initial command list for AUTO1. The initial command list for AUTO1 is specified in member DSIPROFC (CNMS1026) of DSIPRF in the NetView samples as being LOGPROF2 (CNME1032). Therefore, if you are using the NetView samples, add a line to LOGPROF2 to call the AUTO1IC command list.

Testing the Basic Automation Sample Set: Before putting the basic automation sample set into production, verify that the NetView automation table entries result in the actions you anticipate. For information about testing automation, see Chapter 34, "Automation Table Testing," on page 471.

Using the Advanced Automation Sample Set

The advanced automation sample set contains samples that show how you might use NetView to automate functions such as initialization, monitoring, recovery, and shutdown of subsystems and applications. It is not intended to be a drop-in solution to automation but an example of the coordinated automation technique described in Chapter 23, "Establishing Coordinated Automation," on page 355.

The advanced automation sample set includes command lists, several full-screen panels that display the status of the automation, and some network definition samples, such as automation table entries. The command lists are written in both the NetView command list language and REXX. If they are run on a system that has NetView REXX capability, the command lists run in REXX. If they are run on a system that has no NetView REXX capability, they run in the NetView command list language.

Note: In this section, the term ***product*** refers to a subsystem or application that is automated using the advanced automation sample set. "Cross-Reference Listing of Command Lists and Samples" on page 609 contains a cross-reference listing of all samples contained in the advanced automation sample set.

Functions Performed by the Advanced Automation Sample Set

The advanced automation sample set demonstrates how you can automate the initialization, monitoring, recovery, and shutdown of specific products on a system using the techniques discussed in Chapter 23, "Establishing Coordinated Automation," on page 355. The sample set also demonstrates the use of an operator interface tailored to the automated environment using VIEW panels as

discussed in "Creating Full-Screen Panels" on page 364. The advanced automation sample set includes command lists that automate the operation of the products listed in Table 36.

Table 36. Processes Automated by the Advanced Automation Sample Set for Each Product

Product	Initialization	Monitoring	Recovery	Shutdown
JES2	●	●	●	●
JES3	●	●	●	●
VTAM	●	●		●
TSO	●	●		●
IMS	●	●	●	●
CICS	●	●	●	●

Initialization: The advanced automation sample set demonstrates how to automate the initialization of an entire system and of specific products. The advanced automation sample set accomplishes an orderly initialization of a system by first processing command list AOPIVARS (CNME6400), the central command list of the advanced automation sample set. AOPIVARS initializes the automation global variables, loads the automation table to be used, and logs on autotask AUTOMGR, which is the central advanced automation sample set autotask.

The initial command list of AUTOMGR, AOPIMGIC (CNME6402), provides an orderly start-up of all automated products that were initialized in AOPIVARS by logging on an autotask for each automated product and by processing the timer command that periodically triggers the proactive monitoring command list AOPMACT. The initial command list of each autotask that AUTOMGR starts, AOPIGNIC (CNME6403), starts the product for which the autotask is responsible, making sure that all other products upon which that product is dependent are active.

The start command list for each product issues the command to start the product and waits for the message that indicates the product has successfully initialized. The initial command list of each autotask also issues a timer command to periodically run command list AOPMCHEK to check the autotask status. To initialize a specific product, process the start command list for the product.

The initialization portion of the advanced automation sample set uses passive monitoring. The initialization-completion messages for each product are routed to the product autotask from the automation table.

Monitoring: The advanced automation sample set uses passive monitoring of messages for its product initialization, shutdown, and recovery. The advanced automation sample set uses proactive monitoring to periodically check the status of automated products and autotasks.

Passive Monitoring: The basic automation sample set contains examples of passive monitoring of system-related operations. Entries are contained in the automation table for messages requiring simple, routine responses by the operator. Once a message and its appropriate automation procedure are added to the table, the response to the message becomes automatic, replacing the need for the operator to respond to the message.

In the advanced automation sample set, the samples provide automation of system-operations-related passive monitoring in more complex situations. Instead of focusing on responding to a single event, the samples focus on what is required to perform processes such as initialization, recovery, or shutdown of subsystems and applications within a single system.

An Example of Passive Monitoring: Suppose that you are an operator on a system with no automation installed. CICS runs on your system, and you want to keep it running at all times. If CICS ends abnormally for any reason, the message in Figure 205 is displayed.

```
+DFH0606 ABEND xxxx HAS BEEN DETECTED
$HASP395 CICS    ENDED
```

Figure 205. CICS Abend Message

Unless you notice the message when it appears, you find out about the failure only if a user calls to complain or if you browse the log and see the failure message.

With passive monitoring, it is not necessary to know about the problem, because it is corrected automatically. The automation table contains an IF statement that watches for the DFH0606 termination message, traps it, and invokes the command list DFH0606 to restart the application upon receipt of the message. For example, the automation table supplied with the advanced automation sample set automates the DFH0606 action with the statement in Figure 206.

```
IF MSGID = '+DFH0606' THEN
   EXEC(CMD('DFH0606') ROUTE (ONE AUTOMGR))
   DISPLAY(Y) NETLOG(Y);
```

Figure 206. Passive Monitoring in the Advanced Automation Sample Set

As soon as the failure message is received, this statement detects it and restarts the application. No action on your part is required.

Proactive Monitoring: Proactive monitoring involves querying the system and network to monitor the status of the environment. It is implemented in the advanced automation sample set by using timer commands to process command lists at regular intervals. Proactive monitoring is controlled by autotask AUTOMGR, which is the central automation autotask used in the advanced automation sample set. Two components in the proactive monitoring samples are supplied with the advanced automation sample set:

- An active-monitor command list, AOPMACT (CNME6439), which ensures that automated products remain active after initialization.
- An autotask-monitor command list, AOPMCHEK (CNME6440), which periodically checks all autotasks supplied with the advanced automation sample set to ensure that they remain active.

AOPMACT issues query commands to the automated products that you have defined to automation, actively soliciting information on the current status of those components. The status information returned by the components is compared to the desired state of the components as defined in global variables. Where the desired state and the current actual state do not match, messages are sent to notify personnel of a potential problem.

AOPMCHEK periodically sends messages to all advanced automation sample set autotasks and waits for a response. Global variables are used to keep track of the

status of the autotasks. The PPT is used to check autotask AUTOMGR, and AUTOMGR is used to check all other advanced automation sample set autotasks. If an autotask does not respond, a message is sent to inform the system operator of a potential problem.

Autotasks are a powerful tool in automation. However, the fact that autotasks generally run unattended without an associated console means that a failed or unresponsive autotask can go unnoticed. The result might be that all new work for an autotask either is queued and not processed or, if the autotask is logged off, is never received.

The advanced automation sample set uses the following method to minimize the amount of time an autotask failure goes unnoticed:

1. A timer command schedules processing of the autotask-monitor command list, AOPMCHEK (CNME6440), for each autotask from the autotask's initial command list. In the advanced automation sample set, AOPMCHEK is scheduled to be processed periodically. The time period for AOPMCHEK to be called is set in AOPIVARS. To monitor autotask AUTOMGR, which controls the other autotasks, AOPMCHEK is processed under the PPT. AOPMCHEK is processed under autotask AUTOMGR for all other autotasks in the advanced automation sample set. The autotask operator ID and a status of CHECK are passed as parameters to AOPMCHEK.
2. When AOPMCHEK is processed with the string of CHECK, the status of the autotask in question is checked to see if it is already set to CHECK. If it is, an acknowledgment was never sent to AOPMCHEK from the last autotask check, and an error message is sent to the system operator, indicating that the autotask is unresponsive. If the status of the autotask is not already set to CHECK, the status is set to CHECK, and a message is sent to the autotask requesting a response.
3. The message sent to an autotask requesting a response is intercepted by the automation table and turned into a command to process command list AOPMCHEK to generate an acknowledgment.
4. If the autotask is responding, AOPMCHEK is processed to generate an acknowledgment. AOPMCHEK then changes the status of the autotask from CHECK to ACTIVE.

An Example of Proactive Monitoring: This section discusses how proactive monitoring works for the AUTOJES autotask. AOPIGNIC is the initial command list for AUTOJES. AOPIGNIC issues command **1** in Figure 207, assuming the time period to check autotasks is set to 5 minutes in AOPIVARS.

```
EXCMD AUTOMGR,EVERY 5,ID=JESCHK,AOPMCHEK AUTOJES CHECK 1
MSG AUTOJES CHECKING AUTOTASK - AUTOJES 2
DSI039I MESSAGE FROM AUTOMGR : CHECKING AUTOTASK - AUTOJES 3
```

Figure 207. Proactive Monitoring for the AUTOJES Autotask

Command **1** is a check to be performed by AUTOMGR on the status of AUTOJES every 5 minutes. When command list AOPMCHEK is called, the autotask is identified (AUTOJES), and the type of processing to be performed is identified (CHECK).

