Tivoli NetView for z/OS Customization Guide

Copyright Notice

© Copyright IBM Corporation 1997, 2002. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement, an IBM Software License Agreement, or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of IBM Corporation. IBM Corporation grants you limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry the IBM Corporation copyright notice. No other rights under copyright are granted without prior written permission of IBM Corporation. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed, including the warranties of merchantability and fitness for a particular purpose.

U.S. Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corporation.

Trademarks

IBM, the IBM logo, Tivoli, the Tivoli logo, AIX, AD/Cycle, APPN, C/370, CICS, DB2, IMS, KnowledgeTool, Language Environment, LPDA, MVS/ESA, NetFinity, NetView, OPC, OS/2, OS/390, RACF, RS/6000, Tivoli ADSM, Tivoli Enterprise Console, Tivoli Management Framework, Tivoli OPC, Tivoli Plus, Tivoli Professional Services, Tivoli Ready, TME 10, VM/ESA, VSE/ESA, VTAM, z/OS, are trademarks or registered trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the United States, other countries, or both.

Lotus is a registered trademark of Lotus Development Corporation.

Microsoft, Windows, Windows NT, and the Windows logo 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.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Notices

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to valid intellectual property or other legally protectable right of Tivoli Systems or IBM, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user. Tivoli Systems or IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York 10504-1785, U.S.A.

Programming Interfaces

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

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

Chapter 1. Designing Functions ......................................... 1
  Customization Areas ........................................... 1
    Functions to Consider before Making Modifications ......... 3
    Finding Customization Information . 3
    Collecting Data ........................................... 4
    Data Storage and Recording ................................ 6
    Operator Presentation .................................... 6
  Tasks ........................................... 7
    NetView Program as a System Application Program ....... 7
    NetView Program Tasks ................................ 7
    Program Activity within a Task ................................... 8
    Queuing Work to NetView Program Tasks ......... 9
    Message and Command Buffers ................................ 9
    Immediate Commands .................................... 9
    Long-Running Commands .................................. 9
    Data Services Commands ................................ 10
  Defining User-Written Programs on the Host: Exits and Commands ......... 10
    Installation Exit Programs . 10
    Command Processors and Command Lists ................................... 11
  Adding Optional Tasks to the NetView Program . 12
  Choosing a Language .................................... 12
    Input and Output .................................. 12
    Performance ..................................... 13
    Stability ..................................... 13
    Testing ..................................... 13
    Speed of Implementation ................................ 13
    REXX Versus the NetView Command List Language .......... 13
    Language Choices by Function . 14
    Logging .................................... 15
    Cross-Reference for Message and Environment Functions ......... 15
  Customizing PF Keys and Immediate Message Line ......... 23
# Chapter 2. Customizing the NetView Command Facility Panel

- Modifying CNMKEYS ........................................ 24
- Chapter 2. Customizing the NetView Command Facility Panel ...................... 25
  - Using a Screen Format Definition ........................................ 25
  - Screen Format Definition Statements ................................... 25
  - Message Color and Highlighting .......................................... 28

# Chapter 3. Using the VIEW Command

- Chapter 3. Using the VIEW Command .................................... 29
  - Creating Full-Screen Panels .............................................. 29
  - General Help Fields ..................................................... 30
  - Coding the VIEW Command .............................................. 32
  - Return Codes from VIEW and BROWSE ................................ 33
  - Displaying VIEW Return Codes with SHOWCODE ....................... 34
  - Controlling Color and Highlighting of Fields ......................... 35
  - Displaying Variables in Source Panels ................................ 40
  - Compound Symbols ..................................................... 42
  - Issuing Commands from Command Procedures .......................... 43
  - Creating a Rollable Component with VIEW ................................ 44
  - Full-Screen Input Capabilities .......................................... 47
  - Returning Command Line Input ........................................ 54
  - Using PF Keys and Subcommands with VIEW ............................ 55
    - Using PF Keys and Subcommands with the NOINPUT Option ........... 55
    - Using PF Keys and Subcommands with the INPUT Option ............. 55
  - Dynamic Update Capabilities ........................................... 56
  - Sample of Panel Updating ................................................ 57
  - Changing Colors in Browse ............................................. 59

# Chapter 4. Modifying and Creating Online Help Information

- Chapter 4. Modifying and Creating Online Help Information ............... 63
  - Locating Help Source Files ............................................ 63
  - View-Based Help ....................................................... 64
  - Window-Based Help ..................................................... 64
  - Copying and Changing Help Source Files ................................ 67
  - Storing Help Source Files ............................................. 68
  - HELPACT Facility ...................................................... 68
  - Displaying New Help Panels ............................................ 69

# Chapter 5. Customizing Session Monitor Sense Descriptions

- Chapter 5. Customizing Session Monitor Sense Descriptions ............... 71
  - Session Monitor Sense Codes ........................................... 71
  - Examples ............................................................... 72

# Chapter 6. Customizing Hardware Monitor Displayed Data

- Chapter 6. Customizing Hardware Monitor Displayed Data .................... 75
  - Modifying Hardware Monitor Nongeneric Panels .......................... 75
    - Determining a Panel Name .............................................. 75
    - Changing Panel Text .................................................. 78
  - Nongeneric Alert Messages .............................................. 80
  - Using the ACTION Command List ....................................... 81
  - Overlaying Recommended Action Numbers .................................. 81
    - Modifying BNJDNUMB, BNJDNAME, and BNJwwwww .................... 82
  - Changing Color and Highlighting for Hardware Monitor Panels ......... 85
    - Selecting the Color Map .............................................. 86
    - Modifying the Color Map ............................................. 86
    - Prompt Highlight Tokens .............................................. 89
  - Using NMVT Support for User-Written Programming ....................... 90
    - User-Defined Alerts (Nongeneric) .................................. 90
    - User-Defined Alerts (Generic) ...................................... 91
    - Building Generic Alert Panels ...................................... 92
    - Alerts-Dynamic Panel ................................................ 94
    - Recommended Action for Selected Event Panel ....................... 95
    - Event Detail Panel .................................................. 97
    - Modifying Generic Code Point Tables ................................ 99
Chapter 11. Designing HTML Files for the NetView Web Server .......................... 171
  Referencing Files and Commands .......................................................... 171
  Understanding the Base URL ...................................................................... 171
  Adding Links to the Portfolio ..................................................................... 172
  Loading Panels ......................................................................................... 173
  History Log ................................................................................................. 173
  Using REXX to Generate HTML .............................................................. 174
  Security Token ............................................................................................ 174
  Migrating Web Browser Support .............................................................. 174

Appendix A. Color Maps for Hardware Monitor Panels .................................. 177

Appendix B. NetView Macros and Control Blocks ........................................... 181
  General-Use Programming Interface Control Blocks and Include Files ............ 181
  Product-Sensitive Programming Interfaces .................................................. 185

Index ............................................................................................................. 187
Preface

This document describes the parts of the NetView® program that you can customize and points you to sources of related information.

Who Should Read This Document

This document is intended for system programmers who customize the NetView program.

What This Document Contains

This document is organized into the following sections, which address the modification and development of your own programming enhancements.

- **Chapter 1. Designing Functions** on page 1 describes what you need to know before making an addition or change to the NetView program. This chapter also lists facilities and documentation available to help you customize tasks.

- **Chapter 2. Customizing the NetView Command Facility Panel** on page 25 describes how to customize your NetView command facility panel. Customizable functions include: color of fields on the panel; color of command area; default colors for held, action, normal and immediate messages; foreground color for messages. Some other customizable functions are: how much information precedes the message text and how much screen is set aside for held and action messages.

- **Chapter 3. Using the VIEW Command** on page 29 describes the VIEW command processor used to display full-screen panels from user-written programs.

- **Chapter 4. Modifying and Creating Online Help Information** on page 63 describes how to change the content of the online help facility panels in the NetView program.

- **Chapter 5. Customizing Session Monitor Sense Descriptions** on page 71 describes how to modify DSIPARM data set members, how to include additional members, and how to include help for sense codes that have additional meaning for a specific application.

- **Chapter 6. Customizing Hardware Monitor Displayed Data** on page 75 describes how to modify the presentation of generic and nongeneric alerts.

- **Chapter 7. Modifying Network Asset Management Command Lists** on page 103 describes how to customize non-SNA commands and tables.

- **Chapter 8. Customizing the Event/Automation Service** on page 111 describes how to customize the event/automation service for multiple platforms, including workstations and z/OS™ environments.

- **Chapter 9. NetView Instrumentation** on page 159 describes how to customize NetView instrumentation, including the topology display subsystem for the NetView management console (NMC).
Appendix A. Color Maps for Hardware Monitor Panels on page 177 contains a table that lists the panel name, panel number, and color map for hardware monitor panels.

Appendix B. NetView Macros and Control Blocks on page 181 contains a list of NetView macros, control blocks and include files.

Publications

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

Prerequisite and Related Documents

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

You can find additional product information on these Internet sites:

<table>
<thead>
<tr>
<th>Table 1. Resource Web sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM®</td>
</tr>
<tr>
<td>Tivoli Systems</td>
</tr>
<tr>
<td>Tivoli NetView for z/OS</td>
</tr>
</tbody>
</table>

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

- Getting statistics for your automation table and merging the statistics with a listing of the automation table
- Displaying the status of a JES job or cancelling a specified JES job
- Sending alerts to the NetView program using the program-to-program interface (PPI)
- Sending and receiving MVS™ commands using the PPI
- Sending TSO commands and receiving responses

Accessing Publications Online

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

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

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

Ordering Publications

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

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

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

- In the United States: 800-879-2755
• In Canada: 800-426-4968
• In other countries, for a list of telephone numbers, see the following Web site:
  [http://www.tivoli.com/inside/store/lit_order.html]

Providing Feedback about Publications
We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:
• Send an e-mail to pubs@tivoli.com.
• Complete our customer feedback survey at the following Web site:
  [http://www.tivoli.com/support/survey/]

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

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

The handbook provides information about how to contact Tivoli Customer Support, depending on the severity of your problem, and the following information:
• Registration and eligibility
• Telephone numbers and e-mail addresses, depending on the country you are in
• What information you should gather before contacting support

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

[http://www.tivoli.com/nv390]

Under Related Documents, select Other Online Sources.

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

Accessibility Information
Refer to Tivoli NetView for z/OS User’s Guide for information about accessibility.

Keyboard Access
Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

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

Conventions Used in This Document
The document uses several typeface conventions for special terms and actions. These conventions have the following meaning:
Commands, keywords, flags, and other information that you must use literally appear like this, in **bold**.

Variables and new terms appear like this, in *italics*. Words and phrases that are emphasized also appear like this, in *italics*.

Code examples, output, and system messages appear like this, in a monospace font.

Tivoli NetView for z/OS commands are in **ALL CAPS**.

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

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

RACF®

RACF is a component of the SecureWay® Security Server for z/OS and OS/390, providing the functions of authentication and access control for OS/390 and z/OS resources and data, including the ability to control access to DB2® objects using RACF profiles. Refer to: [http://www-1.ibm.com/servers/eserver/zseries/zos/security/racfss.html](http://www-1.ibm.com/servers/eserver/zseries/zos/security/racfss.html)

Tivoli Enterprise™ software

Tivoli software that manages large business networks.

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

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

Specifies the version and release.

VTAM® and TCP/IP

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

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

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

Reading Syntax Diagrams

Syntax diagrams start with double arrowheads on the left (➚) and move along the main line until they end with two arrowheads facing each other (➘). As shown in the following table, syntax diagrams use position to indicate the required, optional, and default values for keywords, variables, and operands.

<table>
<thead>
<tr>
<th>Element Position</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the command line</td>
<td>Required</td>
</tr>
<tr>
<td>Above the command line</td>
<td>Default</td>
</tr>
<tr>
<td>Below the command line</td>
<td>Optional</td>
</tr>
</tbody>
</table>

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

Required Syntax

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

CCPLOADF

Figure 1. Required Syntax Elements

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

TRANSMMSG

Figure 2. Syntax for Variables
Optional Keywords and Variables

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

DISPREG

Figure 3. Optional Syntax Elements

Default Values

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

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

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

RID

Figure 4. Sample of Defaults Syntax

Long Syntax Diagrams

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

Syntax Fragments

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

Required commas and parentheses are included in the syntax diagram. When an operand has more than one value, the values are typically enclosed in parentheses and separated by commas. In Figure 6 on page xiv, the OP operand, for example, contains commas to indicate that you can specify multiple values for the testop variable.
If a command requires positional commas to separate keywords and variables, the commas are shown before the keyword or variable, as in Figure 4 on page xii. For example, to specify the BOSESS command with the\n*sessid* variable, enter:
NCCF BOSESS applid,,sessid
You do not need to specify the trailing positional commas. Positional and non-positional trailing commas either are ignored or cause the command to be rejected. Restrictions for each command state whether trailing commas cause the command to be rejected.

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

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

**Table 3. Syntax Elements Examples**

<table>
<thead>
<tr>
<th>This element...</th>
<th>Looks like this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>CCPLOADF</td>
</tr>
<tr>
<td>Variable</td>
<td>resname</td>
</tr>
<tr>
<td>Operand</td>
<td>MEMBER=membername</td>
</tr>
<tr>
<td>Default</td>
<td>today or INCL</td>
</tr>
</tbody>
</table>

*Figure 6. Sample Syntax Diagram with Commas*

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

For example, to specify the BOSESS command with the *sessid* variable, enter:
NCCF BOSESS applid,,sessid
You do not need to specify the trailing positional commas. Positional and non-positional trailing commas either are ignored or cause the command to be rejected. Restrictions for each command state whether trailing commas cause the command to be rejected.
Abbreviations

Command and keyword abbreviations are described in synonym tables after each command description.
Chapter 1. Designing Functions

NetView enables you to manage complex, multivendor networks and systems from a single point. This chapter describes what you need to know before making an addition or change to the NetView program, and shows some of the facilities available to help you customize tasks.

Customization Areas

Customizing NetView takes place at various stages of network and system implementation. These topics are described in several NetView books. See Table 3 on page 3 for the NetView books that contain more information on the listed topics.

Alias names are used to communicate across networks. You can use alias names to resolve conflicts when duplicate resource names exist in multiple networks. With alias names, the name of the resource, such as a logical unit (LU), a class of service, a source LU (SRCLU), or a LOGON mode table from the sending network, is translated to a name that is unique to the receiving network. Refer to Tivoli NetView for z/OS Installation: Getting Started for more information about how to define alias names.

Filtering controls the amount of data presented to operators. Filtering also controls the amount of data recorded in the network log. The NetView automation table allows you to control the types of messages that each of your network operators receives, and the amount of data recorded to message logs. Refer to the Tivoli NetView for z/OS Automation Guide for descriptions of automation statements and descriptions of how to use automation statements to suppress (filter) messages.

You can also filter event data that network resources send to the hardware monitor. Recording filters control the information that is recorded in the hardware monitor’s database. Viewing filters determine the records that appear on each network operator’s terminal. You can find more information on hardware monitor filtering by referring to Tivoli NetView for z/OS User’s Guide or Tivoli NetView for z/OS Automation Guide for a description of how to use automation statements to set recording filters for specific events. Also refer to the NetView online help for the SRF and SVF commands.

Focal point support enables NetView to be defined as either a focal point node or a distributed entry point node. A focal point is a central network node that receives information from distributed entry point network nodes. The information forwarded from the entry points to the focal point can be messages, alerts, or MSUs. For more information on NetView focal point support, refer to the Tivoli NetView for z/OS Automation Guide.

You can use automation to implement automatic responses to events that occur in your network. Refer to the Tivoli NetView for z/OS Automation Guide for a more information about defining NetView automation statements to improve the productivity of your system and network operators. For additional information the NetView program’s automation, refer to the Tivoli NetView for z/OS Automation Guide.
Use **Generic alerts** and **code points** to obtain problem determination support for devices and applications in your network that the NetView program does not automatically support. Chapter 6, Customizing Hardware Monitor Displayed Data, contains information on how to use the NetView-provided and user-defined code point tables to build hardware monitor Alerts-Dynamic, Alerts-Static, Alerts-History, Event Detail, and Most Recent Events panels.

**National Language Support (NLS)** allows your operators to interact with the NetView program in a language other than English. Refer to Tivoli NetView for z/OS Installation: Configuring Additional Components for a description of how to write your own message translations in any other supported language. The Japanese National Language version provides a Japanese version of NetView panels and messages.

You might need to consider **operator control and security**. To control who can gain access to the NetView program and what effect an operator can have on your network, you should consider some level of logon verification, command authorization, and span of control. Refer to the Tivoli NetView for z/OS Security Reference for a complete description of how to implement the different levels of security verification available in the NetView program, how to limit the commands an operator can issue (command authorization), and which part of the network’s resources an operator can control (span of control).

You can modify the color and format of the NetView command facility panel. Refer to Chapter 2, Customizing the NetView Command Facility Panel, for more information.

You can create or change panels for your online help, online message help, NetView help desk, the hardware monitor, and any user-written, full-screen applications. For a detailed explanation of how to create new panels or modify Tivoli-supplied panels for these components, see Chapter 4, Modifying and Creating Online Help Information, or Chapter 6, Customizing Hardware Monitor Displayed Data.

**Sequential logging** (sequential access method log support) enables you to write variable length records to multiple user-defined logs. You can browse or print these logs using your operating system facilities. For more information about defining sequential log tasks, refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components, Tivoli NetView for z/OS Customization: Using Assembler, or Tivoli NetView for z/OS Customization: Using PL/I and C.

**Session monitor data** can be collected and kept in the session monitor database. To control how much session data is collected and kept, customize several session monitor definition statements. Refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components for more information. Defining performance classes for the response time monitor (RTM) feature is also described in Tivoli NetView for z/OS Installation: Configuring Additional Components. Objectives and boundaries are set for each performance class, and a performance class is then chosen for a session.

**User-written functions** add new function to the NetView program or modify existing ones. You might want to develop your own command lists and user-written code. Refer to the Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for an overview of writing command lists in REXX or in NetView command list language to help you control your network and make the operators’ jobs easier. You can find information about writing code...
such as command procedures and installation exits in Tivoli NetView for z/OS Customization: Using PL/I and C. Information on writing command processors, installation exit routines, and user subtasks in assembler language can be found in Tivoli NetView for z/OS Customization: Using Assembler.

The NetView Resource Object Data Manager (RODM) is a data cache that stores network configuration and status information about system resources. RODM enables you to automate network management functions associated with the resources defined to RODM. In addition, you can write RODM applications to perform other network management and automation tasks. Refer to the Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide for more information.

Functions to Consider before Making Modifications

To customize NetView functions, you can write your own command procedures or modify one of the existing command procedures supplied by the NetView program. Ways to modify existing functions include:

- Filtering or modifying the system management facility (SMF) records written by NetView
- Providing a policy that routes operator messages
- Reformatting, analyzing, or editing operator messages
- Checking command authority

Additional functions you might want to add involve managing additional components in your network, such as X.25 data network components or voice network components. You can develop new applications and integrate them with existing management functions to meet your requirements. Examples of these user-defined functions include:

- Real-time monitoring of specific resources, applications, or components in your network
- Collecting and recording additional SMF data for trend analysis or other data reduction applications you need
- Providing additional response time problem detection and alerting
- Detecting different classes of line problems and providing switched network backup (SNBU).

Finding Customization Information

Table 4 lists customization topics and provides the name of the documentation that includes information about that topic.

<table>
<thead>
<tr>
<th>Table 4. Customization Topics and Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
<tr>
<td>Alias names</td>
</tr>
<tr>
<td>Command Facility Screen Format</td>
</tr>
<tr>
<td>Automation</td>
</tr>
<tr>
<td>Generic alerts</td>
</tr>
</tbody>
</table>
## Customizing NetView

### Table 4. Customization Topics and Documentation (continued)

<table>
<thead>
<tr>
<th>Topic</th>
<th>CGD</th>
<th>GET</th>
<th>OLH</th>
<th>CLS</th>
<th>PLC</th>
<th>ASL</th>
<th>AUT</th>
<th>PIP</th>
<th>ASR</th>
<th>NUG</th>
<th>ADV</th>
</tr>
</thead>
<tbody>
<tr>
<td>National Language Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Operator control:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logon security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Span of control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panels:</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware monitor*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Help</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Help desk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User-written</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential logging</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session monitor data:*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Response time monitor monitor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Session awareness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Suppressing:</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Hardware monitor*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>User-written functions:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command lists</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>User-written programming (PL/I, C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>User-written programming (assembler)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>NetView Pipelines</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- CGD: Tivoli NetView for z/OS Customization Guide
- GET: Tivoli NetView for z/OS Installation: Getting Started
- OLH: NetView online help
- CLS: Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language
- PLC: Tivoli NetView for z/OS Customization: Using PL/I and C
- ASL: Tivoli NetView for z/OS Customization: Using Assembler
- AUT: Tivoli NetView for z/OS Automation Guide
- PIP: Tivoli NetView for z/OS Customization: Using Pipes
- ASR: Tivoli NetView for z/OS Administration Reference
- NUG: Tivoli NetView for z/OS User’s Guide
- ADV: Tivoli NetView for z/OS Installation: Configuring Additional Components

For information about customizing AON, refer to the Tivoli NetView for z/OS Automated Operations Network Customization Guide.

### Collecting Data

Typical sources for collecting data useful in customization procedures are:
- Installation exit interfaces provided in the NetView program
- System or NetView services that provide status, configuration, processing, or authorization information
Data files and network devices that are accessed using system or NetView services

Messages to operators indicating that important events are occurring in a system or an application.

Installation Exits
Some NetView installation exits allow access to network management data. Through these installation exits, and user-written functions you can obtain the text of operator commands, messages, and logons. Data that the NetView program writes to VSAM files and to the SMF log, as well as data on the VTAM communication network management (CNM) interface, can be accessed within other NetView installation exits.


Service Routines
System or NetView services give you access to information such as:
- System date and time
- Addresses of programs
- Addresses of named storage areas
- Valid NetView operators
- Operator span of control
- Values of command list variables.

Reference: Refer to the Tivoli NetView for z/OS Customization: Using Assembler for information about macros such as DSIDATIM, DSICES, DSIFIND, DSIQOS, DSIQRS, and DSIKVS. Refer to the Tivoli NetView for z/OS Customization: Using Assembler and C for information on service routines such as CNMINFC, CNMNAMS, CNMSCOP, and CNMVARS.

Data Files
The NetView program provides specialized disk services and VSAM data services to access network management data files. In addition to these, functions written in a high-level language (HLL), such as PL/I and C, can invoke system allocation and access methods to read from NetView partitioned data sets and request VSAM I/O. CNM interface services also provide access to data coming from devices in the network.

Using the NetView PIPE command, you can read data files using the QSAM and < (From Disk) stages. Through the pipe facility, you also have access to VSAM data using DSIVSAM and DSIVSMX. Refer to the Tivoli NetView for z/OS Customization: Using Pipes for information about DSIVSAM and DSIVSMX.

REXX command lists can make use of the EXECIO command to read from and write to sequential data sets or partitioned data set members.

Reference: Refer to the Tivoli NetView for z/OS Customization: Using PL/I and C for information about VSAM and CNM interface services.

For more information about pipes, refer to the Tivoli NetView for z/OS Customization: Using Pipes.
Operator Commands and Messages
You can issue operator commands within command procedures to request status data. The resulting response messages containing the requested status data can be trapped and processed in the command procedure. You can also process data in other system and network messages in user-written command procedures that are invoked through NetView automation.


Data Storage and Recording
You can use NetView command procedures to store and retrieve data needed for many user-written functions. Command procedures written in REXX, NetView command list language, PL/I, or C can create, set, and read global and task variables.

For permanent storage and for larger volumes of data, you can record certain information in data files rather than naming it and storing it as a command list variable. The NetView program allows you to record this data in a log. For example, you can log activities of your applications along with system or network activities that the NetView program is logging. You might want to produce a separate log of data that you collect.

Reference: Refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components and “Choosing a Language” on page 13 in this book for information on sequential logging.

Operator Presentation
You can customize or extend some of NetView’s operator presentation functions with the VIEW command or by modifying panels that some components of NetView use to present data to operators. See “Chapter 3. Using the VIEW Command” on page 29 and “Chapter 4. Modifying and Creating Online Help Information” on page 63 for more information.

You can also use messages to present information to operators. With messages, the data from user-written functions becomes subject to NetView automation processing, allowing both automatic and manual operation of your functions.

Reference: Refer to the Tivoli NetView for z/OS Customization: Using Assembler for information about DSIWCS, DSIMBS, DSIMQS, DSPSS, and other message services. Refer to the Tivoli NetView for z/OS Customization: Using PL/I and C for information about using CNMSMSG. Refer to the Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for information on REXX file input and output. Refer to the Tivoli NetView for z/OS Customization: Using Assembler for information on using DSIDKS for read access to NetView data sets or files, DSIZVMS for VSAM I/O, and DSIZCSMS for CNM data services.

Customizing NetView
Customizing NetView

Command List Language for descriptions of REXX and NetView command list language write-to-operator (WTO) messages and other message services.

You can also customize the NetView command facility panel. See Chapter 2, Customizing the NetView Command Facility Panel on page 25 for more information.

Tasks

To write functional extensions to the NetView program, keep in mind that the NetView design is based on MVS.

Reference: The MVS/ESA library is a good reference for explanations of how words such as dispatch, task, and the names of various system services are used in this section.

NetView Program as a System Application Program

The NetView program is organized into several parallel tasks, each one capable of being dispatched separately in a multitasking environment. When any one task is idle, any of the others is eligible to run. A system multitasking dispatcher uses the NetView program’s ATTACH system service to create each new task. When a task has no more processing to do and is ready to become idle, the task calls the WAIT system service. The POST system service takes a task out of an idle state, and allows it to be dispatched when new input data is ready to be processed for that task.

NetView Program Tasks

When the NetView program starts, its main task attaches several subtasks of different types, depending on the function to be performed. Each different task type determines the specific system interfaces and operator interfaces that are available under that task, and the type of transactions you can perform.

Each operator station task (OST) supports one NetView operator identified by a unique name. The operator identifiers (OPIDs) are defined in the NetView parameter library. OPIDs are assigned to an OST when an automated operator, known as an autotask, is activated using the AUTOTASK command, or when an operator logs on using a VTAM-connected terminal.

Each NetView-NetView task (NNT) also supports an operator. This type of task is used when the operator logs on to the NetView program from another NetView program rather than from a terminal. The other NetView program can be running in a different machine but must be connected through VTAM. The operator logs on from the other NetView program using the START DOMAIN command.

Each hardcopy task (HCT) supports a 3287 printer connected through VTAM to provide a hardcopy log for operators. See Figure 7 on page 8 for a structural overview of the command facility and its task structure.
There is only one primary program operator interface task (PPT) for each NetView program. When VTAM is running, the PPT opens a special VTAM application control block (ACB) for the VTAM programmable operator interface (POI) to receive unsolicited data from VTAM.

Each optional task (OPT) must be defined by a TASK statement in the NetView parameter library. The program module that runs for an OPT can be any program that meets the specification for optional tasks described in "Adding Optional Tasks to the NetView Program" on page 12.

Each data services task (DST) is a specific case of an optional task. See "Adding Optional Tasks to the NetView Program" on page 12. The TASK statement for a DST can name an initialization member in the NetView parameter library from which statements are read to define parameters for the functions performed by the specified DST.

Program Activity within a Task

After being activated, each type of NetView task waits for a request to perform a specific unit of work. When that unit of work is complete, the task enters a normal wait state. The task runs again when another request to perform a unit of work is received. Each task uses a list of event control blocks (ECBs) when it issues its WAIT. The NetView customization macros and services are provided to ensure that
any implied waiting is done through the ECB list of the task so that all of the task-request interfaces within the NetView program remain enabled.

Every NetView task has its own termination ECB and its own message queue ECB. Some types of tasks (for example, OSTs or DSTs) can have additional ECBs in their ECB lists. The additional ECBs represent processing that the task tests for and performs when it is posted out of its WAIT state.

**Queuing Work to NetView Program Tasks**

While a task is in its normal WAIT state, another task in the NetView program can run. A NetView task that is running can be interrupted at any time by an event in the system, and can be preempted by a higher-priority task until that task issues its normal WAIT. System functions outside of the NetView program can also interrupt the NetView processing by running scheduled interrupt exit routines that are associated with specific NetView tasks.

Data for a task can be placed in its message queue or another work queue, and the task can be posted to perform that work at any time. The data can originate in another NetView task. This can happen when a DST queues message data to an OST to be displayed to an operator. The data can come into the NetView program through an interrupt exit routine that is scheduled by an event such as the completion of a VTAM RECEIVE request.

**Message and Command Buffers**

The data placed in the various task queues is formatted into a special data structure called a message buffer or a command buffer. A header at the beginning of the buffer indicates the type of data the buffer contains and any special formats by which the data must be accessed. Commands are processed by programs called command processors that you provide in your customization programming for the NetView program. Messages are processed either according to predefinitions built into the NetView task, or by NetView automation command processors. Message buffers are also available for automation at various points in NetView processing through installation exits.

**Immediate Commands**

An immediate command starts processing as soon as an operator enters the command. The requested function is performed immediately, even if the task is in the middle of a large queue of work.

An immediate command runs under the OST and NNT subtask environments. Unlike other commands, immediate commands can receive control with the TVBINXIT bit set on. Immediate commands interrupt mainline processing and cannot be interrupted by another command. Immediate commands can be interrupted by other exits in asynchronous activity.

**Long-Running Commands**

A long-running command is a command that can suspend processing to allow other activity, such as operator commands and data retrieval, and then resume processing. All the NetView components are long-running commands. NetView command list language, REXX, PL/I, and C command procedures are also long-running commands. The DSIPUSH macro allows an assembler command to run as a long-running command.
Long-running commands run under an OST, NNT, PPT, or DST (logoff routines only). Long-running commands can be:
- Invoked directly by operator input
- Called by a command list
- Called by another long-running command.

Long-running commands return control to the NetView program after scheduling work but before processing is complete. The NetView program then processes other work that is pending.

You can use long-running command processors to retrieve data from another task or from another domain without allowing the calling function or calling command list to proceed during the retrieval. When the retrieval is executing, the processor’s task can continue to receive messages and accept commands.

**Data Services Commands**

A data services command processor (DSCP) runs under the DST subtask environment. DSCPs perform CNM data services and VSAM data services. DSCPs can also be called for centralized or serialized user-defined functions that do not use CNM interface or VSAM services.

**Defining User-Written Programs on the Host: Exits and Commands**

You can provide two types of user-written programs within the NetView task environments:
- Installation exits
- Command processors.

**Reference:** The programming interface details are provided in *Tivoli NetView for z/OS Customization: Using PL/I and C* and *Tivoli NetView for z/OS Customization: Using Assembler*. In designing user-written functions, you can use the installation exit interface and the command processor interface in the NetView program to fit your own programming into the overall structure of the NetView program.

**Installation Exit Programs**

Installation exits are provided in NetView at several points in the processing of logon and logoff data, command buffers, and message buffers. Different exits are driven based on the origin of the buffer and the stage of the NetView processing that the exit is in. Special exits are driven under DSTs to handle a task’s data during initialization, input, and output.

**Reference:** For a summary of the NetView installation exits, refer to the *Tivoli NetView for z/OS Automation Guide* and *Tivoli NetView for z/OS Customization: Using Assembler*. General installation exits are identified and invoked with preassigned module names of DSIEXnn, and the DST exits are uniquely identified in the task DSTINIT initialization statements.

DSIEX21 is used to access the DSITCPRF member. For more information, refer to the *Tivoli NetView for z/OS Security Reference*. 
Command Processors and Command Lists

NetView command processors and command lists can be started by:
- An operator request
- A command buffer queued to a task for processing by any NetView program
- A command call from another command processor
- An action specified in the NetView automation table.

Reference: To define command lists written in the NetView command list language or REXX to the NetView program, place them in the NetView command list library (ddname DSICLD). Refer to the Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language to find out how to create command lists for specific operating systems.

You must link-edit PL/I, C, and assembler command processors into the NetView load library (ddname STEPLIB), and define them to NetView. To define command processors written in PL/I, C or assembler to NetView, use a CMDMDL statement in the DSICMD member of DSIPARM. Command processors are link-edited into the NetView load library.

You can implement parts of a function in multiple installation exit programs and command processors. A common way of splitting a function across command processors is to divide processing between OSTs and DSTs. Because OSTs receive data from operator stations and return data back to them, a command processor is written to:
- Be called when the command is entered by an operator
- Parse the command data and form a data services request
- Queue a command buffer containing the data services command to be processed by the DST
- Return an error message or a command confirmation message to the operator

The DST completes the function in a separate command processor that is called because of the command buffer that is built and queued by the first command processor. Under the DST, functions requiring the special data services of VSAM, external logging, or the VTAM CNM interface are performed and messages can be returned to the operator task that queued the command. Figure 8 on page 11 shows a typical program design for a function that uses the CNM interface and VSAM services.

With long running commands, you can separate a complex function into a sequence of separate transactions. Command processors can establish a named

![Diagram](image_url)
Customizing NetView

stack entry where an anchor address is saved. A related command processor can later retrieve this address and perform another phase of the same processing.

When naming your commands, observe the following guidelines:
• Start with a letter (alphabetic)
• Avoid special characters such as commas and colons
• Avoid NetView command names, both internal commands and those shipped in DSICMD. NetView internal command names are CSCFDST, HMSTATS, LOGNMVT, LOGRU, MESSAGE, PIPE, and VIEW.
• Avoid the following NetView prefixes:

<table>
<thead>
<tr>
<th>AAU</th>
<th>BNT</th>
<th>EGV</th>
<th>EZL</th>
<th>FLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNH</td>
<td>CNM</td>
<td>EKG</td>
<td>FKB</td>
<td>FLC</td>
</tr>
<tr>
<td>BNI</td>
<td>DSI</td>
<td>EUY</td>
<td>FKV</td>
<td>FMG</td>
</tr>
<tr>
<td>BNJ</td>
<td>DUI</td>
<td>EXQ</td>
<td>FKW</td>
<td>FNA</td>
</tr>
<tr>
<td>BNK</td>
<td>DWO</td>
<td>EYV</td>
<td>FKB</td>
<td>IHS</td>
</tr>
</tbody>
</table>

Adding Optional Tasks to the NetView Program

You can write a completely new subtask in assembler language that the NetView program starts as an optional task (OPT) or subtask.

For an OPT, you must supply code for the subtask’s initialization, installation exit, message, and command processing functions and termination. Because some of these elements are already provided in an existing DST, using the DST as a starting point is more practical.

Reference: For more information on OPTs and DSTs in assembler language, refer to the Tivoli NetView for z/OS Customization: Using Assembler.

Choosing a Language

One application program interface might be more suitable than another for your particular customization requirements. Consider the effects on performance, ease of creation, and maintenance when determining the interface to use. This section describes the languages available and lists reasons that you might choose one language over another.

Input and Output

REXX, PL/I, C, and assembler all offer functions for reading from and writing to direct access storage devices (DASD). The NetView program provides specialized disk services and VSAM data services to access network management data files. In addition, functions written in PL/I or C can invoke system allocation and access methods to read and write data. CNM interface services also provide access to data coming from devices in the network.

Reference: Refer to the Tivoli NetView for z/OS Customization: Using PL/I and C for information about VSAM and CNM interface services. Refer to the Tivoli NetView for z/OS Customization: Using Assembler for information about using DSIDKS for read access to NetView data sets or files, DSIZVSMS for VSAM I/O, and DSIZCSMS for CNM data services.

Refer to the Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for information on REXX file input and output.
Performance

Write performance-critical applications in a compiled or assembled language. Generally, compiled or assembled command procedures execute faster than interpretive (REXX and NetView command list language) command lists.

You must write NetView-driven installation exit routines in assembler, PL/I, or C. Any command processor that accesses NetView control blocks must be written in assembler. Command procedures that can be driven by terminal input or by messages and that do not need to access NetView control blocks can usually be written in REXX or in NetView command list language. Generally, command lists written in REXX perform a little better than those written in NetView command list language. See "REXX Versus the NetView Command List Language". Additionally, the performance of REXX command lists can be improved by compiling the REXX command list.

Preloading a REXX or NetView command list (refer to the NetView online help for the LOADCL command) improves overall performance of the command list.

Reference: For details about compiling REXX command lists, refer to the Tivoli NetView for z/OS Tuning Guide.

For additional performance recommendations, refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components and Tivoli NetView for z/OS Installation: Configuring Additional Components.

Stability

If you anticipate changes to your procedures as you gain more experience or as your operating environment changes, you might want to use command lists to implement the procedures initially. Changes are easier to make in command lists because you can incorporate the changes and test them online without having to restart the NetView program. You can translate procedures into a compiled language, if desired, when you become confident of their stability.

Testing

Testing capabilities for command lists include the ability to trace execution using either operator commands or command list statements. A remote interactive debugger (RID), which displays information to a NetView operator console, can help you in debugging PL/I and C user-written command processors and installation exits. The NetView program does not provide any specific functions to help debug assembler programs.

Speed of Implementation

Because command lists are easy to write, test, and put into production, they can be an appropriate choice in addressing immediate operational needs.

REXX Versus the NetView Command List Language

If all of your systems can run REXX, choose REXX over the NetView command list language for writing command lists. REXX is a structured language that enables you to use subroutines. REXX is the easier language to learn and provides additional functions, such as mathematical capabilities and improved string handling. REXX can read from and write to data sets with EXECIO. Additionally, the performance of REXX command lists can be improved by compiling the REXX command list.
REXX language skills can be used in environments other than the NetView program. However, REXX procedures written for the NetView program probably will not be transportable to other environments because of their function content. In multiple environments, REXX is more useful because you can transfer REXX programming skills to solve NetView problems without learning another language.

If your installation uses several operating systems, it is possible that some of them support REXX and others do not. In this case, you can create bilingual command lists that contain both REXX and NetView command list versions of your instructions. The command lists execute in REXX if REXX is available; otherwise, they process in the NetView command list language.


Refer to the Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for more information about bilingual command lists.

Language Choices by Function

Table 5 lists additional capabilities to consider when choosing which language to use.

Table 5. Language Choices by Function

<table>
<thead>
<tr>
<th>Function</th>
<th>REXX or NetView CLIST</th>
<th>PL/I or C</th>
<th>Assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send message to NetView operator in line mode.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interact with operator through NetView operator’s screen (PAUSE/GO command).</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Invoke NetView commands.</td>
<td>Yes</td>
<td>Yes</td>
<td>Difficult</td>
</tr>
<tr>
<td>Trap and process messages destined for an operator.</td>
<td>Yes</td>
<td>Yes</td>
<td>Difficult</td>
</tr>
<tr>
<td>Access task and common global variables.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Create and access named areas of storage.</td>
<td>Yes REXX; No CLIST</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interact with operator through full-screen panels.</td>
<td>With VIEW command</td>
<td>With VIEW command</td>
<td>Difficult</td>
</tr>
<tr>
<td>Communicate non-SPCI data over the CNM interface.</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Access DASD or VSAM files. <strong>Note:</strong> The PIPE command provides the ability to read from disk. DSIVSAM and DSIVSMX provide access to VSAM files.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Program debugging support provided.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Implement NetView installation exits.</td>
<td>No</td>
<td>Most</td>
<td>Yes</td>
</tr>
<tr>
<td>Access NetView control blocks.</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Reference: Refer to the your specific programming language guides for considerations on writing in mixed languages.
Logging

The NetView program provides several ways to log information. Table 6 lists the available features of the common logging methods.

Table 6. Features of NetView Logging Methods

<table>
<thead>
<tr>
<th>Feature</th>
<th>Network Log</th>
<th>External SMF Log</th>
<th>External User-Defined Log</th>
<th>NetView Sequential Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access method</td>
<td>VSAM</td>
<td>VSAM</td>
<td>Sequential</td>
<td>BSAM</td>
</tr>
<tr>
<td>Device-independent</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>API-PL/1 and C*</td>
<td>CNMSMSG</td>
<td>CNMSMSG</td>
<td>CNMSMSG</td>
<td>CNMSMSG</td>
</tr>
<tr>
<td>API-assembler</td>
<td>DSIWLS</td>
<td>DSIWLS</td>
<td>DSIWLS</td>
<td>DSIWLS</td>
</tr>
<tr>
<td>Begin recording</td>
<td>START</td>
<td>Refer to Tivoli</td>
<td>Refer to Tivoli</td>
<td>Refer to Tivoli</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NetView for z/OS</td>
<td>NetView for z/OS</td>
<td>NetView for z/OS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Installation</td>
<td>Installation</td>
<td>Installation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configuring</td>
<td>Configuring</td>
<td>Configuring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Additional</td>
<td>Additional</td>
<td>Additional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Components</td>
<td>Components</td>
<td>Components</td>
</tr>
<tr>
<td>Browse</td>
<td>NetView</td>
<td>No</td>
<td>Operating system browse</td>
<td>Operating system browse</td>
</tr>
<tr>
<td></td>
<td>BROWSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple log tasks</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Variable length blocks and records</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Primary/secondary data sets or files</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SWITCH, RESUME, AUTOFLIP</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Installation exits</td>
<td>Many</td>
<td>XITXL</td>
<td>XITXL</td>
<td>XITBN, XITBO</td>
</tr>
</tbody>
</table>

Reference: For information about the network log, refer to the Tivoli NetView for z/OS Automation Guide. For information about external logging using the system management facility (SMF), a user-defined log, or sequential logging, refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components.

Cross-Reference for Message and Environment Functions

Table 7 on page 16, Table 8 on page 17, and Table 9 on page 18 provide a cross-reference for the NetView system data, task data, and message functions. With these matrices, you can determine whether the function you are interested in is available to the automation table, REXX, NetView command list language, or assembler. You can also determine what the name of the function is. Each matrix is alphabetized by the name of the REXX function.
Notes:
1. If you are writing assembler-language command processors, refer to the [Tivoli NetView for z/OS Customization: Using Assembler](#) for the BUFHDR mapping within the DSITIB mapping macro, the DSIFR mapping macro, and the DSIAIFRO mapping macro for exact field definitions.
2. If you are writing command lists, refer to the [Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language](#) for more information about NetView command list language control variables and REXX functions.
3. If you are writing in PL/I or C language, refer to the [Tivoli NetView for z/OS Customization: Using PL/I and C](#) for more information about the CNMINFC, CNMINFI, and CNMGETA service routines.
4. If you are writing automation table statements, refer to the [Tivoli NetView for z/OS Automation Guide](#) for a description of the automation table condition items.

### Table 7. Automation Variable Cross-Reference Table for System Data

The data returned is about the system. The same data is returned in every message for every task.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Condition Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID()</td>
<td>NetView address space identifier</td>
<td>Not available</td>
<td>CNMINFI ASID &amp;ASID</td>
<td>ASCBASID</td>
<td></td>
</tr>
<tr>
<td>CURSYS()</td>
<td>Current MVS system name</td>
<td>CURSYS</td>
<td>CNMINFC CURSYS &amp;CURSYS</td>
<td>CVTSNAME (MVS)</td>
<td></td>
</tr>
<tr>
<td>Date(USA)</td>
<td>Current date</td>
<td>Not available</td>
<td>CNMINFC DATE &amp;DATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOMAIN()</td>
<td>Current domain name</td>
<td>DOMAIN</td>
<td>CNMINFC DOMAIN &amp;DOMAIN</td>
<td>MVTCURAN</td>
<td></td>
</tr>
<tr>
<td>MVSLEVEL()</td>
<td>Current MVS system level</td>
<td>MVSLEVEL</td>
<td>CNMINFC MVSLEVEL &amp;MVSLEVEL</td>
<td>CVTPRODN (MVS)</td>
<td></td>
</tr>
<tr>
<td>NETID()</td>
<td>VTAM network identifier</td>
<td>NETID</td>
<td>CNMINFC NETID &amp;NETID</td>
<td>ACB vectors</td>
<td></td>
</tr>
<tr>
<td>NETVIEW()</td>
<td>NetView version and release identifier</td>
<td>NETVIEW</td>
<td>CNMINFC NVVER &amp;NETVIEW</td>
<td>MVTVER</td>
<td></td>
</tr>
<tr>
<td>OPSYSTEM()</td>
<td>Operating system NetView was compiled for</td>
<td>OPSYSTEM</td>
<td>CNMINFC OPSYSTEM &amp;OPSYSTEM</td>
<td>DSISYS Compiler variable</td>
<td></td>
</tr>
<tr>
<td>STCKGMT()</td>
<td>Greenwich Mean Time Store Clock Value</td>
<td>Not available</td>
<td>CNMINFC CLOCK 8-byte value &amp;STCKGMT 8-byte value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUPPCHAR()</td>
<td>In NetView, the character that suppresses the command echo or the command’s message output</td>
<td>Not available</td>
<td>CNMINFC SUPPCHAR &amp;SUPPCHAR</td>
<td>MVTPSPCHR</td>
<td></td>
</tr>
<tr>
<td>SYSPLEX()</td>
<td>1–8 character name of MVS SYSPLEX where the command list is executing</td>
<td>SYSPLEX</td>
<td>CNMINFC SYSPLEX &amp;SYSPLEX</td>
<td>ECVTSPLX</td>
<td></td>
</tr>
<tr>
<td>TIME(option)</td>
<td>Current time</td>
<td>Not available</td>
<td>CNMINFC TIME &amp;TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VTAM()</td>
<td>VTAM level if active</td>
<td>VTAM</td>
<td>CNMINFC VTAM &amp;VTAM</td>
<td>ACB vectors MVTACB ACBOPEN</td>
<td></td>
</tr>
</tbody>
</table>
### Table 7. Automation Variable Cross-Reference Table for System Data (continued)

The data returned is about the system. The same data is returned in every message for every task.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table</th>
<th>HLL Service</th>
<th>NetView Command List</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTCOMPID()</td>
<td>VTAM component identifier</td>
<td>VTCOMPID</td>
<td>CNMINFC VTCOMPID</td>
<td>&amp;VTCOMPID</td>
<td>ACB vectors MVTACB ACBOPEN</td>
</tr>
<tr>
<td>WEEKDAYN()</td>
<td>Decimal number representing day of week</td>
<td>WEEKDAYN</td>
<td>CNMINFI WEEKDAYN</td>
<td>&amp;WEEKDAYN</td>
<td></td>
</tr>
</tbody>
</table>

### Table 8. Automation Variable Cross-Reference Table for Task Data

The data is local to the task. The information is different for each task, but each message on that task has the same information.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table</th>
<th>HLL Service</th>
<th>NetView Command List</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVCLOSE</td>
<td>NetView program termination indicator</td>
<td>NVCLOSE</td>
<td>CNMINFI CLOSING</td>
<td>Not available</td>
<td>MVTCLOSE</td>
</tr>
<tr>
<td>APPLID()</td>
<td>Application name of the current task</td>
<td>Not available</td>
<td>CNMINFC APPLID</td>
<td>&amp;APPLID</td>
<td>TVBAPID</td>
</tr>
<tr>
<td>ARG()</td>
<td>Input parameters for the active command list</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;PARMSTR</td>
<td></td>
</tr>
<tr>
<td>ATTENDED()</td>
<td>Task information</td>
<td>ATTENDED</td>
<td>CNMINFI ATTENDED</td>
<td>&amp;ATTENDED</td>
<td>TVBSYSCN TVBAUTOO TVBDAUT</td>
</tr>
<tr>
<td>AUTCONID()</td>
<td>MVS console name or ID that is associated with an autotask. This MVS console can issue NetView commands to run under this autotask.</td>
<td>Not available</td>
<td>CNMINFC AUTCONID</td>
<td>&amp;AUTCONID</td>
<td>TVBSYSCN TVBCNAME</td>
</tr>
<tr>
<td>AUTOTASK()</td>
<td>Autotask indicator</td>
<td>AUTOTASK</td>
<td>CNMINFI AUTOTASK</td>
<td>&amp;AUTOTASK</td>
<td>TVBAUTOO</td>
</tr>
<tr>
<td>COMPNAME()</td>
<td>Component name that was active when command list invoked</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;COMPNAME</td>
<td></td>
</tr>
<tr>
<td>CURCONID()</td>
<td>MVS console name or ID used by a NetView task to issue MVS commands and receive MVS system messages</td>
<td>Not available</td>
<td>CNMINFC CURCONID</td>
<td>&amp;CURCONID</td>
<td>TVBMCSNNU TVBMCSNA</td>
</tr>
<tr>
<td>DISTAUTO()</td>
<td>Distributed autotask indicator</td>
<td>DISTAUTO</td>
<td>CNMINFI DISTAUTO</td>
<td>&amp;DISTAUTO</td>
<td>TVBDAUT</td>
</tr>
<tr>
<td>HCOPY()</td>
<td>Hardcopy task for this task</td>
<td>Not available</td>
<td>CNMINFC HCOPY</td>
<td>&amp;HCOPY</td>
<td>TVBHCTVB -&gt; TVBOPID</td>
</tr>
<tr>
<td>LU()</td>
<td>Terminal name of the currently running task</td>
<td>Not available</td>
<td>CNMINFC LU</td>
<td>&amp;LU</td>
<td>TVBLUNAM</td>
</tr>
</tbody>
</table>
**Table 8. Automation Variable Cross-Reference Table for Task Data (continued).** The data is local to the task. The information is different for each task, but each message on that task has the same information.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Condition Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVCNT()</td>
<td>Number of domains available</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;NCCFCNT</td>
<td></td>
</tr>
<tr>
<td>NVID(n)</td>
<td>Domain ID array</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;NCCFID number</td>
<td></td>
</tr>
<tr>
<td>NVSTAT(name)</td>
<td>Domain status</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;NCCFSTAT name</td>
<td></td>
</tr>
<tr>
<td>OPID()</td>
<td>ID of currently running task</td>
<td>OPID</td>
<td>CNMINFC OPI, or CNMINFC TASKNAME</td>
<td>&amp;OPID</td>
<td>TVBOPID</td>
</tr>
<tr>
<td>PARMCNT()</td>
<td>Number of input parameters to the active command list</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;PARMCNT</td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>Return code</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;RETCODE Register 15</td>
<td></td>
</tr>
<tr>
<td>TASK()</td>
<td>Type of task</td>
<td>TASK</td>
<td>CNMINFC TASK</td>
<td>&amp;TASK CBHTYPE in DSITVB</td>
<td></td>
</tr>
<tr>
<td>WTOREPLY</td>
<td>WTOR reply text</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;WTOREPLY</td>
<td></td>
</tr>
</tbody>
</table>