When AOPMCHEK is processed with a check requested and the current status of AUTOJES is ACTIVE, the status is set to CHECK, and command **2** in Figure 207 is issued.

As a result, message DSI039I, message **3** in Figure 207 on page 588, is generated and sent to AUTOJES.

The message is intercepted by the automation table statement in Figure 208:

```
IF MSGID = 'DSI039I' & TOKEN(9) = 'AUTOJES' THEN
   EXEC(CMD('EXCMD AUTOMGR,AOPMCHEK AUTOJES ACKNOWLEDGEMENT')
   ROUTE (ONE AUTOJES)) SYSLOG(N) NETLOG(N) DISPLAY(N);
```

Figure 208. Proactive Monitoring for Message DSI039I

When message DSI039I is received and the ninth token is an autotask name, the command in Figure 209 is sent to autotask AUTOJES.

```
EXCMD AUTOMGR,AOPMCHEK AUTOJES ACKNOWLEDGEMENT
```

Figure 209. Automation Table EXCMD Command in Response to DSI039I Message

The command in Figure 209 calls command list AOPMCHEK under autotask AUTOMGR (to acknowledge that AUTOJES is active).

- If AUTOJES is responding, the command is processed, causing command list AOPMCHEK to change the status of AUTOJES from CHECK to ACTIVE.
- If AUTOJES is not responding, the command is not processed and the status of AUTOJES remains CHECK.

When AOPMCHEK is processed with a check requested and the current status of AUTOJES is CHECK, the autotask has not responded to the last check sent to it. That results in an error message being sent to the system operator, indicating that AUTOJES is not responding.

Recovery: The advanced automation sample set demonstrates how to recover automated products upon receipt of a message indicating an abnormal termination of the product. The function is equivalent to an operator's attempting to restart a product after receiving a console message indicating an abnormal termination.

When certain messages are received by the automation table, a restart of the failing product is attempted by processing a command list. The command list sets a timer command that, if processed after a certain period of time, indicates that the recovery attempt has failed. The recovery command list then processes the start command list for the failed product. If the recovery attempt is successful, the timer command that issues the recovery-failure message is purged.

The recovery portion of the advanced automation sample set uses passive monitoring. Messages received by the automation table that indicate either an abnormal failure or an unsuccessful restart attempt initiate the recovery process. Increasing recovery function involves:

- Adding an automation table statement for the messages you want to automate, to indicate that a restart is required
- Adding recovery command lists, as required

Shutdown: The advanced automation sample set demonstrates how to shut down automated products on a system. The automatic shutdown of a system or a specific product follows the same process that an operator attempting to shut down a system or product follows. To shut down a specific product, the shutdown command is issued. Shutdown is complete once the message indicating the product has completed shutdown successfully is received. To shut down all

automated products, the shutdown must be ordered so that certain products are not shut down until products that are dependent upon them have completed their shutdown processes.

The main command list to shut down all automated products is AOPSMAIN (CNME6412). AOPSMAIN shuts down products in an orderly manner by stopping them in the reverse of the order in which they were initialized. AOPSMAIN also ensures that no product shuts down before any dependent products have completed their shutdowns. When AOPSMAIN determines that nothing that depends on a product remains active, the stop command list for the product is invoked. The stop command list issues the stop command for that product and waits for the message indicating the shutdown is complete. To shut down a specific product, process the stop command list for that product.

The shutdown portion of the advanced automation sample set uses passive monitoring. Messages received indicating a product has completed its shutdown process are trapped by the automation table and routed to the shutdown autotask for that product.

Enhancing the Operator Interface: Command list AOPUSTAT (CNME6438) demonstrates how to use VIEW panels to display automation information for automated products. The central panel, CNMS64P0, can be used for keeping personnel up to date on any status changes, as automation makes or discovers them. A change in the status of an automated product results in CNMS64P0 being dynamically refreshed. A description of the operator interfaces contained in the advanced automation sample set is included in "Operator-Interface Command List and Panels" on page 598.

Command Lists Used in the Advanced Automation Sample Set

The following sections describe how command lists and the automation table are used in the advanced automation sample set to initialize, recover, actively monitor and shut down a system.

Note: If you want to manually start or stop a product while automation is in effect, you should use the appropriate initialization or shutdown command list. This ensures that automation keeps accurate information concerning the desired status of the product and the time the status last changed.

Advanced Automation Sample Set Functions

This section lists the command lists and samples that perform the various functions included in the advanced automation sample set.

- Command lists exist to set initial automation global-variable settings for each automated product. The automation global variables contain information such as a product's current status and the MVS command used to start it.

  Related command lists: AOPIVARS, AOPIGUPD.
- Automation table entries exist to suppress unnecessary messages, reply to messages, route messages to the correct operators, and process command lists in response to messages.

  Related sample: DSITBL11 (CNMS6405).
- One controlling autotask activates an autotask for each automated product. Each product's dedicated autotask is responsible for doing the automation tasks related to that product.

  Related command lists: AOPIMGIC, AOPIGNIC, AOPIVARS.

Related samples: CNMS6408, CNMS6409, CNMS6410 (DSIOPF and automated operator profiles).

- Initialization routines issue MVS commands to start up automated products, respond to messages required for initialization, and ensure the products become active within a time limit.
  - Command lists exist that initialize products and ensure that they become active within a reasonable time limit. The command lists set the desired status of the products to active and set the time the current status of the products changed.

    Related command lists: AOPIJES3, AOPIJES2, AOPIVTAM, AOPITSO, AOPIIMS, AOPICICS.
  - Command lists exist that are processed from the automation table for product initialization.

    Related command list: $HASP426.
- Shutdown routines issue MVS commands to shut down automated products, respond to messages required for shutdown, and ensure the products become inactive within a time limit.
  - Command lists exist to shut down products and ensure that they become inactive within a reasonable time limit. The command lists set the desired status to inactive and set the time the current status of the products changed.

    Related command lists: AOPSMAIN, AOPSJES3, AOPSJES2, AOPSVTAM, AOPSTSO, AOPSTSO2, AOPSIMS, AOPSIMS2, AOPSCICS, AOPSPURG.
  - Command lists are processed using timer commands for product shutdown.

    Related command list: VTAMTMRZ.
  - Command lists are processed from the automation table for product shutdown.

    Related command lists: AOPTJRC3, DFS996I, DFS000IB.
- Recovery routines trap messages that indicate an undesirable status of a product and attempt to re-initialize the product.
  - Command lists are processed from the automation table for product recovery.

    Related command lists: IAT3714, IAT3708, $HASP095, $HASP098, $HASP085, DFS629I, DFH0606, IKT002I.
  - Command lists are processed using timer commands for product recovery.

    Related command lists: JESTMRA, IMSTMR, CICSTMRA.
- Active-monitoring routines check the status of automation at intervals and monitor the status of products and the automated operators.
  - An active-monitoring command list exists that checks periodically to ensure that the current status of the automated products and the desired status match. It is started by the AUTOMGR autotask initial command list (AOPIMGIC).

    Related command list: AOPMACT.
  - An autotask-monitor command list exists that makes use of the automation table to periodically monitor the advanced automation sample set autotasks to ensure that they remain active. It is started by the initial command list for autotask AUTOMGR (AOPIMGIC) and by the initial command list for the other advanced automation sample set autotasks (AOPIGNIC).

    Related command list: AOPMCHEK.
- Operator-started command lists and associated display panels exist that present the current status of all automated products and specific information regarding each automated product.

Related command list: AOPUSTAT.

Related panels: CNMS64P0 - CNMS64P5.

The advanced automation sample set provides the following standard message notifications:

- Messages are sent to the system operator in case of automation failure or undesirable conditions that automation cannot resolve.
- A message is sent to the network log on entry to any command list. If you do not want this message in the network log, you can remove the statement that sends it.

Naming Conventions for Advanced Automation Sample Set Command Lists

The command lists in the advanced automation sample set are named according to the following general rules:

- Command lists that perform specific operator tasks and are not triggered by messages have AOP as their first three letters. The fourth letter represents the type of action taken by the command list:
  AOPI... Initialization
  AOPM... Proactive monitoring
  AOPS... Shutdown
  AOPU... Utility
- Command lists called from the automation table have names identical, where possible, to the messages that trigger them in the automation table. For example, $HASP426 is a command list that is called to handle JES2 message $HASP426.
- Command lists called by using a NetView timer from one of the advanced automation sample set command lists generally have the characters TMR contained in the command list name.

Initialization and Active-Monitoring Command Lists

The initialization and active-monitoring command lists are:

AOPIVARS Main initialization command list
DSITBL11 Activates automation table
AOPIGUPD Sets common global variable (called many times)
AUTOMGR Activates AUTOMGR autotask
AOPIMGIC AUTOMGR initial command list
AUTOJES Activate JES autotask
AOPIGNIC Generic initial command list
AOPIJES2 Starts JES2 and ensures that it has started
$HASP426 Specifies options to JES2
AOPIJES3 Starts JES3 and ensures that it has started
AOPMCHEK Periodically checks AUTOJES autotask
AUTOVTAM Activates VTAM autotask
AOPIGNIC Generic initial command list
AOPIVTAM Starts VTAM and ensures that it has started
AOPMCHEK Periodically checks AUTOVTAM autotask
AUTOTSO Activates TSO autotask
AOPIGNIC Generic initial command list
AOPITSO Starts TSO and ensures that it has started
AOPMCHEK Periodically checks AUTOTSO autotask
AUTOIMS Activates IMS autotask
AOPIGNIC Generic initial command list
AOPIIMS Starts IMS and ensures that it has started
AOPMCHEK Periodically checks AUTOIMS autotask
AUTOCICS Activates CICS autotask

AOPIGNIC Generic initial command list
AOPICICS Starts CICS and ensures that it has started
AOPMCHEK Periodically checks AUTOCICS autotask
AOPMACT Actively monitors to ensure that products remain active
AOPMCHEK Periodically checks AUTOMGR autotask

The CNMSTYLE member must be customized to run AOPIVARS. Information about the CNMSTYLE member can be found in *IBM Tivoli NetView for z/OS Installation: Getting Started*.

AOPIVARS:

1. Activates the automation table
2. Sets the global variables used in the advanced automation sample set, such as the start, shutdown, and display commands for each product that is automated
3. Logs on the AUTOMGR automation task ID

AOPIMGIC is the initial command list for AUTOMGR when it is logged on.

1. AOPIMGIC starts an autotask for each automated product. The name of each autotask is built by concatenating the letters AUTO with the name of the product, as set in AOPIVARS. The product name set in AOPIVARS must be four or fewer letters. Shipped with the advanced automation sample set are definitions for autotasks AUTOJES, AUTOVTAM, AUTOTSO, AUTOIMS, and AUTOCICS. Each uses AOPIGNIC as its initial command list.
2. AOPIMGIC then issues a timer command to call the active-monitor command list AOPMACT at a given interval.
3. AOPIMGIC also issues a timer command to periodically invoke command list AOPMCHEK to ensure that AUTOMGR stays active.

AOPIGNIC is a generic initial command list that starts an automated product by calling the start-up command list of the product for which it is called (AOPIJES2, AOPIJES3, AOPIVTAM, AOPITSO, AOPIIMS, or AOPICICS). AOPIGNIC also issues a timer command to periodically call command list AOPMCHEK to ensure that the autotask stays active. Some products require another product to be active before they can be activated. AOPIGNIC has special logic to wait for a predecessor product (if any) to activate before attempting to activate the product at hand.

AOPIJES2 is called by AOPIGNIC if you are operating in a JES2 environment. AOPIJES2 checks the status of JES2:

- If JES2 is already active, AOPIJES2 sets the desired status to ACTIVE and exits.
- If JES2 is not active, AOPIJES2 issues the JES2 start command and waits a specified period of time for JES2 to activate. If JES2 does not become active within that time period, a warning message is issued to the system operator, indicating that JES2 has not yet initialized.

  Starting JES2 causes message $HASP426 (specify options) to be generated, which triggers a command list with the same name. The $HASP426 command list responds to the outstanding request with the user-specified reply set in AOPIVARS at initialization time.

AOPIJES3 is called by AOPIGNIC if you are operating in a JES3 environment. AOPIJES3 checks the status of JES3:

- If JES3 is already active, AOPIJES3 sets the desired status to ACTIVE and exits.
- If JES3 is not active, AOPIJES3 issues the JES3 start command and waits a specified period of time for JES3 to activate. If JES3 does not become active

within that time period, a warning message is issued to the system operator, indicating that JES3 has not yet initialized.

AOPIVTAM is called by AOPIGNIC. AOPIVTAM checks the status of VTAM:

- If VTAM is already active, AOPIVTAM sets the desired status to ACTIVE and exits.
- If VTAM is not active, AOPIVTAM issues the VTAM start command and waits a specified period of time for VTAM to activate. If VTAM does not become active within that time period, a warning message is issued to the system operator, indicating that VTAM has not yet initialized.

AOPITSO is called by AOPIGNIC. AOPITSO checks the status of TSO:

- If TSO is already active, AOPITSO sets the desired status to ACTIVE and exits.
- If TSO is not active, AOPITSO issues the TSO start command and waits a specified period of time for TSO to activate. If TSO does not become active within that time period, a warning message is issued to the system operator, indicating that TSO has not yet initialized.

AOPIIMS is called by AOPIGNIC. AOPIIMS checks the status of IMS:

- If IMS is already active, AOPIIMS sets the desired status to ACTIVE and exits.
- If IMS is not active, AOPIIMS issues the IMS start command and waits a specified period of time for IMS to activate. If IMS does not become active within that time period, a warning message is issued to the system operator, indicating that IMS has not yet initialized.

AOPIIMS can start data communications by answering a WTOR for IMS. It can also start IMS regions, if appropriate lines in the command list are uncommented.

AOPICICS is called by AOPIGNIC. AOPICICS checks the status of CICS:

- If CICS is already active, AOPICICS sets the desired status to ACTIVE and exits.
- If CICS is not active, AOPICICS issues the CICS start command and waits a specified period of time for CICS to activate. If CICS does not become active within that time period, a warning message is issued to the system operator, indicating that CICS has not yet initialized.

AOPMCHEK is called by AOPIMGIC and AOPIGNIC through a timer command to periodically check the advanced automation sample set autotasks to ensure that they remain logged on and active. AOPMCHEK sends a message to an autotask using the NetView MSG command. The message is intercepted by the automation table, which processes a command to the autotask to call AOPMCHEK again with an acknowledgment that it is still active. If an autotask becomes unresponsive, AOPMCHEK sends a message to the system operator.

AOPMACT is called by AOPMGNIC to monitor the automated products to ensure that they remain active. If a product becomes inactive, a message is sent to the system operator.

Recovery Command Lists

The command lists for recovery are:

JES2:

$HASP095	Stores the JES2 abend code
$HASP098	Replies to message based on JES2 abend code
$HASP085	Restarts JES2
JESTMRA	Resets JES2 abend counter

JES3:

IAT3714	Issues reply to JES3 memory dump request and restarts JES3
IAT3708	Resets JES3 status to ACTIVE and purges JES3 recovery check timer
AOPTJRC3	Resets JES3 abend counter

IMS:

DFS629I	Restarts IMS if it ends abnormally
IMSTMR	Updates IMS timer to blank after 15 minutes

CICS:

DFH0606	Restarts CICS if it ends abnormally
CICSTMRA	Updates CICS timer to blank after 5 minutes

JES2 recovery is triggered by message $HASP095.

1. When message $HASP095 is received by the automation table, command list $HASP095 is called. Command list $HASP095 saves the abend code contained in the message and increments the abend counter.
2. Command list $HASP098 is next called from the automation table in response to message $HASP098; it replies to message $HASP098 based on the abend code saved by command list $HASP095.
   - If the abend code is $PJ2, the operator issued the abend command and the status of JES2 is set to STOPPING.
   - Otherwise, a timer command schedules command list JESTMRA to be called after five minutes, and the status of JES2 is changed to ABEND. When JESTMRA is called, it resets the abend counter for JES2 to 0. The desired replies to message $HASP098 are set in AOPIVARS as message- response variables.
3. Command list $HASP085 is called by the automation table and restarts JES2.

JES3 recovery is triggered by message IAT3714. When message IAT3714 is received by the automation table, command list IAT3714 is called. IAT3714 updates the JES3 status to ABEND.