**Table 9. Automation Variable Cross-Reference Table for Message Data.** Data is different for each message or MSU. The message ID is message data.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Condition Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONDL()</td>
<td>Message deletion reason</td>
<td>ACTIONDL</td>
<td>CNMCAGA ACTIONDL</td>
<td>&amp;ACTIONDL</td>
<td>IFRAUDLO IFRAUDTO IFRAUNVD IFRAUDFL IFRAUDF2</td>
</tr>
<tr>
<td>ACTIONMG()</td>
<td>Action message</td>
<td>ACTIONMG</td>
<td>CNMCAGA ACTIONMG</td>
<td>&amp;ACTIONMG</td>
<td>IFRAUACN</td>
</tr>
<tr>
<td>AREAID()</td>
<td>MVS area ID</td>
<td>AREAID</td>
<td>CNMGETA AREAID</td>
<td>&amp;AREAID</td>
<td>IFRAUWMA CPOCAREA MDBCARA</td>
</tr>
<tr>
<td>AUTOTOKE()</td>
<td>MPF automation token</td>
<td>AUTOTOKE</td>
<td>CNMGETA AUTOTOKE</td>
<td>&amp;AUTOTOKE</td>
<td>IFRAAUTOK CPOCAUTO MDBCAUTO</td>
</tr>
</tbody>
</table>
### Table 9. Automation Variable Cross-Reference Table for Message Data (continued)

Data is different for each message or MSU. The message ID is message data.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Condition Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART()</td>
<td>8-byte command and response token</td>
<td>CART</td>
<td>CNMGETA CART</td>
<td>&amp;CART</td>
<td>CPOCCART</td>
</tr>
<tr>
<td>DESC()</td>
<td>2 bytes of MVS descriptor codes</td>
<td>DESC</td>
<td>CNMGETA DESC</td>
<td>&amp;DESC</td>
<td>IFRAUWDS</td>
</tr>
<tr>
<td>GETMLINE</td>
<td>Message text</td>
<td>TEXT</td>
<td>CNMGETD GETFIRST or CNMGETD GETNEXT</td>
<td>GETMLINE command</td>
<td></td>
</tr>
<tr>
<td>GETMPRES</td>
<td>4 bytes of presentation attributes</td>
<td>LINEPRES</td>
<td>Not available</td>
<td>GETMPRES command</td>
<td>HDRTMTPA</td>
</tr>
<tr>
<td>GETMSIZE</td>
<td>2-byte count of number of lines of message</td>
<td>Not available</td>
<td>Not available</td>
<td>GETMSIZE command</td>
<td>CPOCLCNT</td>
</tr>
<tr>
<td>GETMTFLG</td>
<td>2 bytes of text object flags</td>
<td>LINETFLG</td>
<td>Not available</td>
<td>GETMTFLG command</td>
<td>HDRTLNTY</td>
</tr>
<tr>
<td>HDRMTYPE()</td>
<td>NetView message type</td>
<td>HDRMTYPE</td>
<td>ORIG_MSG_TYPE</td>
<td>&amp;HDRMTYPE</td>
<td>HDRMTYPE</td>
</tr>
<tr>
<td>IFRAUGMT()</td>
<td>8-byte hexadecimal Store Clock value when AIFR was created</td>
<td>None</td>
<td>CNMGETA IFRAUGMT</td>
<td>&amp;IFRAUGMT</td>
<td>IFRAUGMT</td>
</tr>
</tbody>
</table>

Customizing NetView

Chapter 1. Designing Functions 19
Customizing NetView

Table 9. Automation Variable Cross-Reference Table for Message Data (continued). Data is different for each message or MSU. The message ID is message data.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Condition Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUIND()</td>
<td>2 bytes of automation IFR indicator flags</td>
<td>IFRAUIND(nn)</td>
<td>CNMGETA IFRAUIND</td>
<td>&amp;IFRAUIND</td>
<td>IFRAUIND</td>
</tr>
<tr>
<td>IFRAUIN3()</td>
<td>1 byte of indicator bits</td>
<td>IFRAUIN3(nn)</td>
<td>CNMGETA IFRAUIN3</td>
<td>&amp;IFRAUIN3</td>
<td>IFRAUIN3</td>
</tr>
<tr>
<td>IFRAUIN3X()</td>
<td>32-bit field of which IFRAUIN3 are the first 8 bits</td>
<td>IFRAUIN3X</td>
<td>CNMCAGA IFRAUIN3X</td>
<td>&amp;IFRAUIN3 X</td>
<td>IFRAUIN3X</td>
</tr>
<tr>
<td>IFRAUNVF</td>
<td>MVS Retain Flags</td>
<td>MVSRTAIN</td>
<td>CNMGETA MVSRTAIN</td>
<td>&amp;MVSRTAIN</td>
<td>IFRAUNVF</td>
</tr>
<tr>
<td>IFRAUSDR()</td>
<td>Original sender of a message or MSU, whereas HDRSENDR is unreliable</td>
<td>IFRAUSDR</td>
<td>CNMGETA IFRAUSDR</td>
<td>&amp;IFRAUSDR</td>
<td>IFRAUSDR</td>
</tr>
<tr>
<td>IFRAUSRB()</td>
<td>2-byte user field from the AIFR. This user field can be referenced either as bits or characters.</td>
<td>IFRAUSRB(nn), IFRAUSB2(n)</td>
<td>CNMGETA IFRAUSRB, IFRAUSB2</td>
<td>&amp;IFRAUSRB IFRAUSB2</td>
<td>IFRAUSRB</td>
</tr>
<tr>
<td>IFRAUSC2()</td>
<td>16-byte user field from the AIFR. This user field can be referenced either as bits or characters.</td>
<td>IFRAUSRC, IFRAUSC2</td>
<td>CNMGETA IFRAUSRC IFRAUSC2</td>
<td>&amp;IFRAUSRC IFRAUSC2</td>
<td>IFRAUSRC</td>
</tr>
<tr>
<td>IFRAUTA1()</td>
<td>6 bytes of control flags</td>
<td>IFRAUTA1(nn)</td>
<td>CNMGETA IFRAUTA1</td>
<td>&amp;IFRAUTA1</td>
<td>IFRAUTA1</td>
</tr>
<tr>
<td>IFRAUWF1()</td>
<td>4 bytes of MVS specific WQE flags</td>
<td>IFRAUWF1(nn)</td>
<td>CNMGETA IFRAUWF1</td>
<td>&amp;IFRAUWF1</td>
<td>IFRAUWF1</td>
</tr>
<tr>
<td>JOBNAME()</td>
<td>8-byte MVS job name</td>
<td>JOBNAME</td>
<td>CNMGETA JOBNAME</td>
<td>&amp;JOBNAME</td>
<td>IFRAUWJA GOJGJBNM MDBGJBNM</td>
</tr>
<tr>
<td>JOBNUM()</td>
<td>8-byte MVS job number</td>
<td>JOBNUM</td>
<td>CNMGETA JOBNUM</td>
<td>&amp;JOBNUM</td>
<td>IFRAUWJU CPOCOJID MDBCOJID</td>
</tr>
<tr>
<td>KEY()</td>
<td>8-byte key associated with a message</td>
<td>KEY</td>
<td>CNMGETA KEY</td>
<td>&amp;KEY</td>
<td>CPOCKEY MDBCKEY</td>
</tr>
<tr>
<td>LINETYPE()</td>
<td>Message MLWTO indicators</td>
<td>Not available</td>
<td>ORIG_LINE_TYPE</td>
<td>&amp;LINETYPE</td>
<td>HDRLNTYDFRAUWF1(3)</td>
</tr>
</tbody>
</table>

GETMTYPE command contains the line type only after CNMGETD has been issued.
### Table 9. Automation Variable Cross-Reference Table for Message Data (continued)

Data is different for each message or MSU. The message ID is message data.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCSFLAG()</td>
<td>2 bytes of MVS MCS flags</td>
<td>MCSFLAG</td>
<td>CNMGETA MCSFLAG</td>
<td>&amp;MCSFLAG</td>
<td>IFRAUMCS</td>
</tr>
<tr>
<td></td>
<td>In command lists, PL/I, and C, MCSFLAG returns a selection of eight MCSFLAG bits. In the automation table, MCSFLAG returns 16 bits that match the assembler control block field.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSGASID()</td>
<td>MVS system address space identifier</td>
<td>Not available</td>
<td>CNMGETA MSGASID</td>
<td>&amp;MSGASID</td>
<td>IFRAUASI IFRAUWAS CPOCASID MDBCASID</td>
</tr>
<tr>
<td>MSGAUTH()</td>
<td>Indicates whether an MVS system message was issued by an authorized program</td>
<td>MSGAUTH</td>
<td>CNMGETA MSGAUTH</td>
<td>&amp;MSGAUTH</td>
<td>CPOCAUTH MDBCAUTH</td>
</tr>
<tr>
<td>MSGCATTR()</td>
<td>2 bytes of MVS message attributes flags</td>
<td>MSGCATTR</td>
<td>CNMGETA MSGCATTR</td>
<td>&amp;MSGCATTR</td>
<td>CPOCATTR MDBCATTR</td>
</tr>
<tr>
<td>MSGCMISC()</td>
<td>1 byte of MVS miscellaneous routing information flags</td>
<td>MSGCMISC</td>
<td>CNMGETA MSGCMISC</td>
<td>&amp;MSGCMISC</td>
<td>CPOCMISC MDBCMISC</td>
</tr>
<tr>
<td>MSGCMMLVL()</td>
<td>2 bytes of MVS message-level flags</td>
<td>MSGCMMLVL</td>
<td>CNMGETA MSGCMMLVL</td>
<td>&amp;MSGCMMLVL</td>
<td>CPOCMMLVL MDBCAUTH</td>
</tr>
<tr>
<td>MSGCMMSGT()</td>
<td>2 bytes of message type flags</td>
<td>MSGCMMSGT</td>
<td>CNMGETA MSGCMMSGT</td>
<td>&amp;MSGCMMSGT</td>
<td>CPOCMMSGT MDBCMMSGT</td>
</tr>
<tr>
<td>MSGCNT()</td>
<td>Number of tokens in a message</td>
<td>Not available</td>
<td>Not available</td>
<td>&amp;MSGCNT</td>
<td></td>
</tr>
<tr>
<td>MSGCOJBN()</td>
<td>8-character originating job name</td>
<td>MSGCOJBN</td>
<td>CNMGETA MSGCOJBN</td>
<td>&amp;MSGCOJBN</td>
<td>CPOCOJBN MDBCOJBN</td>
</tr>
<tr>
<td>MSGCPROD()</td>
<td>MVS system product level of the system that issued the message</td>
<td>MSGCPROD</td>
<td>CNMGETA MSGCPROD</td>
<td>&amp;MSGCPROD</td>
<td>CPOCPROD MDBCPROD</td>
</tr>
<tr>
<td>MSGCSPLX()</td>
<td>1–8 character name of MVS SYSPLEX where the received message originated</td>
<td>MSGCSPLX</td>
<td>CNMGETA MSGCSPLX</td>
<td>&amp;MSGCSPLX</td>
<td>CPOCSPLX</td>
</tr>
<tr>
<td>MSGCSYID()</td>
<td>Decimal system ID (for DOM)</td>
<td>Not available</td>
<td>CNMGETA MSGCSYID</td>
<td>&amp;MSGCSYID</td>
<td>CPOCSYID MDBCSYID</td>
</tr>
<tr>
<td>MSGDOMFL()</td>
<td>1 byte of DOM flags</td>
<td>MSGDOMFL</td>
<td>CNMGETA MSGDOMFL</td>
<td>&amp;MSGDOMFL</td>
<td>CPOMFL MDBDOMFL</td>
</tr>
<tr>
<td>MSGGBGPA()</td>
<td>4 bytes of background presentation attributes</td>
<td>MSGGBGPA</td>
<td>CNMGETA MSGGBGPA</td>
<td>&amp;MSGGBGPA</td>
<td>GOJGBGPA MDBGBGPA</td>
</tr>
</tbody>
</table>
### Table 9. Automation Variable Cross-Reference Table for Message Data (continued)

Data is different for each message or MSU. The message ID is message data.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGGDATE()</td>
<td>7-character date in the form yyyyddd</td>
<td>MSGGDATE</td>
<td>CNMGETA MSGGDATE</td>
<td>&amp;MSGGDATE GOJGDSHP MDBGDSHP</td>
<td></td>
</tr>
<tr>
<td>MSGFGP()</td>
<td>4 bytes of foreground presentation attributes</td>
<td>MSGFGP</td>
<td>CNMGETA MSGFGP</td>
<td>&amp;MSGFGP GOJFGP MDBFGP</td>
<td></td>
</tr>
<tr>
<td>MSGMFLG()</td>
<td>2 bytes of MVS general message flags</td>
<td>MSGMFLG</td>
<td>CNMGETA MSGMFLG</td>
<td>&amp;MSGMFLG GOJGMFLG MDBGMFLG</td>
<td></td>
</tr>
<tr>
<td>MSGGMID()</td>
<td>MVS message ID field</td>
<td>MSGGMID</td>
<td>CNMGETA MSGGMID</td>
<td>&amp;MSGGMID GOJGMID MDBGMID</td>
<td></td>
</tr>
<tr>
<td>MSGSEQ()</td>
<td>MVS message sequence number. This sequence number, together with MSGSYID, determine MSGMID.</td>
<td>Not available</td>
<td>CNMGETA MSGSEQ</td>
<td>&amp;MSGSEQ GOJSEQ</td>
<td></td>
</tr>
<tr>
<td>MSGSYID()</td>
<td>System ID of the MVS system from which the message was issued</td>
<td>Not available</td>
<td>CNMGETA MSGSYID</td>
<td>&amp;MSGSYID GOJGSYID MDBGSYID</td>
<td></td>
</tr>
<tr>
<td>MSGTIME()</td>
<td>11-byte time hh.mm.ss.th character string</td>
<td>MSGTIME</td>
<td>CNMGETA MSGTIME</td>
<td>&amp;MSGTIME GOJGTIMH MDBGTIMH GOJGTIMT MDBGTIMT</td>
<td></td>
</tr>
<tr>
<td>MSGID()</td>
<td>Message ID, which is not always the first item of a message. For example, if the message is a WTOR, a REPLYID precedes the message ID.</td>
<td>MSGID</td>
<td>ORIG_PROCESS</td>
<td>&amp;MSGID</td>
<td></td>
</tr>
<tr>
<td>MSGORIGIN()</td>
<td>Message domain name (or sometimes TAF session name). This always returns the domain name in AIFR buffers.</td>
<td>DOMAINTID</td>
<td>ORIG_DOMAIN</td>
<td>&amp;MSGORIGIN HDRDOMID</td>
<td></td>
</tr>
<tr>
<td>MSGSRCNM()</td>
<td>1–17 character source name from the source object</td>
<td>MSGSRCNM</td>
<td>CNMGETA MSGSRCNM</td>
<td>&amp;MSGSRCNM MSOSUBDA MSOSBNIK MSOSBNID MSOSBNAU</td>
<td></td>
</tr>
<tr>
<td>MSGSTR()</td>
<td>Text of message after the message ID</td>
<td>Not available</td>
<td>CNMGETD GETFIRST or CNMGETD GETNEXT</td>
<td>&amp;MSGSTR</td>
<td></td>
</tr>
<tr>
<td>MSGTOKEN()</td>
<td>Numeric token associated with message</td>
<td>Not available</td>
<td>CNMGETA MSGTOKEN</td>
<td>&amp;MSGTOKEN CPOCTOKN MDBCTOKN</td>
<td></td>
</tr>
<tr>
<td>MSGTSTMP()</td>
<td>Message time stamp</td>
<td>Not available</td>
<td>CNMGETA MSGTSTMP</td>
<td>&amp;MSGTSTMP HDRTSTMP</td>
<td></td>
</tr>
</tbody>
</table>
Table 9. Automation Variable Cross-Reference Table for Message Data (continued). Data is different for each message or MSU. The message ID is message data.

<table>
<thead>
<tr>
<th>REXX Function</th>
<th>Description</th>
<th>Automation Table Condition Item</th>
<th>HLL Service Routine and Options</th>
<th>NetView Command List Language Control Variable</th>
<th>Control Block Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVDElid()</td>
<td>NetView DOM ID</td>
<td>NVDElid</td>
<td>CNMCAgA NVDElid</td>
<td>&amp;NVDElid</td>
<td>IFRAUGMT HDRDOMID</td>
</tr>
<tr>
<td>MSGVAR(n)</td>
<td>Tokens of the message</td>
<td>TOKEN</td>
<td>CNMGETD GETFIRST or CNMGETD GETNEXT</td>
<td>&amp;1 - &amp;31</td>
<td></td>
</tr>
<tr>
<td>PARTID()</td>
<td>First two characters of a VSE message prefix, which, for some VSE messages, indicates the VSE partition ID</td>
<td>PARTID</td>
<td>CNMGETA PARTID</td>
<td>&amp;PARTID</td>
<td></td>
</tr>
<tr>
<td>PRTY()</td>
<td>2-byte MVS message priority</td>
<td>Not available</td>
<td>CNMGETA PRTY</td>
<td>&amp;PRTY</td>
<td>CPOCPRTY MDBCPRTY</td>
</tr>
<tr>
<td>REPLYID()</td>
<td>Reply ID</td>
<td>Not available</td>
<td>CNMGETA REPLYID</td>
<td>&amp;REPLYID</td>
<td>CPOCRCPI MDBCRCPI</td>
</tr>
<tr>
<td>ROUTCDE()</td>
<td>16 bytes of MVS routing codes (128 bits)</td>
<td>ROUTCDE</td>
<td>CNMGETA ROUTCDE</td>
<td>&amp;ROUTCDE</td>
<td>IFRAUWRT CPOCCRC MDBCCRC</td>
</tr>
<tr>
<td>SESSID()</td>
<td>TAF session name</td>
<td>SESSID</td>
<td>CNMGETA SESSID</td>
<td>&amp;SESSID</td>
<td>IFRAAUTAF</td>
</tr>
<tr>
<td>SMSGID()</td>
<td>MVS system message ID for DOM correlation</td>
<td>Not available</td>
<td>CNMGETA SMSGID</td>
<td>&amp;SMSGID</td>
<td>IFRAUWID</td>
</tr>
<tr>
<td>SYSCONID()</td>
<td>The MVS console name or ID that is associated with the message</td>
<td>SYSCONID</td>
<td>CNMGETA SYSCONID</td>
<td>&amp;SYSCONID</td>
<td>IFRAUWUC IFRAUCON CPOCCNID MDBCCNID</td>
</tr>
<tr>
<td>SYSID()</td>
<td>8-byte MVS system name that is associated with the message</td>
<td>SYSID</td>
<td>CNMGETA SYSID</td>
<td>&amp;SYSID</td>
<td>IFRAUWSSN GOJGOSNM MDBGOSNM</td>
</tr>
</tbody>
</table>

Customizing PF Keys and Immediate Message Line

You can set global variables that can be searched for and placed on the PF key line on panels displayed by BROWSE, STATMON, and VIEW commands. On VIEW panels, the immediate message line is also used as the PF key line. The variable names are prefixed by (&)CNMIM and followed by the application name. Valid variables include CNMIMLBROWSE, CNMIMMBROWSE, CNMIMSTATMON, CNMIMVIEW, and CNMIMWINDOW.
For View panels, if the VIEW application has not provided a value for CNMIMDL, VIEW searches the global dictionaries (task, then common) for a variable named CNMIMxxx, where xxx is the application name provided when VIEW was invoked. If the CNMIMxxx variable is not found, VIEW searches for CNMIMVIEW in the same dictionaries. This is similar to the way keys are set for VIEW applications. Finally, if none of these variables is present, the text from message BNH257I is used.

**Modifying CNMKEYS**

```
-------------- DEFINE TEXT FOR KEY LINES --------------
*
* The separator line above is required in any key definition file
* which defines "key line" texts. This separator line MUST begin
* with 9 dashes. All key definitions must precede this line, and
* all "key line" definitions must follow it.
*
* Optionally uncomment and modify the following statements, which
* assign values to the "key line" area of Statmon, Browse and View
* panels. The same rules are followed in this section as above with
* respect to commas and continuation lines. Keep the variable name
* between the delimiters, and PFKDEF will assign the rest of the line
* (including continuations) to that variable. Do not use leading
* blanks.
*
*/CNMIMSTATMON/1=HLP 2=END 3=RET 4=KYS 5=LOG 6=,
*ROL 7=BCK 8=FWD 9=SR 10=SV 11=SC 12=RTV
*/CNMIMLBROWSE/1=HLP 2=END 3=RET 4=KYS 5=RPF 6=,
*ROL 7=BCK 8=FWD 9=TOP 10=LFT 11=RGT 12=RTV
*/CNMIMMBROWSE/1=HLP 2=END 3=RET 4=KYS 5=RPF 6=,
*ROL 7=BCK 8=FWD 9=WON 10=WIN 11=WHO 12=RTV
*/CNMIMVIEW/1=HLP 2=END 3=RET 4=KYS 5=LOG 6=,
*ROL 7=BCK 8=FWD 9=TOP 10=WIN 11=ENT 12=RTV
*/CNMIMWINDOW/1=HLP 2=RFR 3=RET 4=KYS 5=FIN 6=,
*ROL 7=BCK 8=FWD 9=TOP 10=LFT 11=RGT 12=RTV
```

*Figure 9. Excerpt from CNMKEYS Sample to Set PF Keys*

The PFKDEF command list (CNME1010) can assign one or more task global variables from the target file to match the key settings for applicable NetView applications. Figure 9 shows how you can set the PF keys for the Browse, Status Monitor, and View panels.
Chapter 2. Customizing the NetView Command Facility Panel

The NetView command facility panel can be customized. You can customize:

- The colors of fields on the panel
- The information that precedes the message text
- The default colors for held, action, normal, and immediate classes of messages
- The color of the command area
- How much of the panel area is set aside for held and action messages

Using a Screen Format Definition

You can use a screen format (SCRNFMT) definition to specify attributes for the command facility panel and a default value for the color of messages. To activate the screen format definition, use the NetView DEFAULTS and OVERRIDE commands. Refer to NetView online help for details on how to use DEFAULTS and OVERRIDE. A short description of each option that can be specified in a screen format definition is listed under "Screen Format Definition Statements".

Reference: For detailed descriptions of the screen format definition statements, refer to Tivoli NetView for z/OS Administration Reference. CNMSCNFT is a sample screen format definition, provided in Tivoli NetView for z/OS Installation: Configuring Additional Components.

Notes:

1. Color and highlighting must be supported by your hardware and emulator. In addition, you must log on to NetView with a query-type logmode.

2. When you replace an active screen format definition with a new screen format definition, all definition statements are replaced. Any definition statement not specified in the new screen format definition will use the NetView-supplied value. The NetView-supplied values for each definition statement are listed in Tivoli NetView for z/OS Administration Reference.

For example, a screen format definition has been activated with the DEFAULTS command. Subsequently, operators activate customized screen format definitions using the OVERRIDE command. The statements not specified in an operator's screen format definition uses the NetView-supplied value, NOT the value from the screen format definition that was activated with the DEFAULTS command.

Screen Format Definition Statements

The following screen shows the fields that you can customize on the NetView message panel.
Customizing the NCCF Panel

The following formats can be customized:

1. **Title area**
   Use the TITLE statement in a SCRNFMT definition to customize the color of NETVIEW on the screen.

2. **Domain identifier**
   Use the TITLEDOMID statement in a SCRNFMT definition to customize the color of the NetView domain name.

3. **Operator identifier**
   Use the TITLEOPID statement in a SCRNFMT definition to customize the color of the operator name.

4. **Current date**
   Use the TITLEDATE statement in a SCRNFMT definition to customize the color of the date. You can also customize the format of the date using the DEFAULTS and OVERRIDE commands.

5. **Time data was last displayed**
   Use the TITLETIME statement in a SCRNFMT definition to customize the color of the time. You can also customize the format of the time using the DEFAULTS and OVERRIDE commands.

6. **and 7. System states**
   Use the TITLESTAT statement in a SCRNFMT definition to customize the color of the status characters in the upper right corner of the panel.

8. **COLUMNHEAD line**
   Use the COLUMNHEAD statement in a SCRNFMT definition to create a line at the top of the screen with labels for prefixes. This line can have up to 16 tags (C1...C16) in any order. Total length of tags, including one space between each tag, cannot exceed 78 characters. Set the tags using the SCRNFMT definition. The PREFIX and NOPREFIX statements control which tags appear. You can also choose not to have the line appear on the screen.

9. **Output area**
   Use the HELD, ACTION, NORMAL, and NORMQMAX statements of the SCRNFMT definition.

---

**Figure 10. NetView Message Panel**

The following formats can be customized:

1. **Title area**
   Use the TITLE statement in a SCRNFMT definition to customize the color of NETVIEW on the screen.

2. **Domain identifier**
   Use the TITLEDOMID statement in a SCRNFMT definition to customize the color of the NetView domain name.

3. **Operator identifier**
   Use the TITLEOPID statement in a SCRNFMT definition to customize the color of the operator name.

4. **Current date**
   Use the TITLEDATE statement in a SCRNFMT definition to customize the color of the date. You can also customize the format of the date using the DEFAULTS and OVERRIDE commands.