- If this is the first time JES3 has ended abnormally, a reply to the message is sent, the abend count is set to indicate that a JES3 abend has occurred, and a timer command is issued to display the message JES3 RECOVERY FAILED after five minutes.
- If JES3 has ended abnormally a second time or the JES3 reply was not valid, a message is sent to the operator to indicate the need for a manual recovery process, and command list AOPTJRC3 is called to reset the abend count.

When message IAT3708 is intercepted by the automation table, indicating that JES3 activation is complete, command list IAT3708 is started to reset the JES3 status to ACTIVE and purge the JES3 message timer, and AOPTJRC3 is called to reset the IAT3714 reply counter to 0.

CICS recovery is triggered by message DFH0606. When the message is received by the automation table, command list DFH0606 is called. DFH0606 updates the CICS status to ABEND.

- If CICS has ended abnormally in the last five minutes, a message is sent to the operator, indicating so.
- Otherwise, a timer command to drive command list CICSTMRA after five minutes is issued and the CICS start command is issued. CICSTMRA sets the indicator that an abend occurred in the last five minutes to off.

IMS recovery is triggered by message DFS629I. When the message is received by the automation table, command list DFS629I is called. DFS629I updates the IMS status to ABEND.

- If IMS has ended abnormally in the last 15 minutes, a message is sent to the operator, indicating so.
- Otherwise, a timer command to call command list IMSTMR after 15 minutes is issued and the IMS start command is issued. IMSTMR sets the indicator that an abend occurred in the last 15 minutes to off.

Shutdown Command Lists

The command lists for shutdown are:

AOPSMAIN Command list to order shutdown of all automated products

AOPSCICS Initiates CICS shutdown

AOPSIMS Initiates IMS shutdown by issuing a message to users to log off and by setting a timer to call AOPSIMS2.

- DFS000IB - Stores IMS region numbers for shutdown
- AOPSIMS2 - Completes shutdown of IMS
- DFS996I - Captures and stores the WTOR for later use

AOPSTSO Initiates TSO shutdown by issuing a message to users to log off and by setting a timer to call AOPSTSO2.

- AOPSTSO2 - Completes TSO shutdown

AOPSVTAM Initiates VTAM shutdown and sets timer to call VTAMTMRZ if required.

- VTAMTMRZ - Issues the cancel command to shutdown VTAM if necessary

AOPSJES2 Initiates JES2 shutdown by calling AOPSPURG and issuing command to shutdown JES2.

- AOPSPURG - Drains all units (printers, punches, initiators, etc.) You must customize this command list for your environment.

AOPSJES3 Initiates JES3 shutdown.

- AOPTJRC3 - Resets the reply count for message IAT3714 to zero (0)

AOPSMAIN shuts down all products that have values set in AOPIVARS. If a product is dependent on another product to shut down before it can shut down, AOPSMAIN periodically checks if the required prior product shutdown is complete before starting the subsequent shutdown procedure. The products are shut down in the reverse of the order in which they are listed in AOPIVARS. AOPSMAIN calls these command lists:

- AOPSCICS to stop CICS
- AOPSIMS to stop IMS
- AOPSTSO to stop TSO
- AOPSVTAM to stop VTAM
- AOPSJES2 to stop JES2
- AOPSJES3 to stop JES3

AOPSCICS checks the status of CICS:

- If CICS is already down, AOPSCICS sets the desired status to DOWN and exits.
- If CICS is not down, the command to stop CICS is issued. A message is issued to the system operator if the shutdown does not complete within a set time limit.

AOPSIMS is called and broadcasts a message to users to log off because IMS is shutting down in two minutes and issues a timer command to call command list AOPSIMS2 after two minutes:

- AOPSIMS updates the desired status of IMS to DOWN. It also checks the current-status variable and exits if it is already set to DOWN or sets the variable to STOPPING.
- AOPSIMS broadcasts a message warning users to log off within two minutes.
- AOPSIMS displays active regions, and command list DFS000IB, which is issued from the automation table, stores the IMS region numbers.
- AOPSIMS then issues a timer command to call AOPSIMS2 after two minutes. AOPSIMS2 issues commands to stop the IMS regions, data communications, and IMS.
  - If a message indicating successful shutdown is received within a set time period, AOPSIMS2 updates the current-status variable to DOWN, and command list DSF996I, which is issued from the automation table, captures the IMS WTOR for later use.
  - If the message is not received within the set time period, AOPSIMS2 issues a message to the system operator to indicate a problem.

AOPSTSO issues a message to users telling them that TSO is to be shut down. AOPSTSO then issues a timer command to call command list AOPSTSO2. AOPIVARS specifies how long AOPSTSI waits before calling AOPSTSO2. AOPSTSO2 starts after the time elapses and checks the status of TSO:

- If TSO is down, AOPSTSO2 sets the desired status to DOWN and exits.
- If TSO is not already down, it issues the command to stop TSO. A message is issued to the system operator if the shutdown does not complete within a set time limit.

AOPSVTAM checks the status of VTAM:

- If VTAM is down, AOPSVTAM sets the desired status to DOWN and exits.
- If VTAM is not down, AOPSVTAM issues a timer command to call command list VTAMTMRZ after three minutes and issues the command to stop VTAM. If VTAM shuts down correctly, the timer calling VTAMTMRZ is purged.

  VTAMTMRZ checks the status of VTAM and exits if it is down. If VTAM is not down, VTAMTMRZ issues a cancel to VTAM. A message is issued to the system operator if the shutdown does not complete within a set time limit.

AOPSJES2 checks the status of JES2:

- If JES2 is already down, AOPSJES2 sets the desired status to DOWN, and exits.
- If JES2 is not down, it issues command list AOPSPURG. AOPSPURG drains all of the devices to JES2.

  After all devices are drained, the command to stop JES2 is entered. A message is issued to the system operator if the shutdown does not complete within a set time limit.

AOPSJES3 checks the status of JES3:

- If JES3 is already down, AOPSJES3 sets the desired status to DOWN, and exits.
- If JES3 is not DOWN, AOPSJES3 issues the command to shut JES3 down. A message is issued to the system operator if the shutdown does not complete within a set time limit.

Operator-Interface Command List and Panels

The advanced automation sample set includes a utility command list that can be used along with VIEW panels to help operate and monitor an automated system. The operator-interface command list and panels display only those products defined in AOPIVARS.

Note: If the number of automated products grows to more than nine, command list AOPUSTAT and panel CNMS64P0 must be changed.

Automation Display Command List: **AOPUSTAT** displays a panel that shows the current status and time last updated for the automated products.

- From the panel displayed (CNMS64P0), choosing a specific product displays complete information regarding the status of the product, including the commands and command lists used to start and stop the product.
- From the specific product-information panel (CNMS64P1), you can display the message-response variables for that product as set in AOPIVARS. Each of the panels has an associated help panel that explains the function provided by its associated panel.

Automation Display Panels: All the automation display panels are called from command list AOPUSTAT, which can be called using a synonym of AOSTAT.

CNMS64P0 is the main display panel. CNMS64P0 displays all the automated products along with the current status, the status the product should be based on the history of start or stop attempts (DESIRED STATE), the time the status was last checked by proactive monitoring, and the time the status last changed for each product. CNMS64P0 also allows you to enter the number associated with a specific product and view more complete information concerning it with panel CNMS64P1. CNMS64P3 is the help panel for CNMS64P0. Figure 210 shows an example of panel CNMS64P0.