5. **Time data was last displayed**
   Use the TITLETIME statement in a SCRNFMT definition to customize the color of the time. You can also customize the format of the time using the DEFAULTS and OVERRIDE commands.

6. **and 7. System states**
   Use the TITLESTAT statement in a SCRNFMT definition to customize the color of the status characters in the upper right corner of the panel.

8. **COLUMNHEAD line**
   Use the COLUMNHEAD statement in a SCRNFMT definition to create a line at the top of the screen with labels for prefixes. This line can have up to 16 tags (C1...C16) in any order. Total length of tags, including one space between each tag, cannot exceed 78 characters. Set the tags using the SCRNFMT definition. The PREFIX and NOPREFIX statements control which tags appear. You can also choose not to have the line appear on the screen.

9. **Output area**
   Use the HELD, ACTION, NORMAL, and NORMQMAX statements of the SCRNFMT definition.
Note: HELD, ACTION and NORMAL statements set default colors for messages. If message color has been previously set, the default message color will not take effect. See “Message Color and Highlighting” on page 28 for more information.

The NORMQMAX statement specifies how many normal messages are queued for later display (excluding held and action messages). For example, the number of messages kept while you are working in another panel, or while the panel is locked.

When the NORMQMAX is exceeded, the NetView program automates and logs (if required) incoming messages and then discards them, without interrupting the operator. The oldest messages are discarded until the number of queued messages is half the NORMQMAX value.

When the operator returns to the command facility (or the panel is unlocked), message DSI593A indicates how many messages were discarded.

The value of NORMQMAX can range from 0 to 2147483647; the default is 3000. The minimum value allowed is 100 messages, so if you specify less than 100, it will be rounded to 100. Specifying a NORMQMAX value of 0 means an infinite queue, and is basically the same as specifying the maximum value of 2147483647.

Attention: Setting the value of NORMQMAX too high might cause out of storage conditions. Conversely, setting the value too low can prevent your operators from seeing all of their messages even when message traffic rates are low.

The NORMQMAX value also applies to hardcopy printers and to OST-NNT cross-domain sessions. Hardcopy printers can get backlogged because they are slow or because they run out of paper. An OST-NNT session can get backlogged because the message traffic over the session exceeds the send rate for that session.

Area for held and action messages

Use the HOLDPCNT statement in the SCRFMT definition. The NetView program uses 10 lines of the screen for the title line, immediate message area, command area, and a warning held-message: DSI151I. Held messages are not displayed in these 10 lines. You can use HOLDPCNT to specify what percentage of the remaining lines you want to use for held messages. For example, on a 24-line screen, setting HOLDPCNT to 100% will give you 14 lines for held messages.

Specifying HOLDPCNT as 0 means that held messages will not be displayed on the screen. If HOLDPCNT is non-zero, the minimum number of lines used for held messages is two.

You can use HOLDWARN to get a warning message that held messages exist, even though they are not displayed on the screen.

Note: The NetView program will not display the control line of a held message without the data line of the message. This helps prevent operators from accidentally erasing a held message without seeing the text.
Indentation
Use the INDENT and MLINDENT statements in the SCRFMT definition.

Separator line
The LASTLINE statement of the SCRFMT definitions changes the color of
the dashed separator line between the new and old messages of the screen.

Command entry indicator
Use the CMDLINE statement of the SCRFMT definition.

Lock/unlock indicator (***)
Use the LOCKIND statement in the SCRFMT definition.

Immediate message area
Use the IMDAREA statement in the SCRFMT definition.

Command area
Use the CMDLINE statement in the SCRFMT definition to change the
color used for the command input area. You can change the size of the
command area with the INPUT command.

Message Color and Highlighting

Four color and highlighting attributes can be set for messages:
• Foreground color
• Background color
• Intensity
• Highlighting.

Note: Background color is not supported on most 3270 devices and emulators. In
this case, black is used for the background color.

The color and highlighting attributes for messages can be set in several places:
• In the automation table
• For MVS system messages, in the MVS MPF table
• In installation exits
• In a screen format definition.

Of all of the options listed, the screen format definition takes the lowest
precedence. The following rules of precedence apply:
• MPF table color intensity and highlighting for MVS system messages overrides
  the screen format definition for these attributes.
• Automation table specifications of color intensity and highlighting override the
  following:
  – The MPF table specified color intensity and highlighting
  – Screen format definition of color intensity and highlighting
  – DSIEX02A and DSIEX17 specification of color intensity and highlighting
    (these exits are driven prior to automation).
• Installation exit specifications of color intensity and highlighting override the
  MPF and the screen format definition for these attributes. In addition,
  installation exit DSIEX16 (post automation) can override the color intensity and
  highlighting specified in the automation table.

Each of these presentation attributes can be manipulated independently. For
example, an MVS system message that had a match in the automation table with a
color action would be presented in the intensity and highlighting as specified in
the MPF table, but with the color as specified in the automation table.
Chapter 3. Using the VIEW Command

This chapter documents general-use programming interface and associated guidance information.

The VIEW command processor can be used to display full-screen panels from user-written programs. The VIEW command enables users to design their own panels and control the color and highlighting of panel text.

The VIEW command enables command lists or command processors written in PL/I or C to interact with an operator by means of full-screen panels. The data from the command list or PL/I or C variables can be substituted into the panels.

Creating Full-Screen Panels

To create panels for your operators, define the text and format in a data set or file. The panel source consists of a prologue, followed by text and variables that define the panel to be displayed. Figure 11 on page 30 is an example of the information in the help source file. See "General Help Fields" on page 31 for descriptions of each numbered field in the figure.

If your display consists of a sequence of lines or messages, you may find it easier to use the WINDOW command for your full-screen panel. WINDOW enables you to alter its display and to define or redirect subcommands. For more information, refer to the online help for WINDOW.

NetView provides a number of command lists that use the VIEW command to display full-screen panels. Displaying a new panel by invoking VIEW from a command list requires that you either modify an existing command list or write a new one. When you modify an IBM-supplied command list, first copy it into a user data set and change its name.
General Help Fields

The special characters in the source file, such as the dollar sign ($) and the percent sign (%), are described in "Controlling Color and Highlighting of Fields" on page 35.

1 Prologue

An optional section for programmer comments. Each line of the prologue begins with /* in columns 1 and 2. Comments can only be placed in this section. If comments are displayed in the Help or Option Definitions section, a return code of 83 is sent, and the panel is not displayed. Comments that are displayed after these sections are treated as data.

2 Help

Optional definition of the panel. This field follows the prologue and is coded in the following format:

```
Column 1 15
HELP=helppan comment
```

Note: You can also use HELP CMD='command_text'. See the following description for 3.

3 Option Definitions

An optional list of selections the operator can choose. This list can contain panel names or commands. You can add an optional comment after the
panel name or command. At least one blank must separate the panel name or command from the comment. The list cannot exceed 49 entries. The list is coded in the following format:

```
Column 1 3
n panel_name or CMD='command_text' comment
```

Where \( n \) is the character the operator enters to call the panel or issue the command.

To produce a continuation panel, \( n \) is blank, as follows:

```
Column 1 3
panel_name comment
```

In this case, \textit{panel_name} identifies the continuation panel.

\textbf{Text Indicator}

Three required asterisks separate the prologue, help, and panel definitions from the displayed panel text. These asterisks can be followed by the options listed below, which can appear in any order and must be separated by at least one blank.

- The AT1 option is attribute set 1 for color and highlighting attributes. See Table 10 on page 32 and Table 13 on page 36 for more information.
- The AT2 option is attribute set 2 for color and highlighting attributes. See Table 10 on page 32 and Table 13 on page 36 for more information.
- The XVAR option provides variables that can contain up to 31 characters, including periods. Without this option, variables can contain only 11 characters and cannot contain periods. See Table 10 on page 32 and “Compound Symbols” on page 42 for more information on the XVAR option.
- The OPTROW=optchar option enables you to specify that any row (line) that begins with the character defined by \textit{optchar} is an optional row. The maximum number of optional rows is defined as the number of rows supported by the terminal, minus 24 (which can be zero). Optional rows defined on the panel that go beyond this maximum are not displayed. Also, rows (regular or optional) that go beyond the terminal’s limit are not displayed.

For an optional row, all the characters are shifted left one position to compensate for the \textit{optchar}, and the resulting last position (column 80) is treated as a blank.

See the WINDOW command list (CNME1505) and its View panel, CNMKWIND, as an example of how to use OPTROW.

- The WIDE option enables the entire line width to be used on terminals that support more than 80 columns. When WIDE is specified, panel variables that are the last non-blank specifications on their respective lines are substituted. The variables are not truncated until the end of the line, which is defined by the terminal.

See the WINDOW command list (CNME1505) and its View panel, CNMKWIND, as an example of how to use WIDE.
Table 10. Examples of Using Text Indicator Options

<table>
<thead>
<tr>
<th>Coding</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>*** AT1</td>
<td>• Attribute set 1</td>
</tr>
<tr>
<td></td>
<td>• English</td>
</tr>
<tr>
<td></td>
<td>• 11-character variable names, no periods</td>
</tr>
</tbody>
</table>

When three asterisks are followed by the AT2 option, attribute set 2 is used for color and highlighting. For example:

• *** AT2 for English
• For attribute set 1, use *** or *** AT1

For attribute set 1 and variables as long as 31 characters, use *** AT1 XVAR for English.

See “Controlling Color and Highlighting of Fields” on page 35 for more information on attribute sets 1 and 2.

5 Name
The name of the panel.

6 Heading
The text that describes the use of the panel.

7 Panel Text
Up to 24 lines of text that constitute the displayed panel. See also the OPTROW option described under “Text Indicator” on page 31.

Command list variables can appear anywhere in the panel text. See “Displaying Variables in Source Panels” on page 40 for more information.

8 Message Area
The variable &CNMIMDL displays NetView error messages on line 23 of the panel. If the application has not provided a value for CNMIMIDL, VIEW searches the global dictionaries (task, then common) for a variable named CNMIMxxx, where xxx is the application name provided when VIEW was invoked. If the variable is not found, VIEW searches for CNMIMVIEW in the same dictionaries. Finally, if none of these variables is present, the text from message BNH257I is displayed. The default English text for BNH257I is “TO SEE YOUR KEY SETTINGS, ENTER 'DISPFK'”. The text of message BNH257I can be changed in the message translation table.

See “Using PF Keys and Subcommands with VIEW” on page 55 for a list of the subcommands that can be assigned to PF keys and “Customizing PF Keys and Immediate Message Line” on page 23.

9 Command Line
NetView commands are typed on the command line. In a VIEW command with the NOINPUT option specified, a command line is defined by the tilde (˜) attribute symbol. The &CUR option identifies the cursor position within the command line. Only one input field and only one &CUR option is processed per panel. This option is useful for predefining a command in the input field. Otherwise, the cursor defaults in the following order:

1. The last attribute variable that specified 'UY'
2. The first tilde field, if one is present
3. The first position in the upper-left corner

**Coding the VIEW Command**

Code the VIEW command as follows:
VIEW

- VIEW name1 name2
  - NOINPUT
  - INPUT

Where:

INPUT
Specifications that input values and AID information may be returned to the procedure invoking the VIEW command. INPUT also specifies that cursor location may be received from and returned to the procedure invoking the VIEW command. When you use the VIEW command with the INPUT option, use the UNIQUE command to enforce uniqueness (only one occurrence of the command on the roll stack). See "Using the UNIQUE Command" on page 45 for more information.

name1
Specifications the name (1–8 characters) that is used internally by the NetView program. The first character must be alphabetical. A distinct name must be used for each separately rollable application. For compatibility with prior releases of the NetView program, characters 1 and 3–9 are allowed.

name2
Specifications the name (1–8 characters) of the panel to be displayed.

NOINPUT
Specifications that the VIEW command does not return any information to the procedure that invoked it. NOINPUT is the default. If the panel defines a command line, the NetView program treats input as a command. With the NOINPUT option, there is no need for your command procedure to invoke the UNIQUE command.

See Figure 11 on page 30 for the PF keys provided by the NetView program when you specify NOINPUT.

Usage Notes

- By specifying NOINPUT, you can use a command procedure to display online help panels. See "Chapter 4. Modifying and Creating Online Help Information" on page 63, for more information on how to code help panel hierarchies.
- You can use the VIEW command to display data from messages obtained through TRAP processing immediately upon receipt of the message. Updates are also possible from non-message sources on a timed basis. For more information, see "Dynamic Update Capabilities" on page 56.
- The VIEW command is intended to be used only from a command procedure. If you use the VIEW command in command lists to display a panel, minimum processing should be done between exiting the view and the end of the procedure. Operator input might be inhibited between the time the view is ended and the end of the procedure.

Return Codes from VIEW and BROWSE

Table 11 on page 34 lists and describes the return codes that can be received for the VIEW and BROWSE command. The table also provides a brief description of the action you need to take.
### VIEW Command

#### Table 11. Return Codes from VIEW and BROWSE

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Your Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Trapped messages exist for this task.</td>
<td>Discard or process the trapped messages before calling VIEW or BROWSE.</td>
</tr>
</tbody>
</table>
| 4    | Specified panel not found in CNMPNL1, CNMMSGF, or CNMCMDF data sets (MVS).  
     | Specified panel not found in B book within the LIBDEF search chain (VSE).  
     | Specified panel not found among files with a `filetype` of NCCFLST or a `DLBL` of CNMCMDF or CNMMSGF for the online help facility (VM).  
     | Possible input/output (I/O) error. | Put panel definition in correct data set or file. |
| 8    | Panel definition format not valid; no noncomment lines found. | Correct format of panel definition. |
| 12   | You are not authorized to browse the member. | Ask your system programmer to redefine your authorization. |
| 16   | VIEW command processor invoked with parameters that are not valid. `name1` must be 1 to 8 characters and `name2` must be a valid panel ID. Valid parameters are INPUT, NOINPUT, MSG, NOMSG. | Correct command list to use valid option. |
| 24   | Full-screen command processor is available to OST only. | Do not invoke VIEW from a non-OST. |
| 28   | Logical record length of panel not 80 bytes (VM only). | Change file to logical record length of 80 bytes. |
| 32   | Unrecoverable error resulted from macro call. Error could be that CNMMSGF or CNMCMDF has not been installed for online message or command help. For VM, error could be that the panel specified was not a file with `filetype` NCCFLST. Also, refer to message DWO050I in the NetView log. | Install CNMMSGF or CNMCMDF. For VM, change `filetype` to NCCFLST. Otherwise, call IBM for service. |
| 36   | Unrecoverable internal programming error occurred. Also, refer to message DWO050I in the NetView log. | Call IBM for service. |
| 40   | Browse panel CNMBROWS, which is used for browsing members, was not found. | Put CNMBROWS in correct data set or file. |
| 81   | Panel definition format not valid; no text indicator line found, or more than 49 option definitions found. (See Figure 11 on page 31 for more information.) | Correct format of panel definition. |
| 83   | Panel definition format not valid; comment lines in wrong place. | Correct format of panel definition. |

### Displaying VIEW Return Codes with SHOWCODE

The SHOWCODE command list is used by command procedures to display descriptions of the nonzero return codes returned from the VIEW command.

Code the SHOWCODE command as follows:

```
SHOWCODE
```
**VIEW Command**

Where:

**panelid**
Specifies the name of the panel that the VIEW command attempted to display before issuing the return code. This parameter is only required for return codes 4, 8, 12, 28, 81, and 83.

**rc**
Is the name of the variable that contains the return code for which you want to display a description.

SHOWCODE displays descriptions of the nonzero VIEW return codes as messages. Table 12 shows the return codes and their related message IDs.

**Table 12. Nonzero VIEW Return Codes and Related Message IDs**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>CNM335I</td>
</tr>
<tr>
<td>8</td>
<td>CNM336I</td>
</tr>
<tr>
<td>12</td>
<td>CNM337I</td>
</tr>
<tr>
<td>16</td>
<td>CNM338I</td>
</tr>
<tr>
<td>24</td>
<td>CNM340I</td>
</tr>
<tr>
<td>28</td>
<td>CNM341I</td>
</tr>
<tr>
<td>32</td>
<td>CNM342I</td>
</tr>
<tr>
<td>36</td>
<td>CNM343I</td>
</tr>
<tr>
<td>40</td>
<td>CNM9071</td>
</tr>
<tr>
<td>81</td>
<td>CNM388I</td>
</tr>
<tr>
<td>83</td>
<td>CNM390I</td>
</tr>
</tbody>
</table>

Before issuing SHOWCODE from a command procedure, check to make sure the return code is not zero. See Figure 19 on page 58 for an example that uses SHOWCODE to display error messages from VIEW.

**Controlling Color and Highlighting of Fields**

You can change or add to the color and highlighting of the existing panels. Text color and highlighting in the displayed panel are controlled by attribute symbols or variables. After you code attribute symbols in the source panel, they appear as blanks in the displayed panel.

Scanning for attribute symbols or variables in a particular line occurs only if column 1 contains an attribute symbol or panel variable. Otherwise, the line is displayed as is from the panel description in the default color and without variable substitution.

**Note:** Color and highlighting depend on the kind of terminal you are using.

**Attribute Symbols**

You can specify attribute symbols on the source panel to color or highlight text. Edit the source panel and replace the blank space before the text with an attribute symbol selected from the second column of Table 13 on page 36 or Table 14 on page 36.

Variables are parsed only at the first level. Nested VIEW variables are substituted, but not parsed. Therefore, color attribute symbols that are located in nested variables are displayed as data.

An option specified in the header of a panel determines the set of attribute definitions to use for that panel. If you specify no option (***), use the original set
VIEW Command

(attribute set 1). Use attribute set 2 when you specify the option (** AT2) on the text indicator line of the panel definition. See “View-Based Help” on page 64 for more information on the text indicator line.

Table 13. Set 1 Color and Highlighting Attributes

<table>
<thead>
<tr>
<th>Attribute Set 1</th>
<th>Symbol</th>
<th>Hex Character</th>
<th>Intensity</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>%</td>
<td>X'6C'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed white</td>
<td>)</td>
<td>X'D0'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Underscored white</td>
<td>!</td>
<td>X'5A'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>White</td>
<td>~</td>
<td>X'A1'</td>
<td>High</td>
<td>Input</td>
</tr>
<tr>
<td>Turquoise</td>
<td>$</td>
<td>X'5B'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Underscored turquoise</td>
<td>\</td>
<td>X'E0'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Blue</td>
<td>+</td>
<td>X'4E'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed blue</td>
<td>{</td>
<td>X'C0'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Green</td>
<td>@</td>
<td>X'7C'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Yellow</td>
<td>~</td>
<td>X'5F'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Pink</td>
<td>!</td>
<td>X'6A'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Red</td>
<td>¢</td>
<td>X'4A'</td>
<td>High</td>
<td>Text</td>
</tr>
</tbody>
</table>

Table 14. Set 2 Color and Highlighting Attributes

<table>
<thead>
<tr>
<th>Attribute Set 2</th>
<th>Symbol</th>
<th>Hex Character</th>
<th>Intensity</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>%</td>
<td>X'6C'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed white</td>
<td>)</td>
<td>X'D0'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed red</td>
<td>!</td>
<td>X'5A'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>White</td>
<td>~</td>
<td>X'A1'</td>
<td>High</td>
<td>Input</td>
</tr>
<tr>
<td>Turquoise</td>
<td>$</td>
<td>X'5B'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed green</td>
<td>\</td>
<td>X'E0'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Blue</td>
<td>+</td>
<td>X'4E'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed blue</td>
<td>{</td>
<td>X'C0'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Green</td>
<td>@</td>
<td>X'7C'</td>
<td>Normal</td>
<td>Text</td>
</tr>
<tr>
<td>Yellow</td>
<td>~</td>
<td>X'5F'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Reversed yellow</td>
<td>!</td>
<td>X'6A'</td>
<td>High</td>
<td>Text</td>
</tr>
<tr>
<td>Blinking red</td>
<td>¢</td>
<td>X'4A'</td>
<td>Normal</td>
<td>Text</td>
</tr>
</tbody>
</table>

Displaying Special Attributes

If you want to display a particular symbol that doubles as an attribute within a colored or highlighted row, place a double quotation mark ("" ) in front of the symbol. For example, if you want the left brace (\{) to appear in text, enter "\{ in the source panel. If you want to display a double quotation mark ("" ), enter """". When you use a double quotation mark ("" ) in the source panel, the text following the double quotation mark is shifted to the left in the displayed panel. When the same hexadecimal values for these symbols are coded as part of double-byte character text surrounded with shift-out and shift-in control characters, they are not treated as attributes.
Using the + Attribute: Be careful how you use the plus sign (+) for the color blue. If you want to assign the color blue to a variable defined by the NetView command list language, enclose the plus sign within a pair of single quotation marks as follows:

```
&COLOR = '+'
```

To assign the color blue to the REXX variable `A` so that its contents, `G`, are changed to blue, do the following:

```
A = '+G'
```

Without the pair of single quotation marks, the NetView program interprets the plus sign as a continuation character.

Using the $ and the @ Attributes: Because the $ character and the @ character are often used as data inside a command list or REXX variable, VIEW treats them differently when defined in a panel or in a variable. When in a panel, they are treated as attribute symbols as described in Table 13 on page 36 and Table 14 on page 36. When in a variable, they are treated as data. If the associated attributes are needed inside a variable, substitute the greater than (>) and less than (<) signs as synonyms for @ and $ respectively. Use the respective synonym in your command list. In the following NetView command list example, the AMOUNT field displays the string $1,000 in turquoise and the HEIGHT field displays the string @ 6 feet in green.

```
&AMOUNT = '<$1,000'
&HEIGHT = '>@ 6 feet'
```

Here is what the same example would look like in REXX.

```
AMOUNT = '<$1,000'
HEIGHT = '>@ 6 feet'
```

When they are not used in a variable, the less-than and the greater-than symbols are displayed as characters.

Attribute Variables

Attribute variables are assigned in the command procedure that drives the view panel. An alternative to defining attribute symbols on the panel or within the variable data is to define attribute variables that are associated with panel variables. Attribute variables describe attributes associated with panel variables and their text following on the same line. Using an attribute variable provides a wider range for attribute selection and allows you to define input fields. When you use an attribute variable, the contents of the associated panel variable are not scanned for attribute symbols.

An attribute variable name is formed by concatenating a dollar sign onto the front of the panel variable name. For example, in NetView command list language, the attribute for panel variable `&V1` is defined in a variable called `&$V1`.

In REXX, PL/I, and C, the ampersand (`&`) is not used. For a PL/I or C program, attribute variables must be set using CNMVARS in PL/I or `Cnmvars` in C.

The following is the syntax for the contents of an attribute variable:

```
<attribute-variable>=tv tv tv...
```

Chapter 3. Using the VIEW Command 37
where tv is the type value pair. Multiple pairs of the same type in one attribute variable are allowed. The last pair is accepted and the previous pairs are ignored. The values for type value are as follows:

\[
tv = \text{type value} \\
A = \text{Alarm} \\
AN = \text{No audible alarm} \\
AY = \text{Audible alarm (beep) when panel is presented} \\
\text{Note: The alarm specification only applies to the attribute variable for the immediate message line ($CNMIMDL$).}
\]

\[
C = \text{Color} \\
CB = \text{Blue} \\
CD = \text{The default device color when a color value is not specified} \\
CG = \text{Green} \\
CP = \text{Pink} \\
CR = \text{Red} \\
CT = \text{Turquoise} \\
CW = \text{White or neutral} \\
CY = \text{Yellow} \\
\]

\[
F = \text{Field} \\
FA = \text{Protected; data cannot be entered on displayed panel; FA is the default} \\
FI = \text{Unprotected; data can be entered on displayed panel} \\
\]

\[
H = \text{Highlight} \\
HB = \text{Blinking} \\
HD = \text{The default extended highlighting when a highlighting value is not specified} \\
HR = \text{Reverse video} \\
HU = \text{Underscored} \\
\]

\[
I = \text{Intensity} \\
ID = \text{Dark, nondisplayable} \\
\]
IH  High intensity
IN  Normal intensity; the default when an intensity value is not specified
U = Cursor
UN  The cursor is not placed at the beginning of this field; UN is the default.
UY  The cursor is placed at the beginning of this field. UY specifications for multiple variables cause the last variable specified to be accepted and the previous variables to be ignored.

Notes:
1. If you do not want the cursor to be associated with a particular variable, you can place the cursor in any row and column. Use the VIEWICROW and VIEWICCOL variables in the procedure that calls VIEW with the INPUT option. See "Full-Screen Input Capabilities" on page 43 for more information on the VIEWICROW and VIEWICCOL variables.
2. If you use the VIEWICROW and VIEWICCOL variables and also specify UY on an attribute variable, the cursor is positioned by the attribute variable.
3. If you do not use the VIEWICCOL and VIEWICROW variables or specify a cursor for any attribute variable on a panel, the cursor is placed at the beginning of the first input field.

Use one or more blanks to separate the type value pairs. The following is a NetView command list language example where &V1 is defined as a protected field with high intensity in red. &V2 is defined as a protected field in high intensity, in turquoise, with the cursor placed in the field.

&$V1 = 'FA IH CR'
&$V2 = 'IN IH CT UY IH'

In the following REXX example, V1 is defined as an input variable (unprotected field) with no cursor. For V2, all the defaults are used.