Note: Panel CNMS64P0 is automatically updated for any changes by automation to the current status, desired status, or time-status-changed variables, thereby ensuring that CNMS64P0 displays current status information.

```
CNMS64P0                 OPER2               12/21/10     15:53
          PRODUCT /    CURRENT    DESIRED    CHECKED      STATUS
          SUBSYSTEM    STATUS      STATE     AT           CHANGED AT
        ============  =========  ==========  ===========  ==========
    1     JES          ACTIVE     ACTIVE     15:51        15:28
    2     VTAM         ACTIVE     ACTIVE     15:51        15:29
    3     TSO          ACTIVE     ACTIVE     15:51        15:30
    4     CICS         ACTIVE     ACTIVE     15:51        15:32
    5     IMS          ACTIVE     ACTIVE     15:51        15:32
  Select a number and press ENTER for product specific information
    OR press ENTER to refresh this panel.
  Action===>
          PF1= Help   PF2= End   PF3= Return
          PF6= Roll
```

Figure 210. Sample CNMS64P0 Display

CNMS64P1 is a generic panel that is called when a product is selected, on panel CNMS64P0, for which specific information is required. CNMS64P1 displays information contained in panel CNMS64P0, the variable values of the commands used to initialize, shut down, and display the status of the product, and the command lists used to initialize and shut down the product. Specifying **R** on this panel indicates that you want to see the message-response-variable values associated with the product in question. The information is located in panel

CNMS64P2. CNMS64P4 is the help panel for CNMS64P1. Figure 211 shows how panel CNMS64P1 looks when TSO has just started. Once TSO becomes active, the STATUS field would be changed to ACTIVE from STARTING, and the time the product became active would be filled in.

```
CNMS64P1                OPER2               12/21/10     15:30
                           TSO
  BECAME ACTIVE AT:                         WENT INACTIVE AT:
                 TSO     AUTOMATION VALUES
                --------------------------
                CURRENT STATUS    =  STARTING
                DESIRED STATUS    =  ACTIVE
                START     =  S TSO
                STOP      =  P TSO
                DISPLAY   =  D J,TSO
                START COMMAND LIST -  AOPITSO
                STOP COMMAND LIST  -  AOPSTSO
  Press R and ENTER to display message responses for TSO
  OR press ENTER to refresh this panel.
  Action===>
          PF1= Help   PF2= End   PF3= Return
          PF6= Roll
```

Figure 211. Sample CNMS64P1 Display

CNMS64P2 displays the message-response-variable values for the product currently being examined on panel CNMS64P1. The message-response variables are in command list AOPIVARS. CNMS64P5 is the help panel for CNMS64P2. Note that CNMS64P2 only displays information; you cannot change a number on the panel to change the value of a variable.

Miscellaneous Samples

Besides the six VIEW panels previously discussed, the following 10 samples are provided in the advanced automation sample set:

CNMS6401 Command definition statements for MVS command verbs so they can be issued at NetView operator stations. They are not required for the advanced automation sample set to function correctly.

CNMS6402 Command definition statements for JES2 command verbs so they can be issued at NetView operator stations. They are not required for the advanced automation sample set to function correctly.

CNMS6403 Command definition statements for JES3 command verbs so they can be issued at NetView operator stations. They are not required for the advanced automation sample set to function correctly.

CNMS6404 Command definition statements for the advanced automation sample set command lists. They must be added to your existing CNMCMD.

CNMS6405 Automation-table entries for the advanced automation sample set. Add them to your existing automation table before activating the samples. Command list AOPIVARS (CNME6400) activates them in table DSITBL11. If you place them in a table with a different name, modify AOPIVARS to reflect the name you use.

CNMS6406 TSO command list to make a copy of a command list written in the NetView command list language, without comments. CNMS6406 removes lines that conform to the format used by most Tivoli samples to indicate comments. That is, the EXEC removes lines that contain an asterisk in column one followed by a space in column two or that consist of 20 asterisks in a row. Before running

CNMS6406, check that the file you wish to copy to ensure that it conforms to this format. Ensure that no lines other than comments meet the criteria for removal.

CNMS6408 Automated-operator definitions required for the advanced automation sample set. They must be added to your existing DSIOPF definitions.

CNMS6409 Autotask profile for the advanced automation sample set autotask AUTOMGR.

CNMS6410 Generic autotask profile for all advanced automation sample set autotasks except AUTOMGR.

Preparing to Use the Advanced Automation Sample Set

If you are already doing automation without the advanced automation sample set, you can incorporate pieces of the advanced automation sample set into your existing automation. To make full use of the automation capabilities provided by the advanced automation sample set, however, you should start with the advanced automation sample set and make any required modifications to the sample set. You need to change your NetView definitions as discussed in the following sections before bringing up NetView with the advanced automation sample set.

Preparing for NetView Initialization

NetView can be started before other subsystems and applications in the system, including JES and VTAM. The advanced automation sample set assumes that NetView is to be initialized by the operating system and that all other applications and subsystems are thereafter initialized by NetView. This, however, is not a requirement for automation.

Starting NetView before JES: JES can be started at the same time as NetView or delayed until NetView automation is active. The advantage of the second approach is that JES messages that occur during initialization can be captured with the automation table, allowing you to detect what was running on the system at the time of failure during JES start-up. Such information can help you decide if the initialization process should continue normally or if some recovery is required; it might become a vital part of your recovery during the initialization process.

If you decide to start NetView before JES, special setup is required. The setup includes:

- The NetView procedure must be started with the START command using the SUB=MSTR operand.

  For example, if you are using the sample procedures supplied with NetView, the following statement should be added as the last statement of the COMMNDxx member of SYS1.PARMLIB:

  ```
  S CNMPSSI,SUB=MSTR
  S CNMPROC,SUB=MSTR
  ```

  The first statement starts the NetView subsystem address space. The second statement starts the NetView application. It does not matter which statement you put first.
- The NetView procedure must be stored in the SYS1.PROCLIB, not in a user library supported by JES.
- The NetView procedure must contain only a single job step.

  Note: You can circumvent the single-job-step restriction if you:

  - Write a user-written driver that invokes the programs from each step via the MVS LINK macro interface.
  - Combine the DD statements from each step into a single group.
  - Specify your program on the EXEC statement for the job.
- All data sets must be referenced by `VOL=SER` or be cataloged in the master catalog.
- No SYSIN, SYSOUT or VIO data set can be referenced.
- The NOSTART parameter must be coded on the JES statement in the IEFSSNxx member of SYS1.PARMLIB to delay the start of JES until NetView is active.
- The JES statement must be coded before the NETVIEW statement in the IEFSSNxx member of SYS1.PARMLIB.
- If you start your NetView program with `SUB=MSTR`, the JES job log is allocated by default when the DSIRQJOB task requests a job ID for the NetView job. If you do not want the JES job log, you can change the JES job log constant in DSICTMOD.
- After DSIRQJOB receives a job ID from JES, if JES ends abnormally or terminates without notifying DSIRQJOB to release the job ID, DSIRQJOB and NetView cannot be terminated before JES becomes active again. If JES ends abnormally or is terminated by a user from the command line, the user can use the NetView MVS Command Revision to circumvent this. Refer to the *IBM Tivoli NetView for z/OS Installation: Getting Started* in the section entitled "Starting NetView before JES" for additional information about how to implement this circumvention.

Starting NetView before VTAM: Use NetView to start VTAM or any other subtask based on specific criteria being met. For example, the order in which VTAM resources are started and the number of resources started might depend on the time at which the IPL of the system occurs. NetView can be used to drive different command lists (based on the time of the IPL) to activate the specified resources in the order requested for that time of day.

If you bring up NetView before VTAM is active, define those NetView tasks that require VTAM to be initially inactive. (VTAM-dependent tasks are identified in the VTAM sample A01APPLS (CNMS0013) used to define the NetView applications.) This is done by coding `INIT=N` on the TASK statement in the CNMSTYLE member. This prevents unnecessary messages from being produced by the tasks while they wait to connect to VTAM. The tasks can then be started by a command list in NetView that is driven when the IST020I (VTAM initialization complete) message is intercepted by the automation table.

Starting NetView before a System Authorization Facility Product: There are unique issues when using a system authorization facility (SAF) product, such as RACF (Resource Access Control Facility), in conjunction with the NetView product. Using a SAF product for any type of security requires the SAF product to be started before NetView, so you should start the SAF product and required SAF classes prior to starting NetView.

Modifying the Advanced Automation Sample Set

AOPIVARS (CNME6400) must be run for the advanced automation sample set to work. It loads an automation table, sets initial values for the global variables to be used in the advanced automation sample set, and activates autotask AUTOMGR. AOPIVARS needs to be customized for your operating environment in the following ways:

- All value updates of global variables for products that you are not automating should be removed. For example, AOPIVARS contains definitions for both JES2 and JES3, and at least one of them should be removed.
- All variable values assigned for the commands to be used to start, stop, and display the status of products should be set to match the commands and names that your operators use at your installation.
- The time periods set in AOPIVARS for different automation processes can be changed to their optimal values for your operating environment. For example, variables set in AOPIVARS tell automation how often to monitor the status of the automation autotasks.

If you are automating JES2, the AOPSPURG command list requires customization to drain all JES2 units and devices.

Defining Autotasks

The advanced automation sample set uses autotasks to run command lists in its automated environment. Supplied with the advanced automation sample set are operator definitions (CNMS6408) that should be added to your existing operator profile definitions located in DSIOPF. There are operator definition entries for each autotask used in the advanced automation sample set: AUTOMGR, AUTOJES, AUTOVTAM, AUTOTSO, AUTOIMS, and AUTOCICS. If your installation is not going to automate one or more of the products supported by the advanced automation sample set, the operator definition does not need to be added to DSIOPF.