$V1 = 'FI UN'
$V2 = ''

Attributes defined by attribute variables or attribute symbols apply until one of the following is encountered:
• The end of the line
• The explicit placement of an attribute symbol later in the line
• A variable later in the line that has one of the following:
  – A valid attribute variable that specifies new attributes
  – No valid attribute variable, but contains one or more attribute symbols.

Constants or variables defined on a panel can become part of an input field and are updated only when you type over some portion of the input field. When you type on an input field, the entire contents of the input field are assigned to the panel variable.

The first byte of a field defined by a panel variable (the &) is used for attribute specification, and is followed by the contents of the variable. If an attribute
variable corresponds to a panel variable, it takes effect at this first byte even if the panel variable is not found (and is replaced by blanks).

**Note:** If an attribute variable contains a syntax error and the NetView log is active, message CNM944I is written to the log.

---

**Displaying Variables in Source Panels**

When the VIEW command attempts to resolve a variable name coded on the panel definition statement, it first determines whether the variable is a NetView control variable. If the variable name is found, the appropriate NetView control variable is substituted. If the name is not found, VIEW searches for a variable of the same name that is defined by the command procedure invoking the VIEW command. For example, if the variable name &OPID is coded in the panel definition, the value for the control variable &OPID is always substituted rather than the value of a command procedure variable named OPID. If a variable name is not defined to the NetView program or in the calling command procedure, the variable coded on the panel is displayed as a string of blanks.

If the associated attribute variable is not defined, the substituted value of a variable is scanned for attribute symbols. The located attribute symbols are used in controlling color, highlighting, and data fields. If symbols are to be displayed as symbols and not used as attributes then code an associated attribute variable for the variables. This causes the symbols in the data to be treated as data instead of attribute variables.

When an attribute symbol is to be displayed as data, special rules must be followed. See "Displaying Special Attributes" on page 36 and "Attribute Variables" on page 37 for more information on these rules.

**Note:** If the XVAR option is not coded on the panel text indicator line, use only 1 to 11 alphanumeric characters (A–Z and 0–9) for the variable names in VIEW panel definitions. If the XVAR option is coded, variable names can be up to 31 characters long and contain periods. See "Compound Symbols" on page 42 for more information. Alphabetical characters must be in uppercase. Variable names also must conform to any other variable naming conventions set by the language invoking VIEW if the variable is to be referenced by that language. For example, variable names used in PL/I, C, and REXX must start with an alphabetical character.

For the VIEW command to find global variables, the global variables must be referenced by the command procedure prior to executing the VIEW command. Global variables are defined by &TGLOBAL, &CGLOBAL, or GLOBALV in NetView command list language, GLOBALV in REXX, CNMVARS or GLOBALV in PL/I (MVS only), or Cnmvars or GLOBALV in C (MVS only).

**Reference:** Refer to [Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language](#) or [Tivoli NetView for z/OS Customization: Using PL/I and C](#) for more information about global variables.

For the VIEW command to find local or attribute variables when invoked from a high-level language program (MVS only), the variable must be set using CNMVARS in PL/I or Cnmvars in C.
A REXX user can use VIEW to display global variables by issuing a GLOBALV DEFT (or DEFC) instruction to define the global variables before calling VIEW. However, to enable overwriting of global variables by defining them as input fields, a REXX user must do the following:

1. Issue a GLOBALV GETT (or GETC) \texttt{varname} before invoking VIEW.
2. Initialize \texttt{varname} or ensure that it does not have a null value before invoking VIEW.

If 1 and 2 are done, the global \texttt{varname} is displayed and is updated if \texttt{varname} has an attribute variable that makes it an input field. Otherwise, the REXX local \texttt{varname} is displayed and updated. When VIEW accesses a global variable this way, any REXX local variable with the same name is not affected by VIEW.

If you name a NetView control variable (for example, APPLID or OPID) on a VIEW panel, VIEW displays the control variable value and cannot access a REXX local variable with the same name. Control variables cannot be updated.

The following REXX example shows how you can use VIEW to update a global variable:

```rexx
/* */
'GLOBALV GETT XYZ'
IF XYZ = '' THEN
  DO
    XYZ = '
    'GLOBALV PUTT XYZ'
  END
$XYZ = 'FI'
'VIEW NAME1 TESTPANL INPUT'
'GLOBALV GETT XYZ'
SAY XYZ
EXIT
```

If the length of the value assigned to the variable exceeds the length of the variable in the source panel, and if the variable is followed by alphanumeric or special characters (such as !, \, !, @, #, $, %, &, \, +) on the panel definition, the value is truncated. When a variable is followed by characters other than these mentioned (such as a period or a dash), the characters are overwritten.

If the value assigned to the variable contains double-byte text, all the double-byte text must be within DBCS shift-out and shift-in characters. If the panel cannot display all the double-byte text within a pair of DBCS shift-out and shift-in characters, VIEW displays all the text that fits, and displays a period (.) to indicate a truncated character.

For example, if a variable named &DBCSTEXT is defined with a value of “NetView Help Menu” in Kanji, this value may be truncated because the field on the panel is too short, because the operator has scrolled the panel to the right or left, or because an application which uses VIEW has truncated data. For instance, the NetView WINDOW command uses VIEW to handle double-byte character truncation. Here is the hexadecimal representation of the double-byte Kanji characters, showing the text length:

```
....+....1....+....2....+....3....
0494549404545444A440444A4945450
E393635303433333333333333333338F
```

If the panel definition allows fewer than 32 characters for the value of &DBCSTEXT, or if the operator scrolls the text so that fewer than 32 characters can be displayed
VIEW Command

on the panel, VIEW displays all characters that will fit. If VIEW can only display one-half of a double-byte character, it substitutes a period (.) for the displayable part of the character, in the same way that BROWSE handles leading and trailing double-byte text truncation for netlogs. In this example, if the first two bytes were truncated, VIEW would substitute a shift-out (X'0E') for the non-displayable last half of the first double-byte character (X'4399'). If the first three bytes were truncated, VIEW would substitute a period and a shift-out character (X'4B0E') for the entire second double-byte character (X'4356').

If an operator tries to display a VIEW panel that does not have properly defined double-byte shift-out and shift-in pairs, a data stream that is not valid will be sent to the device and unpredictable results, such as the operator being logged off, will occur. Examples of DBCS definitions in which the double-byte shift-out and shift-in characters are improperly matched:

• A greater number of shift-out or shift-in characters (not paired)
• One pair split between two or more variables
• One pair split between a variable and a panel definition
• One pair split across more than one line of a panel.

Compound Symbols

A compound symbol contains at least one period and at least one other character. It cannot start with a digit or a period. If there is only one period, the period cannot be the last character.

The name begins with a STEM (part of the symbol up to and including the first period), which is followed by PARTs of the name (delimited by periods) that are constant symbols, simple symbols, or null. A constant symbol starts with a digit (0–9) or a period. A simple symbol contains no periods and does not start with digits (0–9).

VIEW starts with a compound symbol coded in a panel. Then, VIEW creates a derived variable name by replacing PARTs with their values. VIEW then requests the value of the derived variable for display in the panel.

This example is a small extract from a REXX program:
Implementation Maximum

All HLL and REXX variables are restricted to 31 characters when the panel text indicator has the XVAR option; otherwise, the limit is 11. NetView command list language does not support compound variables or variable names longer than 11 characters. It is important to note the differences from the way REXX displays the string and the way VIEW displays the string.

Usage Notes

1. VIEW does not support mixed case symbols defined in REXX. For example, a.c in Figure 12 is displayed as 5 in VIEW, but REXX will display it as Bill.
2. VIEW displays blanks for the value of the compound variable if the final value is undefined, null, or not valid.
   In Figure 12, a.a, c.a, and x.d.4 are displayed as blanks in VIEW.
3. VIEW does not distinguish unknown compound variable PARTs and those with null values. When a PART is null or unknown, its NAME is used in building the compound variable name. In Figure 12, VIEW searches for $X.D.4, not $X..4, and thus cannot find Annie.
4. Enter *** XV AR in the text indicator section of your panel definition in order to use compound variables. See "Text Indicator" on page 31 for more information.

Issuing Commands from Command Procedures

When a command is issued directly from a command procedure, the procedure is suspended until that command completes. When the called command is complete and the return code is available, the procedure resumes execution. If the called command is a long-running command, it and the calling procedure form a group that is treated as a unit by the NetView ROLL command (roll group).

Note: The BGNSESS FLSCN command is an exception because it allows a calling procedure to complete before the session begins by using the MINOR option of DSIPUSH. Refer to Tivoli NetView for z/OS Customization: Using Assembler for information about DSIPUSH.
Grouping commands and procedures is beneficial if the intent is to build a hierarchy of related panels, using different procedures to build each one. Grouping commands and procedures is not desirable when executing unrelated commands, such as those received from an operator.

To disassociate an unrelated command from the calling procedure, use the CMD command. To illustrate this, assume that the variable `cmdline` contains an operator's command that was entered on your panel. You can queue the `cmdline` command asynchronously by issuing one of the following in your REXX command procedure:

```
'CMD HIGH ' cmdline
'CMD LOW ' cmdline
```

The HIGH or LOW parameter of the CMD command indicates the priority at which the command should be queued.

**Note:** Issuing the CMD command with the HIGH parameter usually interrupts other processing, allowing the queued command to run.

For example, suppose an operator enters the STATMON command on the command line of your panel. By using the CMD command, you can queue the STATMON command rather than calling it directly. This allows the operator to roll back to your command procedure from STATMON, even though STATMON is not complete. Refer to **Tivoli NetView for z/OS Customization: Using Assembler** for more information about the ROLL function and the NetView online help for more information about the CMD command.

Queuing, rather than calling a command, protects your procedure from any reset condition the queued command encounters.

### Creating a Rollable Component with VIEW

A NetView component is a command or command procedure that controls the terminal’s screen, provides for operator entry of arbitrary NetView commands, and is capable of resuming when such commands are complete. In a command procedure, you can create a rollable component using VIEW to provide the necessary screen control.

If you specify the NOINPUT option, VIEW handles the operator command interface for you. If you specify the INPUT option on your VIEW command, VIEW returns the operator’s input to your procedure in the form of named variables, one or more of which may be treated as a command.

The commands contained in these variables must be in uppercase for the NetView program. PL/I and C command procedures should verify that these command strings are in uppercase before issuing CNMCMDC. The NetView command list language provides the UPPER command for translating the contents of a variable to uppercase. REXX command lists can use the UPPER instruction to ensure that commands are in uppercase.

### Using the UPPER Command

Use the UPPER command to change the contents of the specified variables to uppercase.

The format of the UPPER command is:
UPPER

Where:

variable

Specifies the 1- to 11-character name of the variable to be translated to uppercase. The comma in the repeat separator indicates that you can optionally specify more than one variable name on an UPPER command.

Example:

UPPER CMDLINE
CMD HIGH &CMDLINE

Usage Notes:

1. Do not specify the leading ampersand (&) in front of the variable name.
2. If you specify more than one variable, all variables are translated, even if one of the variables has an error condition (not found or the length is not valid).
3. The UPPER command is provided in the NetView command list language only. A similar function is available to REXX command lists with the REXX UPPER instruction.
4. The UPPER command should not be concatenated with other commands in a command string.

Return Codes: The return codes for this command are as follows:

0   Successful completion of all specified variables
4   At least one variable not found, or at least one variable is not valid
8   At least one variable length not within range
12  At least one variable not found and at least one other variable length not within range
16  Not invoked from a command procedure
20  No variables specified

Using the UNIQUE Command

With the UNIQUE command you can search the roll stack for a component that has a subcomponent with the same member name (for command lists and REXX) or module name (for PL/I and C) as the issuing command procedure. If such a component is found, the UNIQUE command allows only one of the two components to remain on the roll stack, either the issuing component or the older component.

The format of the UNIQUE command is:

UNIQUE
Where:

CANCEL

Specifies to reset (CANCEL) the roll group containing the matching element on the roll stack as the currently running component. CANCEL is the default. (The issuing component remains on the roll stack.)

PROMOTE

Specifies to position (PROMOTE) the roll group containing the matching element on the roll stack as the currently running component.

Usage Notes

1. The UNIQUE command is valid only when issued from a command list.

2. The NetView program allows an operator to start many copies of the same command processor. You might not want more than one copy, as when creating a NetView component. By using DSIPOP or DSIPUSH with the PROMOTE option, assembler programmers guarantee the uniqueness of long-running commands. Using the UNIQUE command guarantees uniqueness in a command procedure.

3. Issuing UNIQUE from your procedure has no effect (and gives a 0 return code) if the current copy of the procedure is the only one active. An active long-running command or procedure is one that is in any stage of its processing but is not yet complete. Active procedures include procedures that are suspended (blocked) by some other long-running command. If another copy of the same procedure exists under the same task, the UNIQUE command affects the entire roll group that includes that copy.

4. When you use UNIQUE with the CANCEL option (the default format), the calling procedure is temporarily suspended while the older copy is given control with a reset condition. The NetView program suppresses the cancellation messages normally issued when a procedure is reset. When the canceled copy of the procedure and any others in its group complete, the issuing copy resumes with the next line after the UNIQUE command. The return code is set to 4.

5. Using the UNIQUE command with the PROMOTE option moves the previous copy of the calling procedure and its roll group to the top of the roll stack, ready to resume when the copy issuing UNIQUE completes. The return code is set to 4. The procedure invoking UNIQUE should exit at this point to allow the promoted procedure to regain control. An exit code −5 is used to let the caller know that it can now regain control.

6. When you use UNIQUE in NetView command list language, code a suppression character (&SUPPCHAR) to suppress unwanted command echoes that occur when the command has an error. Code SIGNAL ON HALT in your REXX procedures to suppress the REXX cancellation message. The HALT subroutine should return a −5 return code. When you code SIGNAL ON ERROR in your REXX procedures, a return code of 4 signals the error label.

7. No special processing is required for the ROLL command. It is issued in the same way as other NetView commands. To be consistent with other NetView applications, set PF6 and PF18 to issue the ROLL command.
8. Parameter synonyms are supported.

9. Parameter authorization restrictions are not appropriate for the UNIQUE command.

10. Upon cancellation of a component, REXX, PL/I, and C command procedures can perform a cleanup.

**Return Codes:** The return codes for this command are as follows:

- **0** The calling procedure is unique.
- **4** A matching procedure was found. Action successful.
- **12** Environment is not valid (not called from a procedure).
- **16** Syntax error, argument is not valid.

---

**Full-Screen Input Capabilities**

The VIEW command can receive the following values from the calling procedure:

- The cursor row position
- The cursor column position.

You specify this information with the INPUT keyword and by coding VIEWICROW and VIEWICCOL in the calling procedure. When the panel is displayed, the cursor is positioned at the location specified by VIEWICROW and VIEWICCOL. If you used an attribute variable to associate the cursor with a variable, that overrides cursor positioning by VIEWICROW and VIEWICCOL. Table 15 on page 48 describes these two variables.

The VIEW command allows the following to be returned to the invoking procedure:

- The contents of multiple input-capable variables on a panel
- The attention identifier (AID) information
- The cursor location
- The number of panel rows put out by the VIEW command
- The number of panel columns put out by the VIEW command.

You specify this information with the INPUT keyword and by coding an attribute variable with the FI type value pair.

When you use the INPUT option, an input field is available only if you defined an attribute variable specifying FI. (See "Attribute Variables" on page 37 for information on the type value pair.)

When the panel is displayed, it contains the variable values that you can modify by typing over them. The modified variables are returned to the invoking procedure when you press the AID key. Table 16 on page 49 describes the AID key and the variables that are set on return to the calling command procedure.
### Table 15. Variables Specified in the Calling Command Procedure

<table>
<thead>
<tr>
<th>Language</th>
<th>NetView Command List Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEWICCOL</td>
<td>&amp;VIEWICCOL</td>
<td>The cursor location (column) set by the command procedure that calls VIEW. Use this variable with VIEWICROW to position the cursor anyplace on the panel. An acceptable value is a positive or negative integer less than or equal to the number of columns on the panel. A positive integer positions the cursor relative to the left side; a negative integer, relative to the right side. If you specify an integer greater than the number of columns on the panel, the cursor is placed at the beginning of the first input field. See Figure 13.</td>
</tr>
<tr>
<td>VIEWICROW</td>
<td>&amp;VIEWICROW</td>
<td>The cursor location (row) set by the command procedure that calls VIEW. Use this variable with VIEWICCOL to position the cursor anyplace on the panel. An acceptable value is a positive or negative integer less than or equal to the number of rows on the panel. A positive integer positions the cursor relative to the top; a negative integer, relative to the bottom. If you specify an integer greater than the number of rows on the panel, the cursor is placed at the beginning of the first input field. See Figure 13.</td>
</tr>
</tbody>
</table>

Assume a panel 80 x 24, and the calling procedure specifies:

VIEWICCOL = 2
VIEWICROW = 2

The cursor is placed in the second column from the left, second row from the top.

VIEWICCOL = -2
VIEWICROW = -2

The cursor is placed in the second column from the right, second row from the bottom.

VIEWICCOL = 82
VIEWICROW = 22

The cursor is placed at the beginning of the first input field because one of the variables specifies a value that is greater than the panel size.

*Figure 13. VIEWICCOL and VIEWICROW Examples*
Table 16. Variables Set on Return to Calling Command Procedure

<table>
<thead>
<tr>
<th>REXX, PL/I, and C NetView Command List Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEWAID &amp;VIEWAID</td>
<td>The AID key used to enter the input.</td>
</tr>
<tr>
<td>VIEWCURCOL &amp;VIEWCURCOL</td>
<td>The cursor location (column) when the AID key is pressed.</td>
</tr>
<tr>
<td>VIEWCURROW &amp;VIEWCURROW</td>
<td>The cursor location (row) when the AID key is pressed.</td>
</tr>
<tr>
<td>VIEWCOLS &amp;VIEWCOLS</td>
<td>The number of columns output by the VIEW command. The default number will be 80 if neither WIDE nor OPTROW is coded on the panel text indicator line, or if the terminal only supports 80 columns. Otherwise, VIEWCOLS is set to the number of columns supported by the terminal. See &quot;Chapter 4. Modifying and Creating Online Help Information&quot; on page 63 for more information.</td>
</tr>
<tr>
<td>VIEWROWS &amp;VIEWROWS</td>
<td>The number of rows (lines) of the given panel that were output by the VIEW command, which is determined by the number of regular data lines in the source panel, the number of optional data lines in the source panel, and the number of rows available on the output terminal. See &quot;Chapter 4. Modifying and Creating Online Help Information&quot; on page 63 for more information.</td>
</tr>
</tbody>
</table>

The contents of the VIEWAID variable are defined as PF1 through PF24, PA1, PA2, PA3, or the ENTER key.

If you press PA1, PA2, or PA3, only the AID (VIEWAID) information is returned to the invoking procedure. The cursor row, column locations, and any input fields defined on a panel are not returned.

Note: If you press the ATTN key on an SNA terminal, VIEW with INPUT/NOINPUT ends.

Figures 14 through 18 illustrate source panels using VIEW with the INPUT option to create a rollable component. Figure 14 and Figure 15 on page 50 show the source panels containing input-capable variables to be replaced. These panels use attributes from attribute set 2 (see Table 14 on page 36).
Figure 16 on page 51 is an example of a REXX command list that invokes VIEW with the INPUT option to display PANEL1. The command list assigns initial values to the VARIN1 and VARIN2 input-capable variables in the source panel. The
command list also returns the AID information and command line input to the caller.

```rexx
/* EXAMPLE: NETVIEW COMPONENT USING THE VIEW COMMAND */
/*****************************************************************************/
SIGNAL ON HALT /* RESUME OLD COPY IF ONE EXISTS */
/*****************************************************************************/
'UNIQUE PROMOTE'
if rc = 4 then EXIT -5 /* -5 will cancel caller if it exists */
SIGNAL ON ERROR /* any nonzero rc other than as a result of the */
/* UNIQUE command is an error */
/*****************************************************************************/
/* set up VAR1 and VAR2 as input capable fields */
/*****************************************************************************/
$VARIN1 = 'FI IN CR HB UN'
$VARIN2 = 'FI IH CG HR UN'
/*****************************************************************************/
/* set up COMMAND as an input command line using an attribute */
/* variable. Also define the cursor to stop at this field. */
/*****************************************************************************/
$COMMAND = 'FI UY'
VARIN1 = 'INITIALIZE 1'
VARIN2 = 'INITIALIZE 2'
Do forever
COMMAND = '00'X /* COMMAND = nullchar (this clears */
/* the command line and provides */
/* for insert capability) */

'VIEW USERAPPL PANEL1 INPUT'
UPPER COMMAND
VAROUT1 = VARIN1
VAROUT2 = VARIN2
SELECT
  When viewaid = 'PF2' then exit /* Quit if PF2 */
  When viewaid = 'PF6' then CMD HIGH ROLL /* Roll if PF6 */
  When viewaid = 'PF8' then call PANEL2 /* Next panel if PF8 */
  When viewaid = 'ENTER' then
    SELECT
      when command = NEXT then call PANEL2
    /* Assume any other input given on command line is */
```

Figure 16. Example of a REXX Command List that Drives a Rollable Component (Part 1 of 2)
Figure 16. Example of a REXX Command List that Drives a Rollable Component (Part 2 of 2)

Figure 17 on page 53 is an example of the first panel created from this command list. See Figure 14 on page 50 for the source for this panel. The variables VARIN1 and VARIN2 are replaced with the actual values INITIALIZE 1 and INITIALIZE 2, respectively. The attribute specification is defined by $VARIN1 and $VARIN2 (see "Attribute Variables" on page 37 for more information).

The following attributes are for VARIN1 where the length of the input field continues until the next attribute symbol is encountered. In this case, the attribute symbol is %.

VARIN1 attributes are as follows:
- Input, tab (unprotected)
- Normal intensity
- Red
- Blinking
- No cursor position.
The following attributes are for VARIN2 where the length of the input field continues until the end of the line.

VARIN2 attributes are:
- Input, tab (unprotected)
- High intensity
- Green
- Reverse video
- No cursor position.

COMMAND attributes are:
- Input, tab (unprotected)
- Position the cursor at the beginning of this field.

Figure 17. Display Panel of Component with Variables Replaced by REXX Command List

Enter a command on the command line OR...
Enter NEXT or press PF8 to view the next panel.

Action==> 
PF2= End
PF6/PF18= Roll PF8=Next

Figure 18 on page 54 shows a second display panel from the command list. See Figure 15 on page 50 for the source for this panel.
Returning Command Line Input

When you specify NOINPUT for the NetView program to start processing at the command line, you should define a tilde (˜) on the panel to be displayed.

The tilde definition defines an input field that is returned to the NetView program as a command. An &CUR coded after the tilde on the same line determines where the cursor is positioned.

The &CUR is useful for predefining a partial command. For example:

`˜ V NET,ACT,ID=&CUR`

coded on a panel displays:

`V NET,ACT,ID=_`

with the remaining ID to be completed by the operator.

If more than one is defined on the panel, the last &CUR is processed and previous ones are ignored. If more than one tilde (˜) is defined on the panel, the first tilde is processed and any subsequent ones are changed to a percent (%) sign.

If you specify INPUT for the NetView program, code the command line as you would code any other input-capable field. Do not use the &CUR and tilde definitions. The procedure that displays the panel issues the commands. See "Issuing Commands from Command Procedures" on page 43 for information on issuing CMD HIGH.
Using PF Keys and Subcommands with VIEW

PF keys and VIEW subcommands are treated differently with the two view options, INPUT and NOINPUT. The following two sections explain the differences.

Using PF Keys and Subcommands with the NOINPUT Option

When you use VIEW with the NOINPUT option, you can define your PF keys using the PFKDEF command. The values you assign can be NetView commands, or VIEW subcommands. The following is a list of the VIEW subcommands; some have the same name as similar NetView commands:

- **Help** Displays the help panel previously coded:
  
  ```
  HELP=helppanel
  ```

- **End** Exits to the originating component.

- **Return** Returns to the last panel from which a selection was made.

- **Top** Returns to the first page of a multipage panel.

- **Bottom** Goes to the last page of a multipage panel.

- **Backward** Returns to the previous page of a multipage panel.
  
  In addition to assigning the Backward subcommand to a PF key, you can also enter the following command on the command line to scroll backward a specific number of pages:
  
  ```
  B n
  ```
  
  Scrolls backwards n number of pages or panels.

- **Forward** Goes to the next page of a multipage panel.
  
  In addition to assigning the Forward subcommand to a PF key, you can also enter the following command on the command line to scroll forward a specific number of pages:
  
  ```
  F n
  ```
  
  Scrolls forward n number of pages or panels.

- **Entry Point** Shows the panel that the operator first saw upon entry to help.

**Reference:** Refer to the PFKDEF command in the [Tivoli NetView for z/OS Administration Reference](https://www.ibm.com/docs/en/netview) for more information.

Using PF Keys and Subcommands with the INPUT Option

When you use VIEW with the INPUT option, you can use settable PF keys defined using the PFKDEF command or you can interpret PF keys in your command list. You need to code the panel definition and parameters differently depending on the option you select.