Each DSIOPF entry can designate the autotask's operator profile, stored in DSIPRF, which in turn can designate an initial command list for the autotask. Two initial command lists for the autotasks are shipped with the advanced automation sample set:

- AOPIMGIC (CNME6402) is the initial command list for autotask AUTOMGR.
- AOPIGNIC (CNME6403) is the generic initial command list used for the other autotasks.

AUTOMGR is an autotask that is defined and shipped as a sample with the advanced automation sample set. You can use the definitions supplied for AUTOMGR as a model when defining your autotasks. Figure 212 shows the DSIOPF entry for the AUTOMGR that is in the advanced automation sample set sample CNMS6408.

```
*****************************************************************
*    (C) COPYRIGHT IBM CORP. 2010                               *
*    LAST CHANGE:    07/12/10                                   *
*    DESCRIPTION: NETVIEW OPERATOR DEFINITIONS/PASSWORDS        *
*                 FOR AUTOMATED OPERATORS                       *
*    CNMS6408 CHANGED ACTIVITY:                                 *
*    CHANGE CODE  DATE      DESCRIPTION                         *
*    -----------  --------  ------------------------------------*
*****************************************************************
*  THESE OPERATOR DEFINITIONS SHOULD BE ADDED TO YOUR EXISTING  *
*     DSIOPF OPERATOR DEFINITIONS.                              *
*****************************************************************
⋮
AUTOMGR  OPERATOR   PASSWORD=AUTOMGR
         PROFILEN   DSIPROFM
⋮
```

Figure 212. CNMS6408 Excerpt (AUTOMGR Operator Definition)

Figure 213 shows the operator profile for the AUTOMGR autotask that is part of the advanced automation sample set.

```
*********************************************************************
*  (C) COPYRIGHT IBM CORP. 2010                                     *
*     ALL RIGHTS RESERVED.                                          *
*     US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION      *
*     OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT         *
*     WITH IBM CORPORATION.                                         *
*  IEBCOPY SELECT MEMBER=((CNMS6409,DSIPROFM,R))                    *
*  LAST CHANGE:    07/12/10                                         *
*  DESCRIPTION: AUTOMATED OPERATOR (AUTOMGR) PROFILE DEFINITION     *
*  CNMS6409 CHANGED ACTIVITY:                                       *
*  CHANGE CODE  DATE      DESCRIPTION                               *
*  -----------  --------  ----------------------------------------*
*********************************************************************
*  MINIMAL AUTOMATED OPERATOR PROFILE STATEMENTS FOR AUTOMGR        *
*  STARTED WITH AUTOTASK COMMAND TO RUN AS AN UNATTENDED OPERATOR.  *
*  AUTOMGR IS THE AUTOMATED OPERATOR THAT MANAGES THE OTHER         *
*    AUTOMATED OPERATORS USED IN THE CONSOLE AUTOMATION SAMPLE SET. *
*********************************************************************
DSIPROFM     PROFILE  IC=AOPIMGIC
             AUTH     MSGRECVR=NO,CTL=GLOBAL
             END
```

Figure 213. CNMS6409 Excerpt (DSIPROFM Operator Profile)

As long as AUTOMGR is defined in DSIOPF, it is a valid NetView operator ID. When AUTOMGR starts, the AOPIMGIC command list runs, because it is the initial command list defined on the operator profile associated with AUTOMGR.

Defining Command Definition Statements

CNMCMD is the place where you can define command synonyms for your command lists.

The advanced automation sample set includes entries that must be added to your existing CNMCMD for the advanced automation sample set to work. The entries are contained in CNMS6404 of the advanced automation sample set. There is an entry for each advanced automation sample set command list. Because the synonyms defined in those entries are used throughout the advanced automation sample set, the command lists will not work until you have included the entries in your CNMCMD member.

Note: If a command synonym conflicts with any command synonym already defined in your system, one solution is to change the synonym supplied in the advanced automation sample set and in all of the command lists in the advanced automation sample set where it is referenced.

In addition to the required entries described above, three additional sets of command synonyms are provided with the advanced automation sample set:

- CNMS6401 contains command definition samples for MVS command verbs.
- CNMS6402 contains command definition samples for JES2 command verbs.
- CNMS6403 contains command definition samples for JES3 command verbs.

The commands are supplied for your convenience but are not required in order to use the advanced automation sample set.

Modifying the Automation Table

The advanced automation sample set includes automation table entries required for using the advanced automation sample set. You must ensure that the

automation table entries that pertain to the areas you want to automate are included in your production automation table along with automation table entries that are shipped with the product. You can do this as follows:

- If you do not currently have an automation table that you are using in your production environment, you can copy the supplied table entries from DSITBL11 (CNMS6405) into the automation table sample (DSITBL01) that is supplied with NetView, or you can include DSITBL11 in DSITBL01 with a %INCLUDE statement.
- If you are currently using a production automation table, you can copy DSITBL11 (CNMS6405) into it or use a %INCLUDE to include DSITBL11. If this results in message numbers being duplicated in the table, the NetView automation table entries for those messages should be combined into one statement. If they are not, only the first of the duplicate statements is triggered when the message is processed, unless you use a CONTINUE action on the statement.
- If the automation table you are to use for automation has a name other than DSITBL11, then update command list AOPIVARS (CNME6400) load the new table name instead of DSITBL11.
- If you are not automating a given product (for example, TSO), make sure you do not copy the DSITBL11 (CNMS6405) entries for the product into your production automation table. If you do, operators who try to start the task manually will not receive the messages associated with the command at their terminals.
- If you change the automation table while NetView is running, you must recycle the table by using the `AUTOTBL MEMBER=`*`automem`* command, where *automem* is the name of the member containing the automation table.

Several automation tables can be defined. You might want to have separate automation tables for when you are running the advanced automation sample set and when you are not. The name of the automation table loaded in command list AOPIVARS must match the name of the table with the advanced automation sample set entries.

Customizing the Advanced Automation Sample Set

If you are using the advanced automation sample set to help you develop automation, the advanced automation sample set makes it easy to customize the samples for your operating environment. You can add functions to support an additional product, operating system, command list, or automation table statement. Once you understand the basic way the advanced automation sample set is designed, you can make any changes required by your site. The following sections suggest ways you can add to the advanced automation sample set.

Customizing with Global Variables

Any additional functions you add to the sample set should use the same global-variable conventions that the sample set uses. Information concerning each product being automated is shared between command lists using task and common global variables. Each product has variables associated with it and containing information required to automate it, such as the command that starts the product and the command that stops the product. The variables are initialized during automation initialization (command list AOPIVARS), and command lists performing any function (initialization, recovery, monitoring, utility, shutdown) can then access the variables needed for a specific product.

This section describes how complex global variables are built from common global variables and provides an example of using a complex global variable.

Building and Naming Complex Global Variables: Some of the global variables used are built from a composite of several other values. For example, a common global variable containing the command to start a product requires a resource common prefix (RCP) to identify to the system the command is intended to automate, a value to signify the variable type (START), and an indication of the product the variable refers to. To make customizing easier, a pattern is followed when building complex global variables.

The system being automated is saved in a resource common prefix (RCP) global variable. RCP (REXX) or &RCP (NetView command list language) is the variable that identifies the system that NetView is running on. The value of the RCP or &RCP variable can be up to 5 characters long and is automatically set in AOPIVARS to the domain ID of the system on which AOPIVARS is running.

The function for which the variable provides information is indicated using 3 characters. For example, CST means that the variable provides START information for the product. The function types used in the advanced automation sample set are:

CST START command for product
CSP STOP command for product
CSA Current STATUS of product
CDS Desired STATE of product
CDP Command to DISPLAY status of product
CT1 Last time product went active
CT2 Last time product went inactive
STC START command list
SPC STOP command list
PDD Product dependency
WTI Number of seconds to wait for complete initialization
WTS Number of seconds to wait for complete shutdown
CTI Last time status checked by active monitor
CNA Name of the product
CO*n* Message response (*n* can be any number)

The product identifier is a 3-character variable and is also used when building complex global variables. The product identifiers used in the advanced automation sample set are listed in Table 37.