Using Settable PF Keys

To use settable PF keys with VIEW, complete each of the following steps:

1. In the panel definition, create a variable named CNMIMDL that has no attribute-variable ($CNMIMDL) which makes it an input field. Define the immediate message line by putting &CNMIMDL in column 1 of the line. Do not put anything else on that line.
VIEW Command

If the VIEW application has not provided a value for CNMIMDL, VIEW searches the global dictionaries (task, then common) for a variable named CNMIMxxx, where xxx is the application name provided when VIEW was invoked. If this variable is not found, VIEW searches for CNMIMVIEW in the same dictionaries. This is similar to the way keys are set for VIEW applications. Finally, if none of these variables are present, the text from message BNH257I is used.

2. In the panel definition, create a variable named CNMCMDL that does have an attribute-variable ($CNMCMDL) which makes it an input field. CNMCMDL defines the command area.

3. Optionally, create another variable named CNMDIMD to define a default immediate message. This message is displayed by NetView whenever the CNMIMDL message has been displayed and there are no other immediate messages. If you do not create CNMDIMD, NetView defaults it the same way it defaults CNMIMDL.

All these variables support attribute ($) variables.

For example, you might call VIEW with an error message in CNMIMDL and a default message in CNMDIMD, with $CNMIMDL set to CR and $CNMDIMD set to CG. The error message will be displayed in red, but if the user presses a RETRIEVE key or delay-type key, for example, the red message is replaced by the default message, in green.

The REXX command WINDOW is a good example of coding VIEW panels to set PF keys. Enter BROWSE WINDOW to see the REXX source for this command.

Notes:
1. VIEW-input applications that do steps 1 and 2 always have their VIEWAID variable set to ENTER after invoking VIEW, because other keys are converted as if the user typed the command text and pressed ENTER.
2. The &CNMIMDL variable is nulled out when control is returned to the command list from VIEW, if VIEW detected that the immediate message area was overwritten by NetView after the VIEW panel was output (for example, by an immediate command entered by the operator).
3. The special variables CNMIMDL and CNMDIMD are supported in VIEW-noinput as well as VIEW-input. CNMCMDL only has special meaning in VIEW-input.

Dynamic Update Capabilities

The VIEW command enables you to dynamically update the contents of the panel being displayed. The updates can be controlled by:

- The calling procedure
  When VIEW detects that a message TRAP is satisfied, VIEW returns control to the calling procedure to allow the update of local variable values displayed on the VIEW panel. VIEW refreshs the display with the new values when control is returned to VIEW using the RESUME command.

- Any automation or procedure running on the same task
  If the variables named on your VIEW panel are not defined by the calling procedure, VIEW attempts to read values from task global variables. For more information, refer to the online help for the GLOBALV command and the PIPE
VAR stage. Values of task global variables can be updated by any procedure
called on the same task (same operator ID) and VIEW immediately refreshes the
display when the procedure completes.

- Any procedure in NetView

If the variables named on your VIEW panel are not defined by the calling
procedure and do not exist as task global variables, VIEW attempts to read
values from common global variables. For more information, refer to the online
help for the GLOBALV command and the PIPE VAR stage. Any procedure in the
NetView program can update the values of common global variables; however,
VIEW refreshes the display only when an event (such as receipt of a message)
occurs at the task that started VIEW.

While a panel is displayed, automation from timers, messages, or alerts can drive
command procedures that update some of the variables substituted into the
displayed panel. Any processing under the OST where the panel is displayed
causes a dynamic update of the panel with new values for any variables that have
changed.

To make information on the panel easier to see, and make it easier to enter
information on the panel while a panel is dynamically updated, assign values to
attribute variables for all variables on the panel that can be changed dynamically.
This enables VIEW to send only the updated information to the screen without
rewriting the entire screen for each update.

When VIEW detects certain changes to common, task, and local variables or their
associated attribute variables, VIEW must rewrite the entire panel.

If the entire screen is redisplayed, changes typed by the operator on the screen
being redisplayed are lost. Following is a list of these changes:

- The attribute variable for a given data variable has changed to indicate that a
  field has been changed from protected to unprotected or vice-versa.
- An attribute variable for a given data variable now has a valid value. It either
did not exist or it had a value that is not valid.
- An attribute variable for a given data variable now has no value or a value that
  is not valid. It previously had a valid value.
- The value for a data variable has changed, and a valid attribute variable is not
  associated with the data variable.

To continue processing of the VIEW command after variables used by the
displayed panel are updated, use the RESUME command.

**Sample of Panel Updating**

The following figures show the dynamic updates of the contents of a panel.

Figure 19 on page 58 is an example of a command list called RESDYN which is
shipped as part of sample CNMS1101. RESDYN uses the RESOURCE command
output as data to be displayed in a panel using the VIEW command. The
displayed data is updated on a time interval that you specify when calling the
command list. The default time interval is 10 seconds.
VIEW Command

```/* -------------- Dynamic Resource Display (option 12) -------------- */
/* A demonstration of using VIEW and TRAP to dynamically update a */
/* full screen display. We use the SPILL option of pipe's KEEP */
/* (new for V5) to create a message after the specified refresh */
/* interval. This message is TRAPPED, causing VIEW to return */
/* control to this procedure WITHOUT removing the displayed panel. */
/* The 'RESUME', below is a REINVOCATION of the original VIEW!!! */
/* */
/* Note that the first call to "fillVars" passes an extra little */
/* bit of pipe to the subroutine. The purpose is to get the first */
/* word of the second data line (STC name) for the panel. */
/* */
/* ------------------------------------------------------------------ */
resdyn:
interval = 10 /* refresh at 10 second intervals */
privMsgID = 'CNMRESDYN' /* special purpose "msgid" for trapping */
getSTC = '% STC:|DROP 1|TAKE 1|EDIT W1|VAR JBN'
Call fillVars getSTC /* set local variables with data from RESOURCE */
'TRAP AND SUPPRESS MESSAGES' privMsgID /* TRAP our special message */
'PIPE VAR privMsgID | KEEP RESDYN' interval 'SPILL' /* make msg later */
'VIEW RESDYN CNMSRESP'
DO WHILE (rc = 2) /* RC indicates "message trapped"? */
'MSGREAD' /* just getting msg off trap queue */
CALL fillVars
'PIPE VAR privMsgID | KEEP RESDYN' interval 'SPILL' /* make msg later */
'RESUME' /* Invoke VIEW, previously suspended */
/* NOTE: RC, at this point, is RC from VIEW, which was resumed. */
END
'pipe hole | keep resdyn' /* empty safe created above */
return

/* ----------- Obtain data for RESDYN display (option 12) ----------- */
/* Notice that the stem variable "out." is in our local variable */
/* dictionary. VIEW could always read these value; new for V5R1, */
/* we will have an opportunity to update them while VIEW is active. */
/* *------------------------------------------------------------------*/
fillVars:
ARG xtraStg /* use extra first time only */
'PIPE (NAME RESDYN END %)',
' NETVIEW RESOURCE',
'SEPARATE DATA', /* No use for DS1386I title line */
'STC: FANOUT', /* MAYBE need extra copies */
' EDIT SKIPTO /=/ 2.* STRIPL 1 ',
' COLOR WHITE',
'$STEM OUT.',
xtraStg
TM = date() time()
$TM = 'CB HR'
return```

Figure 19. Example of a REXX Command List to Update a Panel

Figure 20 is an example of the output from the RESDYN command list.
VIEW manages the PF keys and the command line without the intervention of the RESDYN command list.

Changing Colors in Browse

The template shown in Figure 22 on page 60 is used when browsing members of a partitioned data set. Note the various applications of the color attributes shown in Table 13 on page 36 and Table 14 on page 36. The characters %, $, ¬, and + each assign a specific color to the screen area immediately following their positions. To change a color area on the screen, you need only change the color attribute. You can only change existing attribute fields; changing any other field can result in errors when browsing.
Figure 22. BROWSE Command Panel Definition Showing Color Attributes (Part 1 of 2)
Figure 22. BROWSE Command Panel Definition Showing Color Attributes (Part 2 of 2)
Chapter 4. Modifying and Creating Online Help Information

The NetView program contains a help facility, which has two types of help information.

The first type of help is view-based help, which is displayed by using the VIEW command. The second type is window-based help, which is displayed by using the WINDOW command.

This chapter explains how you can add, delete, or modify help information and is arranged in the sequence you use to accomplish this. The sequence follows:

1. Locate the help source file.
2. Copy and change the source file.
3. Store the copy.
4. Display the help to test your changes.

Locating Help Source Files

Source files define the panel contents that are displayed.

Help information is contained in a separate file and is shipped as a member in a partitioned data set (PDS). English help source files are stored in the NETVIEW.V5R1M0.CNMPNL1 data set.

Notes:

1. Japanese help source files are stored in the NETVIEW.V5R1M0.SCNMPNL2 data set.
2. Copies of the command and message help are stored on the Web Server. If you customize the command and message help in NetView data set members, you may want to make the same changes to the Web Server files.

Verify that your organization has not changed the library name.

Before you create a new help source, try to locate an existing online help that is similar to the one you want to create. Generally, when you have a help source file displayed, the file name is in the top left corner.

For command help information, you can locate the source file you want to change by browsing the HELPemap. Window-based help files are prefixed with the < character. See HELPemap Facility on page 68 for more information on the HELPemap. Help information for groups of messages is stored as members of the PDS, one member for each group. The member name is determined by truncating the message ID prior to the last numeric digit. For example, help for messages DSI001I and DSI002I are stored in member DSI00. Help for message EKGV68001I is stored in member EKGV6800.

If a message or command help panel is currently being displayed, you can use the SHOWDATA command to locate the source file. Figure 23 on page 64 displays the information returned after entering SHOWDATA on the command line.

Note: In Figure 23 on page 64, the following are true:

1. The panel is located in member EUYCLIST of the CNMPNL1 data set.
2. The !+! listed in the response from the SHOWDATA command is generated by special processing from the help search procedure and can be ignored.

View-Based Help

The source file contents include the text of the displayed panel and the definition statements associated with the panel. A definition statement includes:

- A prologue
- The help panel name
- The continuation panel name
- A list of associated help panels

To view the source file for a View-based help panel, enter:

```
BROWSE CNMPNL1.panelid
```

Where `panelid` is the name that is displayed in the upper-left corner of the source for the help. For additional information, see "Creating Full-Screen Panels" on page 29.

Window-Based Help

Figure 24 on page 65 is an example of the source format of the Window-based help information. Descriptions of each numbered field follow the figure.
## Prologue

An optional section for programmer comments.

---

### Repeat and Return

#### Syntax

<table>
<thead>
<tr>
<th>Command or Operand</th>
<th>Synonym</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPEAT</td>
<td>R or RFIND</td>
</tr>
<tr>
<td>RETURN</td>
<td>RET (for BROWSE, HELP, HELPDESK, STATMON, and VIEW)</td>
</tr>
</tbody>
</table>

**Note:** The command facility has no synonym for RETURN.

---

### Purpose of Command

The `REPEAT` command reissues the last `FIND` command while you are browsing the network log or a member of a partitioned data set. Since this command is sensitive to the current position of the cursor, it is normally entered using a PF key.

By repeatedly pressing the PF key set to REPEAT, you can find successive occurrences of a specified character string. After the first occurrence of a character string has been found, the REPEAT key will find the next occurrence. After the last occurrence of a character string has been found, the REPEAT key can be used to continue the search, wrapping around from the bottom line to the top line (or from the top line to the bottom line if the `FIND` command included the PREV parameter.)

The `RETURN` command returns you to the previous component or the last selection panel that you used.

You should not issue this command from a command list.

---

### Restrictions

...
Modifying Online Help

2 Message or Command
The message or command to which the text applies. If the help information is for a command that can be used in more than one component, the command name is prefixed with the component name. Command names must be preceded by 14 equal (=) signs and a blank space.

3 Message or Command Help Title
The title of this help source file.

4 Tags
Information can be presented in different ways. These can include:
- :H2. is used to highlight command names.
- :XMP. and :EXMP. are used to surround examples.
- :IF DTYPE=PANEL followed by :ENDIF marks a section that is shown when HELP presents a full-screen display.
- :IF DTYPE=MSGS followed by :ENDIF marks a section that is shown when HELP presents a line mode display. This occurs when HELP is called at an autotask or when full-screen displays are otherwise unsupported.
- :LINK. is used to move from one topic to another. The :LINK. tag must be in uppercase and begin in column one; it precedes the display line to which it pertains. This line becomes a tab stop and is highlighted by WINDOW. If more than one line of text is to be highlighted for linking, the :LINK. tag must precede each line. See the example coding in Figure 25 on page 67.

The operator makes a selection by placing the cursor on the line or by issuing a FIND command that selects the line. Optionally, you can designate a keyword that the operator can type to issue the command. The keyword is enclosed in parentheses immediately following the :LINK. tag.
- :CMD. is used to precede a command that can be executed immediately when that line is selected. The command line can contain variable text (for example, HELP msgno) that the operator can overlay with specific data, then press the ENTER key to execute the command. The :CMD. tag has an end tag, :ECMD., and must follow the line of command text. Both :CMD., and its end tag must be in uppercase and begin in column 1.

A portion of EUYSLIST is shown in Figure 25 on page 67 to show how the :IF DTYPE and :LINK. statements are coded.
Copying and Changing Help Source Files

Before you create a new help source file, try to locate an existing online help file that is similar to the one you want to create. See “Locating Help Source Files” on page 63.

If you find a comparable panel, copy it using a screen editor. Change the panel by typing over the existing text or by adding text. If you cannot find a similar online help file, use a screen editor to build a new one.

If you want to modify or create a help source file while the NetView program is running, define your panel data set without secondary extents. Otherwise, a panel can be filed in a new extent, requiring that you close and restart the NetView program to use the panel.

The conventions for structuring a new panel are the same as those for modifying an existing panel. All help source files must have a fixed-length blocked record format and a logical record length of 80 bytes (RECFM=FB, LRECL=80), unless you are using a fully qualified data set name listed in the HELPMAP. See “HELPMAP Facility” on page 68 for more information. Null characters are also counted within this 80-byte record. In addition, you might need to change a command list or another panel that is affected by your new panel.

You can customize the HELPDESK to include topics specific to your installation. NetView provides a template file, CNMHDSKU, that can be edited to create these topics.

1. Add the new topics to CNMHDSKU.
2. Add the new topic identifiers to the table of contents in file CNMHDSK0.

Note: If you want to customize any of the existing HELPDESK files (CNMHDSK1–CNMHDSK9), put the information in a separate file and use the %INCLUDE statement. Otherwise, that information will need to be added each release.
Modifying Online Help

After creating or modifying a help file, store it in a data set concatenated to DDNAME CNMPNL1. As an alternative, you can also modify the panel with an SMP USERMOD. See "Storing Help Source Files" for more information.

Storing Help Source Files

Ensure your panel names do not use the same prefixes used by NetView-supplied panel names.

Store all help source files that you create or modify. Two methods for storing help files follow:

- Concatenate the user partitioned data set that contains the modified help file to the CNMPNL1 DD statement in the NetView startup procedure before the data set NETVIEW.V5R1M0.CNMPNL1. If the Support Center modifies the panel, those changes will not be added to your help file.
- Include your modified help file into a System Modification Program (SMP) USERMOD and apply the USERMOD so that SMP stores the modified panel in NETVIEW.V5R1M0.CNMPNL1. SMP automatically notifies you of any future changes that the Support Center makes to the panel you modified. For more information on how to use an SMP USERMOD, refer to the System Modification Program library.

Notes:
1. The default data set for the Japanese version of the product is NETVIEW.V5R1M0.SCMPNL2.
2. English help source files are stored in the NETVIEW.V5R1M0.CNMPNL1 data set. Verify that your organization has not changed the library name.

HELP MAP Facility

The HELP command scans the HELPMAP for the required command help member name using the arguments as search targets. HELP uses the arguments in the following manner:

- With no arguments
  When you enter HELP without supplying any arguments, you get component-level HELP for the component you are in.
  If the target arguments are not found in the table, HELP searches for a pair of parentheses () and uses the associated panel name.
- With one argument
  When one argument is supplied, HELP attempts to resolve the argument as a command synonym, if possible.
- With two or three arguments
  When two or three arguments are supplied, the search target is constructed by concatenating the arguments with commas. For example:
  
  HELPMAPU is a specific HELPMAP for user-defined help files created for commands. A %INCLUDE statement contained in HELPMAP embeds HELPMAPU that provides the mapping for those help files created by the user.

Note: Do not map user-defined help files to HELPMAP. These changes interfere when IBM applies maintenance to HELPMAP.
A portion of CNMHELPF is shown in Figure 26 to show how the help names are listed. Those that are prefixed with the < character are window-based help files; others are view-based help files.

You can add fully qualified data set names within single quotes to the HELPMAP. See the following example as a guide:

<USER.CNMPNL1(MYCMDHLP) MYCOMMAND

Figure 26. Example of the HELPMAP

You can add fully qualified data set names within single quotes to the HELPMAP. See the following example as a guide:

<‘USER.CNMPNL1(MYCMDHLP)’ MYCOMMAND

Displaying New Help Panels

After you have created a new help panel, use the HELP command to view the new panel, and any associated commands or panels, to ensure that they display properly.
Chapter 5. Customizing Session Monitor Sense Descriptions

NetView provides help for VTAM sense codes through the session monitor SENSE command. You can request help for either 2-byte or 4-byte sense codes. The information used to present explanations for the sense codes is stored as a set of members in the DSIPARM data set. You can customize these members or include additional members to include help for sense codes that have additional meaning for a specific application.

Session Monitor Sense Codes

The session monitor sense code descriptions are stored as DSIPARM members named CNMBnnn, where nnn is the first three hexadecimal digits of the 2-byte and 4-byte sense codes described in the member. For example, help for sense codes 08B2 and 08B60001 is stored in DSIPARM member CNMB08B. The CNMB08B member shipped with the NetView product is shown in Figure 27 on page 72.

The general conventions are:
- The descriptions are first grouped by the leftmost two bytes of the sense code, using a separator of $$$KEY xxxx???? where xxxx is the hexadecimal value of the leftmost two bytes. The description of the 2-byte sense code xxxx (or 4-byte sense code xxxx0000) follows this separator.
- Extended sense code descriptions, identified by the rightmost two bytes of a 4-byte sense code, are grouped using a separator of $nnnn where nnnn is the hexadecimal value of the rightmost two bytes. The extended description follows this separator.
- Text descriptions must be contained in columns 1–57 of the DSIPARM member. This text is not DBCS-enabled.

Note: Any modifications you make to existing DSIPARM CNMBnnn members may be replaced by maintenance or another release of the NetView product. You can update the comments at the beginning of the DSIPARM CNMBnnn members to document your changes, and store any members you create or modify in a data set concatenated before the NetView-supplied DSIPARM data set. This helps keep your modifications from being overlaid by subsequent maintenance or product changes.
Customizing Session Monitor Sense Descriptions

Following are some examples of adding and modifying sense code description members in DSIPARM:

- To add additional help for sense code 08B2 or 08B20000, change the NetView-supplied help as follows:

```plaintext
***********************************************************************
* 5697-B82 (C) COPYRIGHT IBM CORP. 1986, 1997 *
* DESCRIPTION: SAMPLE -- SENSE CODES *
* CNMB08B CHANGED ACTIVITY: *
* CHANGE CODE DATE DESCRIPTION *
* ----------------------- --------------- *
***********************************************************************

$$$KEY 08B2
Data transmission failure: the data transmission between an application program in an SNA MS entry point and an application program in a subentry point was incomplete, causing abnormal termination of the function. Bytes 2 and 3 following the sense code contain sense code specific information.

$0000
No specific code applies.

$0001
A time-out has occurred while waiting for transmission of data between the two application programs. For example, a service processor has timed out while waiting to receive data from the main processor.

$0002
A time-out has occurred while waiting for transmission of data between two applications.

$$$KEY 08B5
Network Node Server Not Required: Sent by an APPN end node control point to a network node control point (1) to deactivate CP-CP sessions with the NNCP, or (2) to reject a CP-CP session BIND from the NNCP. The end node no longer requires network node services from the receiver.
Note: This sense data value is carried within the X’35’ control vector on an UNBIND(Type = X’01’) for case (1) above, or on an UNBIND(Type = X’FE’) for case (2).
VTAM Hint: A possible cause of this error is that the Network Node Server for the CP-CP session attempt is not in the Network Node Server List.

$$$KEY 08B6
CP-CP Sessions Not Supported: Sent by a network node control point to reject a CP-CP session BIND from another APPN control point; support for CP-CP sessions on that TG was removed since the time when the TG was first activated.
Note: This sense data value is carried within the X’35’ control vector on an UNBIND(Type = X’01’). Bytes 2 and 3 following the sense code contain sense-code-specific information.

$0000
No specific code applies.

$0001
During link activation on a switched link, it was discovered that the partner node does not support CP-CP sessions on this TG.
```

Figure 27. CNMB08B Sense Code Help

---

**Examples**

Following are some examples of adding and modifying sense code description members in DSIPARM:

- To add additional help for sense code 08B2 or 08B20000, change the NetView-supplied help as follows:
Customizing Session Monitor Sense Descriptions

Data transmission failure: the data transmission between an application program in an SNA MS entry point and an application program in a subentry point was incomplete, causing abnormal termination of the function. Bytes 2 and 3 following the sense code contain sense code specific information.
The SNA MS entry points currently defined are SYSTEM1 and SYSTEM2.

Note the two lines of help information added for this installation-specific sense code.

- To add help for a new sense code 08B3 or 08B30000, add the following information immediately after the NetView-supplied information for sense code 08B2. For example:
  $$\text{$$$KEY 08B3????}
  \text{This sense code is generated by application XYZ when a failure occurs between components of the application.}$$

Note the two lines of help information added for this installation-specific sense code.

- To add help for a new sense code 08B60002, add the following information immediately after the NetView-supplied information for sense code 08B60001. For example:
  $$\text{0002}
  \text{During link activation on a switched link, it was discovered that the partner node does not permit sessions with this partner.}$$

Note the three lines of help information added for this installation-specific sense code.

- To add help for a new sense code 08C1xxxx, create a new member in DSIPARM named CNMB08C, and include the following statements:
  $$\text{$$$KEY 08C1????}
  \text{This sense code is generated by application ABC when a failure occurs in a component of the application. The third and fourth bytes of the sense code identify the failing component ID.}$$

Note the four lines of help information added for this installation-specific sense code.
Customizing Session Monitor Sense Descriptions
Chapter 6. Customizing Hardware Monitor Displayed Data

This chapter describes how to modify the presentation of generic and nongeneric alerts. In prior releases of NetView, Recommended Action panels, Event Detail panels, and alert messages were stored at the host. Each nongeneric alert had a unique set of panels and messages. Many of these remain in the current release of NetView. With generic alerts, generic alert code points are used to dynamically build the hardware monitor panels.

This chapter describes how to do the following:
• Modify the text of nongeneric Recommended Action and Event Detail panels
• Modify nongeneric alert messages
• Overlay recommended action numbers from a generic alert
• Control the use of color and highlighting for hardware monitor panels
• Include user-defined errors, such as creating and modifying generic code points or adding resource types to the hardware monitor

Note: Color maps for hardware monitor help panels and command description panels are available only in prior releases of NetView.

If your panels or alert messages have been translated into a language that requires double-byte characters, take care to preserve the integrity of the double-byte character set (DBCS) strings.

Modifying Hardware Monitor Nongeneric Panels

Recommended Action panels and Event Detail panels are defined for event conditions that are not based on generic alert records. If several event conditions use the same Recommended Action panel or Event Detail panel, the panel is physically defined under a single name, the actual panel name. Any other name under which the actual panel can be displayed is the panel alias. Determining whether the panel name is an actual name or an alias is the first step in modifying panel text.

You can make changes to the panel text, and these changes are reflected in all its aliases. You can also make changes to a panel alias, resulting in the creation of a new panel under the former alias name.

Determining a Panel Name

To determine a panel name and whether it is a panel name or an alias, you must know the event associated with the text you want to change and then identify a resource for which the event is logged. Use the following steps as a guide to help you determine the type of name:
1. To identify a resource, display the Alerts-Static, Alerts-History, or Most Recent Events panel.
2. Enter sel# C, where sel# is the selection number on the panel of the event associated with the text you want to change. Message BNJ962I displays a 5-digit code associated with the event. If message BNJ378I is displayed, the event is generic and stored panels are not associated with the event.
   If you receive a product ID and alert ID rather than a 5-digit code, the associated record is a generic alert. Generic alerts do not have unique prestored
panels in the hardware monitor. See "Using NMVT Support for User-Written Programming" on page 90 for more information on generic alerts.

3. Examine the 5-digit code, \textit{xxxyy}, that NetView returns. The variables are described as follows:

\begin{itemize}
  \item \textit{xxx} is the NetView-designated product code, or block ID, for the resource.
  \item \textit{yy} is an individual panel identifier.
\end{itemize}

4. Determine which panel contains the text you want to change, as follows:

\begin{itemize}
  \item For a Recommended Action panel, the panel name (or panel alias) is \textit{BNIxxxyy}, where \textit{xxx} and \textit{yy} are the codes you identified in step 3.
  \item For an Event Detail panel, the panel name (or panel alias) is \textit{BNKxxxyy}, where \textit{xxx} and \textit{yy} are the codes you identified in step 3.
  \item Determine whether \textit{BNIxxxyy} or \textit{BNKxxxyy} is an actual or alias panel name:
    \begin{itemize}
      \item Use an editor, such as ISPF/PDF to examine the directory listing of panel names. This listing is in the NetView-provided partitioned data set (PDS) named NETVIEW.V5R1M0.BNJPNL1. The word \textit{alias} is displayed to the right of panel names that are aliases.
      \item See the appropriate section of this book for the action you want to perform: "Changing Panel Text" on page 78, "Changing from Alias to Actual" on page 78, "Deleting an Actual or Alias" on page 79, or "Adding an Actual or Alias" on page 79.
    \end{itemize}
\end{itemize}

\textbf{Figure 28} is an example of a BNJBLKID table.
Figure 29 on page 78 is an example of a BNJALxxx table.
Changing Panel Text

If BNIXxyy or BNKxyyy is an actual panel name (not an alias), follow these steps to change the panel wording. BNIXxyy panels must contain exactly 14 noncomment lines; BNKxyyy panels must contain exactly seven noncomment lines. Comment lines contain an asterisk (*) in column 1.

1. Use an editor, such as ISPF/PDF, to edit the PDS member containing the panel. The PDS name is NETVIEW.V5R1M0.BNJPNL1 (unless it is changed during installation), and the member name is the same as the panel name.

2. Save the changed member.

The changes apply to all event conditions that use the panel or any of its aliases.

Changing from Alias to Actual

If you want to make a panel that now appears under an alias into an actual panel, follow these steps:

1. Use an editor, such as ISPF/PDF, to edit the PDS member containing the panel alias. The PDS name is NETVIEW.V5R1M0.BNJPNL1 (unless it is changed during installation), and the alias member name is the same as the panel name.

2. Save the changed member. TSO converts the panel alias into an actual panel.

Figure 29. Sample BNJALxxx Table
Customizing Hardware Monitor Displayed Data

A new actual panel is created under the name that was formerly the alias.

Reference: For more information about MVS utilities and JCL, refer to the MVS/Extended Architecture library.

Deleting an Actual or Alias
To delete an actual or alias panel name, do one of the following:

- Delete the PDS member containing the actual or alias panel name. The PDS name is NETVIEW.V5R1M0.BNJPNL1 (unless it is changed during installation), and the member name is the same as the panel name.
- Use the utility IEHPROGM. For example, to delete aliases BNK04B2E and BNK04B2F using this utility, you could code the following:

```
//DELMEBR2 JOB MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DS1 DD VOL=SER=vsnum,DISP=SHR,UNIT=device_type
//SYSIN DD *
  SCRATCH VOL=device_type=vsnum,DSNAME=panel_dsname,
    MEMBER=BNK04B2E
//STEP2 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=A
//DS1 DD VOL=SER=vsnum,DISP=SHR,UNIT=device_type
//SYSIN DD *
  SCRATCH VOL=device_type=vsnum,DSNAME=panel_dsname,
    MEMBER=BNK04B2F
/*
```

In this example, device_type is the device type, vsnum is the volume serial number on which the data set resides, and panel_dsname is the name of the data set containing the panels.

Reference: For more information on MVS utilities and JCL, refer to the MVS/Extended Architecture library.

Adding an Actual or Alias
If you want BNixxyy or BNKxxxyy to be a new (or replacement) panel name or alias, follow these steps:

- Enter a new panel using an editor, such as ISPF/PDF, and copy an existing panel that is similar to the desired panel. Then, change the copied panel.
- Add the new panel name or an alias, using the utility IEBUPDTE.

For example, to add BNK04B2E as an alias of BNK04B2A using IEBUPDTE, code the following:
Customizing Hardware Monitor Displayed Data

In this sample, panel_dsname is the name of the data set where the panel is stored, and vsnum is the volume serial number on which the data set resides. Although the sample defines only one new alias, up to 15 aliases are valid.

Reference: For more information on MVS utilities and JCL, refer to the MVS/Extended Architecture library.

Nongeneric Alert Messages

To change the Event Description: Probable Cause text of any selection on an Alerts-Static, Alerts-History, Alerts-Dynamic, Event Detail, or Most Recent Events panel that is not associated with generic alerts, follow these steps:

1. Determine the event of the associated text and identify a resource against which the event is logged.
2. For the resource identified in Step 1, display the Alerts-Static, Alerts-History, Alerts-Dynamic, Event Detail, or Most Recent Events panel.
3. Enter sel#, where sel# is the selection number of the event associated with the text you want to change. Message BNJ962I displays a 5-digit code associated with the event. If message BNJ378I is displayed, the event is generic. If you receive a product ID and an alert ID rather than a 5-digit code, the associated record is a generic alert. Generic alerts do not have unique prestored Event Description: Probable Cause text messages in the hardware monitor. See “Using NMVT Support for User-Written Programming” on page 90 for more information on generic alerts.
4. Examine the following 5-digit code, xxxyy, that NetView returns.
   
   
   xxx Is the NetView-designated product code, or block ID, for the resource
   yyyy Is an individual hexadecimal panel identifier
5. Use an editor such as ISPF/PDF to retrieve and edit the CSECT that contains the text you want to change. The name of the CSECT is BNJVMxxx (PDS member in NETVIEW.V5R1.BNJSRC1), where xxx is the block ID you identified in Step 4.
Customizing Hardware Monitor Displayed Data

6. Locate the message text within BNJVMxxx. The message number for this text is the decimal equivalent of \textit{yy}, where \textit{yy} is the hexadecimal identifier you determined in Step 4.

7. Change the assembler language macro DSIMDS.

\textbf{Reference:} For the syntax of DSIMDS, refer to \textit{Tivoli NetView for z/OS Customization: Using Assembler} for the text you want to change.

8. Save the changed CSECT.

9. Reassemble the CSECT, and link-edit the CSECT into the load module of the same name.

\section*{Using the ACTION Command List}

You can use the ACTION command list to get more information on a recommended action that is displayed in the hardware monitor. See \textit{Chapter 4 Modifying and Creating Online Help Information} on page 63 for information on how to modify the Action Help panels displayed by the ACTION command list. \textit{Dnnn}, \textit{Ennn}, and \textit{Innn} are recommended action numbers found on the Recommended Action panels. \textit{Rnnn} numbers are actions found on the resolution action panel. The following describes what the ACTION command list displays for recommended action numbers:

\begin{itemize}
  \item \textbf{ACTION Dnnn} Displays a NetView-provided, detailed description of a recommended action.
  \item \textbf{ACTION Ennn} Displays a description of a recommended action, created by your system programmer, for a user-defined generic alert action.
  \item \textbf{ACTION Innn} Displays a description of a recommended action created for a NetView-provided generic alert action.
  \item \textbf{ACTION Rnnn} Displays a description of an actual action created for a NetView-provided resolution action.
\end{itemize}

\section*{Overlaying Recommended Action Numbers}

Because details of a particular generic alert Recommended Action can vary depending on the sending product, Action Help panels cannot be provided for all possible generic actions. Therefore, on NetView Action Help panels built for generic alerts, each recommended action is preceded by an I-number (Tivoli-supplied action) or an E-number (user-supplied action).

On Recommended Action panels of the hardware monitor, each recommended action is identified with a special action number. Figure 30 shows a sample Recommended Action panel with three recommended actions (D225, D001, and D238).
Customizing Hardware Monitor Displayed Data

I-number and E-number actions do not have associated NetView-supplied panels. However, the NetView program allows users to overlay I-numbers and E-numbers with action numbers, to create panels that are specific to the sending product.

You can do this by modifying either table BNJDNUMB, which correlates a Product Set ID with action numbers, or table BNJDNAME, which correlates a Product Common Name with action numbers. BNJDNUMB is searched before BNJDNAME.

Modify table BNJDNUMB or BNJDNAME in NETVIEW.V5R1M0.BNJPNL2 and create BNJwwwww PDS members.

Modifying BNJDNUMB, BNJDNAME, and BNJwwwww

This section uses the names BNJDNUMB and BNJwwwww to indicate a PDS member.

BNJDNUMB

BNJDNUMB correlates a product-set identification (PSID) with a unique file or PDS member (BNJwwwww) that contains the action numbers to use for this product. To modify BNJDNUMB, use an editor such as ISPF/PDF.

Note: If the NetView program receives a generic alert whose PSID does not exist in BNJDNUMB and whose product common name does not exist in BNJDNAME, the default I-number or E-number is not modified.

The format for BNJDNUMB follows:

```
xxx
yyyyyyyyy BNJwwwww comment
  .  .  .
  .  .  .
```

Where:

Figure 30. Recommended Action Panel for Selected Event

I-number and E-number actions do not have associated NetView-supplied panels. However, the NetView program allows users to overlay I-numbers and E-numbers with action numbers, to create panels that are specific to the sending product.

You can do this by modifying either table BNJDNUMB, which correlates a Product Set ID with action numbers, or table BNJDNAME, which correlates a Product Common Name with action numbers. BNJDNUMB is searched before BNJDNAME.

Modify table BNJDNUMB or BNJDNAME in NETVIEW.V5R1M0.BNJPNL2 and create BNJwwwww PDS members.

Modifying BNJDNUMB, BNJDNAME, and BNJwwwww

This section uses the names BNJDNUMB and BNJwwwww to indicate a PDS member.

BNJDNUMB

BNJDNUMB correlates a product-set identification (PSID) with a unique file or PDS member (BNJwwwww) that contains the action numbers to use for this product. To modify BNJDNUMB, use an editor such as ISPF/PDF.

Note: If the NetView program receives a generic alert whose PSID does not exist in BNJDNUMB and whose product common name does not exist in BNJDNAME, the default I-number or E-number is not modified.

The format for BNJDNUMB follows:

```
xxx
yyyyyyyyy BNJwwwww comment
  .  .  .
  .  .  .
```

Where:
Customizing Hardware Monitor Displayed Data

**xxx** Specifies the number of entries in BNJDNUMB. This number must begin in column 1 and should be three characters long with leading zeros, if necessary.

**yyyyyyyy** Specifies up to nine characters representing the PSID. This entry must begin in column 1.

**BNJwwwww**
Is the name of the PDS member beginning in column 11, that contains generic alert recommended action code points and associated action numbers. Names such as BNJDNUM2, BNJDNUM3, and so forth, are recommended. However, you can use any unique name. The name BNJDNUM1 is already used for generic alerts produced by the hardware monitor.

Entries in BNJDNUMB must be in ascending order. Comment lines contain an asterisk (*) in column 1.

The first line cannot be a comment line. Figure 31 shows an example of BNJDNUMB. In this example, 7-character NetView PSIDs for the various operating systems map the action number file or PDS member used by the NetView program to BNJDNUM1.

<table>
<thead>
<tr>
<th>002</th>
<th>5665361 BNJDNUM1 NETVIEW R2 MVS/370</th>
</tr>
</thead>
<tbody>
<tr>
<td>5665362 BNJDNUM1 NETVIEW R3 MVS/XA</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 31. Example of BNJDNUMB Table*

**Determining the PSID:** Because the sending product can be either a hardware product or a software product, the PSID is defined as follows:

- For hardware products, the PSID is defined with the four numeric characters identifying the machine type found in the X'00' subfield, Hardware Product Identifier (located in the first X'11' subvector of the first X'10' subvector in the generic alert).
- For software products, the PSID is defined with the nine uppercase alphanumeric characters of the serviceable component identifier in the X'02' subfield, software product serviceable component identifier (located in the first X'11' subvector of the first X'10' subvector in the generic alert).

**Note:** If the X'02' subvector does not exist, use the seven uppercase alphanumeric characters of the program product number in the X'08' subvector, software product program number (located in the first X'11' subvector of the first X'10' subvector in the generic alert).

Two methods are available to determine the PSID of a generic alert that is logged to the hardware monitor database:

- Select sel# C from Alerts-Static, Alerts-History, or Most Recent Events panels to display a message containing the PSID.
- Make a selection from the Event Detail menu to display page 1 of the PSID panel. This panel displays the sending PSID.
BNJDNAME

BNJDNAME correlates a product common name with a unique file or PDS (BNJwwwww) that contains the action numbers to use for this product. To modify BNJDNAME, use an editor such as ISPF/PDF.

The format for BNJDNAME follows:

```
xxx
yyyyyyyyyyyyyyyyyyyyyyyyyyyyy BNJwwwww comment
```

Where:

- `xxx` Specifies the number of entries in BNJDNAME. This number must begin in column 1 and must be three characters long with leading zeros, if necessary.

- `yyyy...y` Specifies up to 30 characters representing the software product common name or up to 15 characters specifying the hardware common name.

- `BNJwwwww` is the name of the PDS member beginning in column 32, that contains generic alert recommended action code points and associated action numbers. Names such as BNJDNUM2, BNJDNUM3, and so forth, are recommended. However, you can use any unique name. The name BNJDNUM1 is already used for generic alerts produced by the hardware monitor.

- `comment` Comments must start in column 45.

NetView provides the following data in this PDS member:

```
001
NETVIEW BNJDNUM1 NETVIEW PRODUCT
```

Figure 32. Sample BNJDNAME Table

**Determining the Product Common Name:** Because the sending product can be either hardware or software, the product common name is defined as follows:

- For hardware products, the hardware common name is defined by the EBCDIC characters found in the X'0E' subfield, Hardware Product Common Name (located in the first X'11' subvector of the first X'10' subvector in the generic alert).

- For software products, the software common name is defined by the EBCDIC characters found in the X'06' subfield, Software Product Common Name (located in the first X'11' subvector of the first X'10' subvector in the generic alert).

To determine the product common name of a generic alert that is logged to the hardware monitor database, make selection 2 from the Event Detail menu. This selection will display the common name (hardware or software) of the sending product.

**BNJwwwww**

Each BNJwwwww member contains generic alert recommended action code points and associated action numbers. To create the BNJwwwww files or members specified in table BNJDNUMB, use an editor such as ISPF/PDF. Each BNJwwwww PDS member should be stored in the first data set in the concatenation string for the DD statement BNJPNL2. This DD statement is in the NetView startup procedure.
Avoid defining your panel data set with secondary extents when modifying or creating a panel while the NetView program is running. If a secondary extent is defined while NetView is running, a secondary extent failure can occur causing error recovery and loss of a single instance of a request. If a second attempt is made to execute the request, error recovery might succeed in the execution of the request. However, recycling NetView would be required for a full data set.

The format for BNJwwwwww follows:

```
xxxx yyyyyyyyy dnum
.
.
.
```

Where:

- **xxxx**: Is the 4-character generic alert recommended action code point (EBCDIC version of the recommended action code point as defined by the generic alert architecture). This field must begin in column 1.

- **yyyyyyyy**: Is the 8-character alert ID number (EBCDIC version of the alert ID number as defined in the X'92' subvector architecture). This field is optional. If present, it must begin in column 11.

- **dnum**: Is the 4-character unique action number. This field begins in column 21. Action numbers can be any combination of four EBCDIC characters. The limiting factor of the action number is the ability of the ACTION command list to use these four characters and display the associated panel.

Entries in each BNJwwwwww file or member must be in ascending hexadecimal order. If a nonhexadecimal number is used, it is skipped.

The BNJwwwwww file or member specified in BNJDNUMB or BNJDNAME is searched serially until a match is found or the end of the file is reached. After the first * is found in column 1, the serial searching stops.

You can place blanks in the alert ID field, along with specific alert IDs, for a particular action code point.

**Figure 33** shows a sample BNJwwwwww user-defined table.