Table 37. Product Identifiers in the Advanced Automation Sample Set

Product	REXX Variable	NetView Command List Language Variable	Value
JES2 or JES3	JES	&JES	JES
VTAM	VTAM	&VTAM	VTM
TSO	TSO	&TSO	TSO
CICS	CICS	&CICS	CCS
IMS	IMS	&IMS	IMS

The main automation initialization command list is AOPIVARS. AOPIVARS builds the global variables that contain the information for each product. AOPIVARS builds its common global variables by invoking command list AOPIGUPD, whose purpose is to build a global variable and set the value of it based on the input to

the command list. In AOPIVARS, the resource common prefix for the automated system and the product identifiers for all automated products are set up in common global variables.

Example of Using a Complex Global Variable: This section is a complete step-by-step example of how a complex global variable is built and used in automation. The example uses NetView command list language variables, but the building process is the same in REXX. The variable that stores the command to start TSO is used. Assume a system identifier (&RCP) of CNM01. The system identifier for your system can be different. Because you are building a start variable, the function identifier is CST. The product variable, a 3-character product identification, is &TSO, which has a value of TSO.

You might use the information in Figure 214 to define &RCP and &TSO.

```
&CGLOBAL RCP TSO
&RCP = CNM01
&TSO = TSO
```

Figure 214. Defining Variables for the Start TSO Variable

Now, you can build the start TSO variable and set the value, as shown in Figure 215.

```
&START = &CONCAT &RCP CST&TSO
```

Figure 215. Building a Start TSO Variable

The NetView command list language &CONCAT function concatenates the values &RCP and CST&TSO The value of variable &TSO is substituted and combined with CST. In the example, &START is given the value CNM01CSTTSO, a value derived in the following way:

&START = Resource Common Prefix + Function + Product
&START = &RCP + CST + &TSO
&START = CNM01 + CST + TSO
&START = CNM01CSTTSO

When the command list runs, the value of variable &START is CNM01CSTTSO. &START can now be defined as a common global variable with the statement shown in Figure 216.

```
&CGLOBAL &START
```

Figure 216. Statement Defining &START as a Common Global Variable

Defining &START as a common global actually causes a substitution on the &START variable to be performed before the variable is defined as a common global, because the &CGLOBAL function requires no ampersand on the variable being defined. In the statement to define RCP as a common global variable, no ampersand precedes the RCP. &CGLOBAL RCP defines the variable &RCP as a common global. So, in the above example, CNM01CSTTSO is substituted for &START in the &CGLOBAL statement, so that CNM01CSTTSO is defined as a common global variable.

You can now give a value to the variable by directly updating the value, as in &CNM01CSTTSO = 'S TSO', but doing so is inconvenient, because each system

identifier, function, and product must be remembered for each variable. Instead, you can update the variable indirectly. &START is already defined as CNM01CSTTSO. Use it to change the value of the variable, as shown in Figure 217.

```
&&START = 'S TSO'
```

Figure 217. Updating a Common Global Variable Indirectly

The value of &START is to be substituted into the statement when the command list is processed, forming the actual assignment shown in Figure 218.

```
&CNM01CSTTSO = 'S TSO'
```

Figure 218. Substituting a Common Global Variable in an Assignment

AOPIGUPD is called to build the common global variables for the advanced automation sample set. The process that AOPIGUPD goes through is similar to the method just described.

Using that method of sharing common variable values provides an easy way to share information between automation command lists. If you want to add a new system identifier, function, or product to the advanced automation sample set, use the same global variable convention to add the new item.

Fine-Tuning the Advanced Automation Sample Set

If you decide to use the advanced automation sample set as a guide for your automation, you might be able to make changes that eliminate unnecessary overhead in your production environment. Messages that cannot be issued in your system environment should be removed from the automation table. If a product that is supported by the advanced automation sample set is not available at your site, then the production copies of any affected command lists and samples can be changed to no longer handle that product. Copies should be kept of any command lists, samples, and panels that are to be changed so they can be referred to later if needed. Any command lists and panels that are now unused because of removed automation table entries or changes to advanced automation sample set command lists can be removed from the production automation system to conserve DASD and minimize confusion. Again, be sure to keep a copy of any deleted command lists as backup in case they are needed in the future.

Adding a Product: Before adding a new product to be handled by automation, the scope of what needs to be handled must first be identified. Do you want automation to handle only initialization of the new product, or do you want automation to handle the new product as completely as possible (initialization, proactive monitoring, recovery, shutdown)?

Once the scope is identified, the actual messages that are to be handled need to be identified.

- Try to keep the number of messages small.

  For example, if a product has a number of failure messages but each is always followed by the same message indicating that the product cannot continue, then automate only that message.
- Each message being added might require a change to the automation table.
- If the message is a system message, ensure that the message gets passed to the NetView program through the operating system message processing facility.

- If a command list needs to be driven for status changes of the new item, then the command list must be written, or perhaps an existing command list can be changed to act correctly on receipt of the message.

Once the scope is identified, changes to existing samples are required and new command lists might have to be created.

- Adding an automated product requires changes to the automation table and additional changes to command lists, depending on the scope of what you want automation to handle.
- In all cases, the additions should follow the same pattern used by the advanced automation sample set, making the additions easy to define.

Below is an outline of possible command lists and samples that need to be updated to add full automation capabilities for an additional product:

AOPIVARS	Set global variables for the new product
DSIOPF	Add new automated operator for the product
DSITBL11	Add automation table entries
AOPInew	Create new initialization command list for the product
AOPSnew	Create new shutdown command list for the product
new clists	If required, add other new command lists, called through the automation table
MPF	Ensure that system messages are forwarded to NetView

The generic operator profile DSIPROFG (CNMS6410), generic autotask initial command list AOPIGNIC (CNME6403), main shutdown command list AOPSMAIN (CNME6412), and automation-status-display command list AOPUSTAT (CNME6438) with associated panels can all be used for additional automated products. New command lists might need to be written to handle the start-up and shutdown of the new item. New command lists might also need to be created to handle recovery of the new item when a message or messages drive the automation table.

Handling a New Message with Automation: Handling a new message with automation requires first that you understand exactly what you wish automation to do upon receipt of the new message. If you want to suppress the message, suppress it with the operating system's message processing facility. If you want to route the message with automation or to start a command list, then a change to the automation table is necessary. In addition, the operating system's message processing facility might have to be updated to forward system messages to NetView. If an automation table statement for another message already accomplishes what you want the statement for your new message to accomplish, then consider simply adding your new message to the existing automation-table statement. If you do need a new automation-table statement, place the statement in an appropriate place in your table for efficient processing. For example, if you are using BEGIN-END sections, place the new statement in an appropriate section.

Changing Timer-Command Intervals: Throughout the advanced automation sample set, timer commands call command lists or issue commands according to certain time intervals. By editing AOPIVARS, you can change the time intervals to suit your own environment. For example, the autotask monitor checks each autotask every two minutes to ensure that it is active. You might not need to check the autotasks that often, or might want to check them more often. The

active-monitor command list checks the status of all automated products every three minutes. You might want the active monitor to check more or less often depending on your operating environment.

Preloading Command Lists: The LOADCL and DROPCL commands can be used to load commonly processed command lists into main storage and drop them. Loading command lists into storage greatly reduces their processing time by eliminating the need to load the command list each time it runs. The advanced automation sample set command list AOPIGUPD is preloaded by default in command list AOPIVARS because of the number of times AOPIVARS invokes it.

Testing Added or Changed Automation

When making any changes to the advanced automation sample set, remember that, because actions are taken in an automated environment, it is best to test in a controlled test environment before using as production automation. You can test the automation manually by tracing processing of a manually called command list. For example, use &CONTROL ALL if you are running the NetView command list language portions of the command lists, or TRACE if you are using REXX. The processing tracing can be removed once the command list is tested. Messages to the network log or to an operator console can be added in command lists and then removed once proven correct. Automation capabilities can be tested by starting command lists directly as if they had been called from the automation table.

Cross-Reference Listing of Command Lists and Samples

Following is a list of the samples and command lists that are shipped as part of the sample set for automation. Where no sample name is given in the table, the renaming JCL (CNMS62J1) does not provide a new name for that sample, and the shipped name continues to be used.

Basic Automation Sample Set

Samples

Table 38. Samples in the Basic Automation Sample Set

Shipped Name	Sample Name	Description
CNMS6205	ACOTABLE	Automation-table entries
CNMS6206	CNMCMD	CNMCMD CMDDEF entries
CNMS6211	CLRLOG	Clears SYS1.LOGREC for future recording
CNMS6212	CLRSMF	Clears SYS1.MANX or SYS1.MANY for future recording
CNMS6213	LGPRNT	Prints SYS1.LOGREC
CNMS6214	DSIPRT	Prints the primary or secondary network logs or both
CNMS6221	$CLRSMF	Input to the CLRSMF procedure
CNMS6222	$SOFT	Input to the LGPRNT procedure, step name SOFT
CNMS6223	$SYSEXN	Input to the LGPRNT procedure, step name SYSSEXN
CNMS6224	$SYSUM	Input to the LGPRNT procedure, step name SYSUM

Command Lists

Table 39. Command Lists in the Basic Automation Sample Set

Shipped Name	Command Synonym Name	Description
CNME6201	AUTO1IC	Initial command list for AUTO1 autotask. Starts $DSPOOL command list every 24 hours
CNME6202	$DSPOOL	Resets global parameters Z$DSPOOL and Z$DSPLHRS to normal
CNME6203	$DSPOOL2	Sets global parameters Z$DSPOOL and Z$DSPLHRS to drain more spool space if necessary
CNME6204	$HASP646	Control calling $DSPOOL2 based on utilized spool space
CNME6205	IEE362A	Control calling CLRSMF procedure to print SMF file after it is closed

Advanced Automation Sample Set

Samples

Table 40. Samples in the Advanced Automation Sample Set

Shipped Name	Sample Name	Description
CNMS64P0		Panel to display automation status of all products
CNMS64P1		Panel to display automation information for a specific product
CNMS64P2		Panel to display message response variable values
CNMS64P3		Help panel for panel CNMS64P0
CNMS64P4		Help panel for panel CNMS64P1
CNMS64P5		Help panel for panel CNMS64P2
CNMS6401	CNMCMD	CMDDEF statements for MVS commands
CNMS6402	CNMCMD	CMDDEF statements for JES2 commands
CNMS6403	CNMCMD	CMDDEF statements for JES3 commands
CNMS6404	CNMCMD	CMDDEF statements for advanced automation sample set command lists
CNMS6405	DSITBL11	NetView automation table entries required by the advanced automation sample set
CNMS6406	AOPUMCMT	TSO command list to copy command lists written in the NetView command list language without comments
CNMS6408	DSIOPF	Automated operator definitions
CNMS6409	DSIPROFM	AUTOMGR autotask operator profile
CNMS6410	DSIPROFG	Generic autotask operator profile

Command Lists Sorted by Shipped Name

Table 41. Command Lists in the Advanced Automation Sample Set by Shipped Name

Shipped Name	Command Synonym Name	Description
CNME6400	AOPIVARS	Automation initialization main command list
CNME6401	AOPIGUPD	Sets value of a common global variable
CNME6402	AOPIMGIC	AUTOMGR initial command list
CNME6403	AOPIGNIC	Generic initial command list for autotasks
CNME6404	AOPIJES3	Starts JES3
CNME6405	AOPIJES2	Starts JES2
CNME6406	AOPIVTAM	Starts VTAM
CNME6407	AOPIIMS	Starts IMS
CNME6408	AOPICICS	Starts CICS
CNME6409	AOPITSO	Starts TSO
CNME6410	$HASP426	Responds to $HASP426 SPECIFY OPTIONS
CNME6412	AOPSMAIN	Main shutdown command list
CNME6413	AOPSJES3	Shuts down JES3
CNME6414	AOPSJES2	Shuts down JES2
CNME6415	AOPSVTAM	Stops VTAM
CNME6416	AOPSIMS	Begins IMS shutdown
CNME6417	AOPSIMS2	Stops IMS components
CNME6418	AOPSCICS	Stops CICS
CNME6419	AOPSTSO	Begins TSO shutdown
CNME6420	AOPSTSO2	Stops TSO
CNME6421	AOPTJRC3	Resets reply count for IAT3714
CNME6422	$HASP098	Replies to message $HASP098
CNME6423	$HASP095	Stores JES2 abend code
CNME6424	AOPSPURG	Drains all units
CNME6425	VTAMTMRZ	Shuts down VTAM with cancel
CNME6426	DFS996I	Stores IMS WTOR
CNME6428	DFS000IB	Stores IMS region numbers for shutdown
CNME6429	IAT3714	Issues reply for JES3 start type
CNME6430	IAT3708	Updates JES3 status to active
CNME6431	$HASP085	Attempts to restart JES2 if it ended abnormally
CNME6432	JESTMRA	Resets JES2 abend counter
CNME6433	DFS629I	Restarts IMS
CNME6434	IMSTMR	Updates IMS timer to blank
CNME6435	DFH0606	Restarts CICS
CNME6436	CICSTMRA	Updates CICS timer to blank
CNME6437	IKT002I	Updates status of TSO to abend
CNME6438	AOPUSTAT	Displays panels containing automation status information

Table 41. Command Lists in the Advanced Automation Sample Set by Shipped Name (continued)

Shipped Name	Command Synonym Name	Description
CNME6439	AOPMACT	Actively monitors automated products
CNME6440	AOPMCHEK	Monitors advanced automation sample set autotasks

Command Lists Sorted by Command Synonym Name

Table 42. Command Lists in the Advanced Automation Sample Set by Synonym

Shipped Name	Command Synonym Name	Description
CNME6431	$HASP085	Attempts to restart JES2 if it abended abnormally
CNME6423	$HASP095	Stores JES2 abend code
CNME6422	$HASP098	Replies to message $HASP098
CNME6410	$HASP426	Responds to $HASP426 SPECIFY OPTIONS
CNME6408	AOPICICS	Starts CICS
CNME6403	AOPIGNIC	Generic initial command list for autotasks
CNME6401	AOPIGUPD	Sets value of a common global variable
CNME6407	AOPIIMS	Starts IMS
CNME6405	AOPIJES2	Starts JES2
CNME6404	AOPIJES3	Starts JES3
CNME6402	AOPIMGIC	AUTOMGR initial command list
CNME6409	AOPITSO	Starts TSO
CNME6400	AOPIVARS	Automation initialization main command list
CNME6406	AOPIVTAM	Starts VTAM
CNME6439	AOPMACT	Actively monitors automated products
CNME6440	AOPMCHEK	Monitors advanced automation sample set autotasks
CNME6418	AOPSCICS	Stops CICS
CNME6416	AOPSIMS	Begins IMS shutdown
CNME6417	AOPSIMS2	Stops IMS components
CNME6414	AOPSJES2	Shuts down JES2
CNME6413	AOPSJES3	Shuts down JES3
CNME6412	AOPSMAIN	Main shutdown command list
CNME6424	AOPSPURG	Drains all units
CNME6419	AOPSTSO	Begins TSO shutdown
CNME6420	AOPSTSO2	Stops TSO
CNME6415	AOPSVTAM	Stops VTAM
CNME6421	AOPTJRC3	Resets reply count for IAT3714
CNME6438	AOPUSTAT	Displays panels containing automation status information
CNME6436	CICSTMRA	Updates CICS timer to blank
CNME6435	DFH0606	Restarts CICS
CNME6428	DFS000IB	Stores IMS region numbers for shutdown

Table 42. Command Lists in the Advanced Automation Sample Set by Synonym (continued)

Shipped Name	Command Synonym Name	Description
CNME6433	DFS629I	Restarts IMS
CNME6426	DFS996I	Stores IMS WTOR
CNME6430	IAT3708	Updates JES3 status to active
CNME6429	IAT3714	Issues reply for JES3 start type
CNME6437	IKT002I	Updates status of TSO to abend
CNME6434	IMSTMR	Updates IMS timer to blank
CNME6432	JESTMRA	Resets JES2 abend counter
CNME6425	VTAMTMRZ	Shuts down VTAM with cancel

Message Suppression Samples

Table 43. Message Suppression Samples

Shipped Name	Sample Name	Description
CNMS6201	MPFLSTAC	Conservative MVS MPF message suppression
CNMS6202	MPFLSTAA	Aggressive MVS MPF message suppression

Log Analysis Samples

Table 44. Log Analysis Samples

Shipped Name	Sample Name	Description
CNMS6207		JES2 and JES3 log analysis program
CNMS62J2		Runs log analysis program

Setup Samples

Table 45. Setup Samples

Shipped Name	Sample Name	Description
CNMS62J1		Renaming JCL

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Programming Interfaces

This publication primarily documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Tivoli NetView for z/OS. This publication also documents information that is NOT intended to be used as Programming Interfaces of Tivoli NetView for z/OS. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```
NOT Programming Interface information
End of NOT Programming Interface information
```

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml .

Adobe is a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Product Number: 5697-NV6

Printed in USA

SC27-2846-01