```
1002 0562
1002 93987791 0890
1002 02556B79 0777
```

**Figure 33. Sample BNJwwwwww User-Defined Table**

For alert D2556B79, the code point 1002 uses D777 as its action number. For alert 93987791, code point 1002 uses D890 as its action number. For all other alerts from this sending product, code point 1002 uses D562 as its action number.

---

**Changing Color and Highlighting for Hardware Monitor Panels**

For the hardware monitor displays, you can alter the color, highlighting, and intensity of the display’s text. You can also enable the display to produce an audible alarm. Consider the needs of the display users before you modify these four attributes as assigned by the NetView program.
Customizing Hardware Monitor Displayed Data

**Note:** Changing the length of any attribute, row placement, or column placement will yield unpredictable results.

For any string of display text that is preceded by a blank, you can modify up to four attributes as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>Text is red, yellow, blue, white, green, turquoise, or pink.</td>
</tr>
<tr>
<td>Highlighting</td>
<td>Text is underscored, blinking, or in reverse video.</td>
</tr>
<tr>
<td>Intensity</td>
<td>Text is more intense (monochrome terminals only).</td>
</tr>
<tr>
<td>Alarm</td>
<td>Text causes an audible alarm at the user’s terminal.</td>
</tr>
</tbody>
</table>

You can change these attributes for specific displays or for all displays. For example, you can select a single color for prompt lines on all displays.

The procedure for modifying these attributes begins with a color map. A color map is a table that embeds characters, representing the various attributes, in a color buffer. These characters in the color buffer control the appearance of the text.

The automation table can also be used to set or change the color and highlighting of specific alerts for hardware monitor display.

**Reference:** For more information, refer to the Tivoli NetView for z/OS Automation Guide.

### Selecting the Color Map

The first step in modifying a hardware monitor display is to determine which color map controls the display you want to change. Appendix A, Color Maps for Hardware Monitor Panels contains a matrix of the panel name, panel number, and color map for hardware monitor panels.

After you identify the color map you need, edit the map using an editor such as ISPF/PDF. The color maps are contained in the PDS named NETVIEW.V5R1M0.BNJPNL2 (unless the name is changed during installation). The member name is the color map name.

**Note:** If you want a particular attribute to apply to the same portion of each panel, modify the color map BNJOVERW, which overwrites all other panel-specific color maps. Be sure to test the results of BNJOVERW on each panel before putting it into your production system. This map can produce unexpected results.

### Modifying the Color Map

After you select the color map, you can modify it. A color map consists of a series of lines of data, called map elements. The top line of a color map is always the number of subsequent map elements. Map elements begin in column 1, and are paired with comments that begin in column 41.

Each map element specifies, for a particular display row, the attribute, the attribute’s placement in the row, and the length in characters. Each item in the map is followed by a comma, except for the last one, which is followed by a period.

**Note:** Changing any attribute’s length, row placement, or column placement can yield unpredictable results.
Figure 34 shows a sample color map. Explanations of the numerical references follow on pages 87–88.

The first item in the color map represents the number of subsequent lines of data, or map elements. A map can have any number of map elements. The sample map has 13 map elements.

2, 3, and 4 describe the three types of map elements as follows:

2. This type of map element contains attribute information in the following format:
   - The first item is the number of attributes in the map element. This number can be 1–4. A map element might have only one set of attributes, for example, pink color, or any combination of attributes, such as pink color and underscoring. The sample map element has one attribute, the color blue (BLU).
   - The second item is the number of the display row that reflects the attribute. In the sample, the attribute is to appear in row 1.
   - The third item is the number of the display column that contains the attribute character. In the sample, the attribute character is to be placed in column 1. Consequently, the displayed text will begin in column 2.

   **Note:** Be sure that the display text you want to modify is preceded by a blank space. Otherwise, the character representing the attribute in the color buffer overwrites some of the display text, and some characters are replaced with blanks. For example, in the following string you cannot make the colon a different color from the text:
   
   **EVENT DESCRIPTION:** PROBABLE CAUSE

   - The fourth item is the maximum character length of the attribute. In the sample, the specified attribute covers 79 characters on the display, or columns 2–80.
   - The last item is the attribute or sequence of several attributes. In the sample, the color blue is the specified attribute. You can specify up to four attributes, but only one from each category. If you want multiple attributes to apply to the same character or string, you must specify the attributes for each category in this order:

     1. Alarm: ALM produces an audible alarm.
     2. Intensity:
        - HIG intensifies the color.
        - NOH returns the color to normal intensity.
     3. Highlighting:
Customizing Hardware Monitor Displayed Data

- UND underlines the character or string.
- BLI causes the character or string to blink.

4. Color:
- RED produces red.
- YEL produces yellow.
- BLU produces blue.
- WHI produces white.
- GRE produces green.
- TUR produces turquoise.
- PIN produces pink.

This map element makes the text in row 1, columns 2–80, blue. As the map element’s corresponding comment confirms, this blue string of text is the display header.

3 This type of map element uses the repetition factor option to copy the attribute or attributes specified for a particular row onto subsequent rows. A repetition map element uses the following format:

• The number 99 signals the repetition of an element.
• In SIZE-x-y:
  - SIZE represents the total number of rows in the panel. Use the word SIZE as shown; do not replace it with a number.
  - x is the number of unused or blank lines between the end of the panel data and the prompt line. In the sample, no blank or unused lines occur between the end of the panel data and the prompt line.
  - y is the number of the starting row that is to copy, or repeat, the attribute or attributes from the preceding row. In the sample, attributes from row 6 are to be repeated on the subsequent rows, starting with row 7.
• The last item (2) is the number of attributes on row 6 that are repeated. In the sample, the two attributes specified in the map for row 6 are to be repeated.

This map element copies the two attributes specified for row 6 onto subsequent rows starting at row 7, and continues to the prompt line.

4 This type of map element uses the variable row placement option to specify the row that contains the attribute. This option uses the following format:

• The first item (1) is the number of attributes in the map element. This number can be 1–4. In the sample, the map element has one attribute, the color blue (BLU).
• The second item (SIZE-x) indicates the display row that reflects the attribute, where:
  - SIZE represents the total number of rows in the display. Use the word SIZE as shown; do not replace it with a number.
  - x is the number of lines above the command line. For example, for the Alerts-Static display:
    - SIZE-4 is the first prompt line.
    - SIZE-3 is the second prompt line.
    - SIZE-2 is the message line.
    - SIZE-1 is the NetView status line.
    - SIZE-0 is the command line.

In the sample, the attribute is to appear on the first prompt line.
Customizing Hardware Monitor Displayed Data

Note: Be sure that the command line is defined on byte 80 of the NetView status line. Otherwise, some bytes can be overwritten.

- The third item (1) is the number of the display column that contains the attribute character. In the sample, the attribute character is placed in column 1. Consequently, the displayed text begins in column 2.

Note: Be sure that the display text you want to modify is preceded by a blank space. Otherwise, the character representing the attribute in the color buffer overwrites some of the display text, and some characters are replaced with blanks.

- The fourth item (50) is the maximum character length of the attribute. In the sample, the specified attribute covers 50 characters on the display.

- The last item (BLU) is the attribute or sequence of several attributes. You can specify up to four attributes, but only one from each category. If you want multiple attributes to apply to the same character or string, you must specify the attributes in the order shown on page 57. In the sample, the color blue is the specified attribute.

This sample map element makes the text in the first prompt line, columns 2–51, blue.

Prompt Highlight Tokens

The prompt highlight token table BNJPROMP is located in the PDS named NETVIEW.V5R1M0.BNJPNL2. You can modify this table. The maximum size of the table is 25 prompts, with the prompt being a 15-byte character field. If you decide to modify the table, use the Comment column for notes about the table. For performance reasons, this table is not processed when building the Alert Dynamic panel. Color is a 3-byte character field beginning at column 20. You can select only those colors that are valid in the color maps. Table 17 is a sample of the format for the prompt highlight token table.

Table 17. Prompt Highlight Tokens

<table>
<thead>
<tr>
<th>Prompt Token</th>
<th>Color</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL#</td>
<td>WHI</td>
<td>PROMPT SEL#</td>
</tr>
<tr>
<td>LDM</td>
<td>WHI</td>
<td>PROMPT LDM</td>
</tr>
<tr>
<td>LSL1</td>
<td>WHI</td>
<td>PROMPT LSL1</td>
</tr>
<tr>
<td>LSL2</td>
<td>WHI</td>
<td>PROMPT LSL2</td>
</tr>
<tr>
<td>RESNAME</td>
<td>WHI</td>
<td>PROMPT RESNAME</td>
</tr>
<tr>
<td>RESNAME1</td>
<td>WHI</td>
<td>PROMPT RESNAME1</td>
</tr>
<tr>
<td>RESNAME2</td>
<td>WHI</td>
<td>PROMPT RESNAME2</td>
</tr>
<tr>
<td>'A'</td>
<td>WHI</td>
<td>PROMPT A</td>
</tr>
<tr>
<td>'B'</td>
<td>WHI</td>
<td>PROMPT B</td>
</tr>
<tr>
<td>'P'</td>
<td>WHI</td>
<td>PROMPT P</td>
</tr>
<tr>
<td>'EV'</td>
<td>WHI</td>
<td>PROMPT EV</td>
</tr>
<tr>
<td>'ST'</td>
<td>WHI</td>
<td>PROMPT ST</td>
</tr>
<tr>
<td>'DM'</td>
<td>WHI</td>
<td>PROMPT DM</td>
</tr>
<tr>
<td>'M'</td>
<td>WHI</td>
<td>PROMPT M</td>
</tr>
<tr>
<td>'DEL'</td>
<td>WHI</td>
<td>PROMPT DEL</td>
</tr>
<tr>
<td>'S'</td>
<td>WHI</td>
<td>PROMPT S</td>
</tr>
</tbody>
</table>
### Using NMVT Support for User-Written Programming

Network management vector transport (NMVT) support enables user-written programs to report errors to the hardware monitor through generic alerts. Prior to generic alerts, Recommended Action panels, Event Detail panels, and alert messages were stored at the host in the NetView program. Each nongeneric alert had a unique set of panels and messages.

**Note:** The original NMVT encoding contains many SNA major vectors including Alerts. Subsequent encoding such as MDS_MU and CP_MSU contains many of the same major vectors and are covered under the term NMVT in this section.

Coded generic alerts are contained in the NMVT. Generic alert code points are used to dynamically build the hardware monitor panels. Nongeneric alerts are used mainly for migration purposes. You should create new user-defined alerts using generic alerts.

**Reference:** For more information on major vectors and subvectors of an NMVT, refer to the SNA library.

This section contains a sample generic alert and the associated panels that are built by the hardware monitor. (See [Figure 35 on page 93](#) through [Figure 39 on page 97](#).) This section also describes how each panel is built.

### User-Defined Alerts (Nongeneric)

Sixteen block IDs (X'F000'–X'F0F0'), which are part of NMVT major vector X'0000', are reserved for generating user-defined alerts.

The hardware monitor reserves USER0bb–USERFbb (where bb are required blank space X’40’ characters to pad the name to 7 characters) for use as the corresponding 7-character software identifier in the software product program number (X’08’) subfield of the first product identifier (X’11’) subvector of the NMVT. These are mapped to the block IDs from X’F000’ to X’F0F0’.

The hardware monitor allows a 1-byte alert description code within the basic alert (X’91’) subvector of the NMVT. This code lets you further qualify the alert. Put your alert description code in the second byte of the 2-byte Alert Description Code field. The hardware monitor ignores the first byte of that field.

### NMVT-to-Panel ID Mapping

Using the block ID derived from the software product program number and the alert description code, the hardware monitor maps the NMVT to the following:

- 14-line panel
Customizing Hardware Monitor Displayed Data

A 14-line panel appears on the Recommended Action panel of the hardware monitor for the NMVT. The PDS member name for this 14-line panel is in the range between BNIF00xx and BNIF0FxX, where the range of block IDs is from X'F00' to X'F0F', and xx is the hexadecimal value of the alert description code. The lines can be up to 80 characters long.

- 7-line panel
  A 7-line panel appears on the hardware monitor’s event detail panel for the NMVT. The 7-line panel’s PDS member name is in the range between BNKF00xx and BNKF0FxX, where the range of block IDs is from X'F00' to X'F0F', and xx is the hexadecimal value of the alert description code.
  The first eight translated characters of each of the first three X'A0' or X'A1' qualifier subvectors are displayed on an eighth line, immediately following the Event Detail panel. Write the Event Detail messages, with titles on the seventh line, to describe the qualifiers.

- 48-byte alert description
  A 48-byte alert description appears on the Alerts-Dynamic, Alerts-Static, Alerts-History, Event Detail, and Most Recent Events panels. The 48-byte text descriptions for a block ID are in a NetView message CSECT whose link-edit load module name is in the range between BNJVMF00 and BNJVMF0F.

Panel Formats
For each new Recommended Action panel or Event Detail panel, use the same format as in the existing panels to add a panel to the NetView panel library or a concatenated user library.

For each new 48-byte alert description CSECT, use the same format as an existing BNJVMxxx CSECT. BNJVMxxx CSECTs are coded using the macro DSIMDS. No variable substitution is permitted for 48-byte alert descriptions.

User-Defined Alerts (Generic)
Generic alerts allow coded alert data to be transported within the alert, eliminating the need for stored panels. The coded data can be one of the following:

- An index into predefined tables, containing short units of text that are used to build a panel
- Textual data that appears directly on the panel

Coded data is maintained in code point tables which can be customized (For more information on customizing code point tables, see “Modifying Generic Code Point Tables” on page 99). The text strings indexed by the code points, and the display of textual data that was sent in the alert, are in the same format no matter which product sent the alert. Also, the same terminology is used to define similar problems within different products because each product uses terminology defined by Tivoli.

Generic alerts produce the same Alerts, Recommended Action, and Detail panels as the hardware monitor’s nongeneric alert support, but the panels are built dynamically rather than using stored panels. Code points index into the tables defined by Tivoli and the user.

The alert description and probable cause code points are used to build the hardware monitor Alerts-Dynamic, Alerts-Static, Alerts-History, Event Detail, and Most Recent Events panels. The user cause, install cause, failure cause, and recommended action code points are used to build the hardware monitor Recommended Action panel. The detail data code points are used to identify the
Customizing Hardware Monitor Displayed Data

Qualifiers that can appear on the hardware monitor Recommended Action or Event Detail panel. Products use the same set of architected product-independent terminology to define their Alert, Recommended Action, and Detail panels. Text data transported in the NMVT is displayed on the Event Detail panel.

The NetView program ships generic code point tables that can be customized (for more information on customizing code point tables, see “Modifying Generic Code Point Tables” on page 99). The generic code point tables shipped by NetView are:

- BNJ92TBL—Alert description code points
- BNJ93TBL—Probable cause code points
- BNJ94TBL—User cause code points
- BNJ95TBL—Install cause code points
- BNJ96TBL—Failure cause code points
- BNJ81TBL—Recommended action code points
- BNJ82TBL—Detail data code points
- BNJ85TBL—Detailed data code points, subfield X’85’
- BNJ86TBL—Actual action code points.

Using the GENALERT Command

You can use the GENALERT command to create your own alerts. The GENALERT command is described in the NetView online help.

Building Generic Alert Panels

Figure 35 on page 93 is an example of a generic alert NMVT. Unique panels are built using the information contained in a generic alert record.

Reference: For more information on NMVTs, refer to the SNA library.
Customizing Hardware Monitor Displayed Data

Figure 35. Sample Generic Alert Record

X'41038D5002000000'  Response Header
X'01230000'  Major Vector Length and Key
X'0A011005901020A2827'  01 SV - Date/Time
X'0B92000000'  92 SV - Alert Description
X'1603'  code point
X'1A2B3C4D'  93 SV - Probable Cause(s)
X'693'  code point
X'0403'  code point
X'2012'  code point
X'195'  code point
X'0601'  95 SV - Install Cause(s) and Action(s)
X'1502'  01 SF - install cause(s)
X'13E1'  code point
X'38391'  code point
X'0681'  code point
X'0101'  code point
X'1504'  code point
X'2796'  96 SV - Failure Cause(s) and Action(s)
X'0601'  01 SF - failure cause(s)
X'0503'  code point
X'3302'  code point
X'068200'  code point
X'1E1'  code point
X'0604'  code point
X'0C8200'  82 SF - qualifier(s)
X'53'  code point
X'11F0F0406040F1C6'  81 SF - recommended action(s)
X'0611'  code point
X'0500'  code point
X'3110'  code point
X'0081'  code point
X'238321'  code point
X'1705'  05 SV - Resource Hierarchy
X'151000'  10 SF
X'0D7B4499F9F900F1'  name/type pair
X'0D3C95D5C9F440F9'  name/type pair
X'4D6100'  10 SV - PSID
X'341104'  11 SV - Product Identifier
X'0B2C3C61C9C2D44040F0F3'  02 SF - software product serviceable component ID
X'0804F1P9F2F0F0F3'  04 SF - software product common level
X'0A07C6C60C6C9E3D9F3'  06 SF - software product common name
X'07096E0351225'  07 SF - software product customization ID
X'161101'  09 SF - software product customization date and time
X'130012'  11 SV - Product Identifier
X'F9C89F89F1C1F0F5'  00 SF - hardware product identifier
X'9C1F0C1F0C1F0F0'  83 SF
X'1738'  98 SV - Detailed Data
X'0782113400'  82 SF - qualifier
X'0004'  82 SF - qualifier
X'0782009111'  82 SF - qualifier
X'F2F2'  82 SF - qualifier
X'0782000E00'  82 SF - qualifier
X'0D0C'  48 SV - Correlation
X'2548'  60 SF - correlation for supporting data
X'07C3C4D34F04F4'  82 SF - qualifier
X'5C3D5D40F0F1'  82 SF - qualifier
X'0D82'  82 SF - qualifier
X'0D0A1C3D6D460C5D9D9'  82 SF - qualifier
X'682000D1010F'  82 SF - qualifier
X'3E61'  31 SV - Self Defining Text Message
X'0E02113400500'  02 SF - Coded Character Set ID
X'051C0584'  12 SF - National Language ID
X'139112'  21 SF - Sender ID
X'2620'  10 SF - Text Message
X'03C9C2408E4C26C9C5D3C440C9C45D5E3C9C6C9C5E240E3C8C540E3C5B7E340D482'  83 SF

Figure 36 on page 94 through Figure 38 on page 97 describe how each unique panel is built using the information contained in a generic alert NMVT. Figure 36 on page 94 shows a sample Alerts-Dynamic panel. Explanations of the numerical references follow the panel.
An entry on the Alerts-Dynamic panel is built from a number of subvectors (X’92’, X’93’, and X’05’). Figure 35 on page 93 creates the results for Figure 36.

1 The RESNAME and TYPE come from the last name and type pair in the X’05’ subvector. The sample display shows a RESNAME of PU9999 and a TYPE of LINE.

2 The * indicates that the RESNAME preceding the TYPE does not belong to the TYPE. The TYPE is always associated with the last name in the hierarchy, but the name depends on how the X’05’ is coded. The Do Not Display Resource Name Indicator bit is set to 1 for the last name and type pair (subvector X’05’, subfield X’10’, second name and type pair, eighth byte, second bit).

3 The ALERT DESCRIPTION is derived from code point X’1603’ in the X’92’ subvector. The code point provides an index into a table containing the alert description text messages. The sample shows an ALERT DESCRIPTION of COMM SUBSYSTEM FAILURE.

4 The PROBABLE CAUSE is derived from code point X’0403’ in the X’93’ subvector. The code point provides an index into a table containing the probable cause text messages. The sample shows a PROBABLE CAUSE of COMM SUBSYSTEM CTRL.

5 The + is included because the X’93’ subvector in Figure 36 contains more than one probable cause code point. The + indicates that more probable causes can be seen on the Event Detail panel.

Figure 37 on page 95 shows a sample Recommended Action panel. Explanations of the numerical references follow the panel.
The Recommended Action panel is built from a number of subvectors (X'94', X'95', and X'96') and subfields (X'01', X'81', X'82', and X'83').

1. The resource names (PU9999 and LINE04) are taken from the X'05' hierarchy names list subvector. In Figure 35 on page 93, only names from the X'05' subvector are used because the Hierarchy Complete Indicator bit (byte 2 bit 0) in the indicator bit X'05' subvector is set to X'0'. If this bit was set to 1, the NetView program would concatenate the names in the X'05' subvector to the names supplied by VTAM.

2. The resource types (PU and LINE) are derived by converting the type codes in the X'10' subfield of the X'05' subvector (X'F1' and X'F9') into displayable resource types. For more information on changing resource types, see “Adding or Modifying Resource Types” on page 103.

3. The X'94' subvector (NONE) carries user-caused information. Because the X'94' subvector is not included in Figure 35 on page 93, user-caused information is not displayed.

4. The two install-caused probable causes:
   - INCORRECT MICROCODE FIX
   - INCORRECT SOFTWARE GENERATION:

are built from code points (X'1502' and X'13E1') in the X'01' subfield within the X'95' subvector. The E in the X'13E1' code point indicates an X'83' subfield is needed to complete the install cause.
Customizing Hardware Monitor Displayed Data

5 The qualifier on the install cause (ACF/IBM) is displayed because of the X'83' subfield of the X'95' subvector. The X'83' subfield contains the value X'91' indicating that the qualifier is taken from the product ID subfield (X'06' Software Product Common Name) of the first product identifier subvector (X'11').

6 The two install-caused actions:
   I013 - VERIFY X.25 SUBSCRIPTION NUMBER
   I085 - APPLY CORRECT SOFTWARE LEVEL

are taken from code points (X'0101' and X'1504') in the X'81' subfield of the X'95' subvector.

7 The two failure-caused probable causes:
   COMMUNICATIONS SUBSYSTEM
   LINE ADAPTER MICROCODE

are taken from code points (X'0503' and X'33C2') in the X'01' subfield of the X'96' subvector. The C in the X'33C2' code point indicates that two detail data subfields, either X'82' or X'85' subfields, are needed to complete the failure cause. This example uses X'82' subfields. While either X'82' or X'85' subfields can be used here, a combination of the two would not be valid. Within a subvector, all of the detail qualifiers must be X'82' subfields or X'85' subfields.

8 Indicates the ADAPTER NUMBER 04 is broken down from the first X'82' subfield in the X'96' subvector. The number can be:
   00   No information is taken from the PSID subvector
   61   A code point for adapter number
   00   Hexadecimal data follows
   04   Hexadecimal data to be displayed

   LINE ADDRESS RANGE 00 - 1F is broken down from the second X'82' subfield in the X'96' subvector. The range can be:
   00   No information is taken from the PSID subvector
   53   A code point for line address range
   11   EBCDIC data follows
   F0F046D40F1C6
       EBCDIC data to be displayed

10 The failure-caused actions:
   I032 - DUMP CHANNEL ADAPTER MICROCODE
   I026 - RUN APPROPRIATE TRACE
   I136 - CONTACT COMMUNICATIONS SYSTEMS PROGRAMMER
   I010 - PERFORM 9999 PROBLEM DETERMINATION PROCEDURES

are taken from the code points (X'0611', X'0500', X'3110', and X'00E1') in the X'81' subfield of the X'96' subvector. The E in the X'00E1' code point indicates that an X'83' subfield is needed to complete the failure cause.

11 The qualifier on the failure cause (9999) is displayed because of the X'83' subfield of the X'96' subvector. The X'83' subfield contains the value X'21', indicating that the qualifier is taken from the first hardware PSID subfield (X'00') of the PSID subvector (X'11').

Figure 38 on page 97 and Figure 39 on page 97 show sample Event Detail panels. Explanations of the numerical references follow the figures.
Event Detail Panel

The Event Detail panel is built from subvectors X'92', X'93', X'98', X'01', X'31', and X'48', and subfield X'82'.

The resource names (PU9999 and LINE04) are taken from the X'05' hierarchy names list subvector. In Figure 35 on page 93, only names from the X'05' subvector...
Customizing Hardware Monitor Displayed Data

are used because the Hierarchy Complete Indicator bit (byte 2, bit 0) in the X'05' subvector is set to X'0'. If this bit was set to 1, the NetView program would concatenate the names in the X'05' subvector to the names supplied by VTAM.

2 The resource types (PU and LINE) are derived by converting the type codes in the X'10' subfield of the X'05' subvector (X'F1' and X'F9'), into displayable resource types. For more information on changing resource types, see "Adding or Modifying Resource Types" on page 102.

3 The DATE/TIME RECORDED is the time the record is logged to the hardware monitor database. The created field shows the time the record was created by the sending product. It is taken from the X'10' subfield of the X'01' subvector.

4 EVENT TYPE is derived from byte 4 (Alert Type) the X'92' subvector.

5 DESCRIPTION is derived from the code point (X'1603') in the X'92' subvector, as is the description on the Alerts panel. However, a longer version of the text is displayed on this panel.

6 PROBABLE CAUSES are taken from the code points (X'0403' and X'2012') in the X'93' subvector. A longer version of the text is displayed on this panel than was displayed on the Alerts panel. Also, all of the probable causes are displayed.

7 QUALIFIERS are derived from either X'82' or X'85' subfields. The NetView program ignores X'01' subfields and associated sub-subfields (including X'82' and X'85') in a X'98' subvector.

While either X'82' or X'85' subfields can be used here, a combination of the two would not be valid. Within a subvector, all of the detail qualifiers must be X'82' subfields or X'85' subfields.

This example uses X'82' subfields, and the qualifiers are decoded as follows:

First in the X'98' subvector:

21 Data should be taken from the first hardware PSID subfield (X'00') of the PSID subvector (X'11').
34 Code point indicating communication control unit.
00 Hexadecimal data follows.
0004 Hexadecimal data to be displayed.

Second in the X'98' subvector:

00 No data is taken from the PSID subvector.
09 Code point indicating event code.
11 EBCDIC data follows.
F2F2 EBCDIC data to be displayed.

Third in the X'98' subvector:

00 No data is taken from the PSID subvector.
0E Code point indicating reason code.
00 Hexadecimal data follows.
00DC Hexadecimal data to be displayed.

Page 2 of the Event Detail panel (see Figure 38 on page 97) contains the following information:
Customizing Hardware Monitor Displayed Data

1. CONTROL PROGRAM TEXT is the text title displayed because of the subfield X'21' of subvector X'31'. The text itself is taken directly from subfield X'30' of the X'31' subvector and displayed on the screen.

2. The CORRELATION FOR SUPPORTING DATA is displayed from the X'48' subvector. Subfield X'60' specifies that the network-qualified procedure correlation identifier be used to uniquely identify a session.

Either X'82' or X'85' subfields are used for supporting data. This example uses two X'82' subfields to identify the supporting data.

While either X'82' or X'85' subfields can be used here, a combination of the two is not valid. Within a subvector, all of the detail qualifiers must be X'82' subfields or X'85' subfields.

3. The product ID (ACF/IBM) is taken directly from the first product identifier (X'11') subvector in the first PSID (X'10') subvector. Figure 35 on page 93 uses the Software Product Serviceable Component Identifier (X'02') subfield.

4. The alert ID number (1A2B3C4D) is taken from subvector X'92' bytes 7–10.

Modifying Generic Code Point Tables

This section explains how to modify the generic alert code point tables that are shipped with the NetView program. You can modify the tables before or after NetView initialization. If after, use the CPTBL command to dynamically activate the changes. The CPTBL command is described in NetView online help.

Table Formats

Each table contains a different type of code point. The tables are:

- BNJ92TBL: Alert description code points
- BNJ93TBL: Probable cause code points
- BNJ94TBL: User cause code points
- BNJ95TBL: Install cause code points
- BNJ96TBL: Failure cause code points
- BNJ81TBL: Recommended action code points
- BNJ82TBL: Detail data code points
- BNJ85TBL: Detail data code points, X'85' subfield
- BNJ86TBL: Actual action code points.

The fourth and fifth characters of the table name identify the subvector or subfield that contains the code points.

The first entry in the code point table is the control entry. Columns 1 and 2 represent the subvector number which specifies which of the code point tables is being created or updated. Acceptable values are 92, 93, 94, 95, 96, 81, 82, 85, or 86. During initialization, this number must match the table name. Column 3 must be blank and all remaining columns are unused and are ignored. (You should not use this area for comments because it may be used for other purposes in the future.) When using the CPTBL command, the name of the file that contains the code point definitions does not have to be one of the predefined names. NetView uses this control entry to determine the table type.

The format of each subsequent entry in the code point table is:
Customizing Hardware Monitor Displayed Data

- Columns 1–4 contain the 4-character hexadecimal code point number. Valid characters are 0–9 and A–F. The code point range from X’E000’ to X’EFFF’ is reserved for your use. To use code points outside this range, contact the Tivoli Support Center.

If a code point is defined more than once in a given table, the first entry is used, subsequent entries are ignored, and an informational message is generated.

- Column 6 contains the embed flag (Y) indicating that qualifier data associated with the X’82’, X’83’, or X’85’ subfield is placed before the code point’s text, embedded within the code point’s text, or follows on the same line after the code point’s text. Any character other than Y indicates that the embed flag is off. If the embed flag is turned on, the embed information included in the generic alert is embedded at the point marked by a dollar sign ($). Embedded text is only supported for BNJ81TBL, BNJ86TBL, BNJ94TBL, BNJ95TBL, and BNJ96TBL. Because no variable substitution is allowed for probable cause and alert description, an embed flag is ignored in BNJ92TBL and BNJ93TBL.

- Columns 8–72 contain the text description for this code point. The maximum length of the text varies as follows:
  - Probable cause: 40 characters for the first entry of a given code point, 20 for the second. (See 4 in “Example of BNJ92TBL Code Points Table” on page 101 for an explanation of the second entry.)
  - Alert description: 40 characters for the first entry of a given code point, 25 for the second. (See 4 in “Example of BNJ92TBL Code Points Table” on page 101 for an explanation of the second entry.)
  - Detail data: 40 characters
  - Others: 108 characters.

Start in column 2 when continuing the text on the next line.

- Columns 73–80 are ignored and can be used for optional sequence numbers.

Notes:

1. Code points in table BNJ82TBL must be left-justified and padded with zeros. For example, you enter code point 12 as 1200.

2. The text for the code point entries added to the NetView BNJ81TBL code point table should begin with Ennn. The text for the code point entries added to the NetView BNJ86TBL code point table should begin with Rnnn. The use of Ennn and Rnnn allows the code points to be supported by the ACTION command list (for more information on the ACTION command list, refer to the NetView online help). The action text in BNJ81TBL and BNJ86TBL should begin this way. Otherwise, when BNJDNUMB is used to generate recommended action numbers, it overlays the first 4 bytes of the recommended action text.

3. The hardware monitor searches the tables for the specific code points. If a match is not found, the hardware monitor searches some tables for a general code point. A general code point is the code point with the last 2 bytes set to zero. For example, if the specific code point is 1620, the general code point is 1600. If a general code point is found, its text is returned as if it matched the original code point. A general code point contains text that is valid for all specific code points that it applies to. General code points are not available for BNJ82TBL and BNJ85TBL (for information on general code points, refer to the SNA library).

4. All code point tables are in uppercase. However, if you want to enter your own code in lowercase or mixed case, NetView does not convert the text to uppercase.
Use of %INCLUDE Statements
The use of %INCLUDE statements in the code points tables allows you to organize your code points information for easier maintainability.

You can choose to have one main table for each code point type. This table can contain the code points shipped with the NetView program and %INCLUDE statements for user-defined subtables and subtables defined by other products.

BNJxxTBL (where xx is the table number) are tables Tivoli does not recommend modifying. Use these tables as main tables for each code point. If customization of these tables is required, use the BNJxxUTB (where xx is the table number) file which is included by the main table (BNJxxTBL) for this purpose.

Example of BNJ92TBL Code Points Table
An example of a code points table is shown in Figure 40. Explanations of the numerical references follow the figure.

* An asterisk in column 1 indicates a comment line.
* The following line is the control entry indicating table type.
1 92
* Blank lines are allowed for readability.
2 %INCLUDE BNJ92UTB
3 4
0100 SIMPLE CODE POINT TEXT;
5 E123 THIS TEXT IS EXACTLY FORTY CHAR LS LONG XX;
6 E123 THIS IS THE SAME IN 25 XX;
6 FFFF

Figure 40. Sample of BNJ92TBL Code Points Table

1 The first non-comment line is the control entry.

2 Code point tables can use %INCLUDE statements to embed other files into the code point table.

3 The code point (0100) is a 4-character hexadecimal number, starting in column 1.

4 The text description in columns 8–72 appears on the hardware monitor displays.

5 The hardware monitor has different panel formats that allow different length text for alert descriptions (92) and probable causes (93). The maximum length of the text for either entry is 40 characters. Abbreviated text is required, if the text exceeds 25 characters for alert descriptions or 20 characters for probable causes. Errors occur for text entries greater than 40 characters.

6 Any entries in the table with code point FFFF and no text are ignored (to allow for migration). Entries with code point FFFF and text are treated as any other code point.

Example of BNJ94TBL Code Points Table
Another example of a code points table is shown in Figure 41 on page 102.
Customizing Hardware Monitor Displayed Data

* An asterisk in column 1 indicates a comment line.
* The following line is the control entry indicating table type.

```
%INCLUDE BNJ94UTB
0100 Y CODE POINTS TEXT WITH DETAIL INSERTS $ AND $
0200 CODE POINTS TEXT ILLUSTRATING CONTINUATION OF THE TEXT TO A SECOND LINE
0100 DUPLICATE TEXT
```

Figure 41. Sample of BNJ94TBL Code Points Table

1. Code point tables can use %INCLUDE statements to embed other files into the code point table.

2. The embed flag (Y in column 6) indicates that qualifier data is embedded at the point marked by a dollar sign ($).

3. Start in column 2 when continuing text on the next line. The text on the first line starts in column 8 and continues through column 72.

4. Because this code point has already been defined in the table, this entry is ignored and an informational message is generated.

Activating the Modified Code Point Tables
The CPTBL command is very similar to the AUTOTBL command and is used to dynamically activate changes made to code point tables after NetView is initialized (for a description of the CPTBL command, refer to NetView online help). Use the TEST option on the CPTBL command to verify the syntax of a code point table before activation.

Adding or Modifying Resource Types
You can add new resource types for hierarchical displays in the hardware monitor by modifying the member BNJRESTY.

BNJRESTY is a member of the data set NETVIEW.V5R1M0.BNJPNL2, defined by the definition statement BNJPNL2 in the NetView start procedure.

Figure 42 shows the format for BNJRESTY. Explanations of the numerical references follow the figure.

```
10 DISK your comments
```

Figure 42. Sample Contents of BNJRESTY

1. A 2-character hexadecimal number, starting in column 1, flows to the NetView program in the X’05’ subvector. Valid characters are 0–9 and A–F. If you include duplicate hexadecimal codes, the system uses the first entry of the duplicated code. Numbers from X’E0’ to X’EF’ are reserved for customer-defined resource types.

2. The four characters in columns 4–7 are taken as the resource type. Valid characters are 0–9, A–Z, and any printable special characters. A resource type of less than 4 characters must begin in column 4, and be padded on the right with blanks. Do not use delimiters, such as a comma (,), period (.), or equal sign (=), as characters in the resource type.
An optional comment can begin anywhere after the resource type.

If BNJRESTY is modified while the hardware monitor task BNJDSERV is active, the new resource types are not recognized. Use STOP TASK=BNJDSERV followed by STARTCNM NPDA so that the NetView program can recognize any new resource types or use the RTTBL command to activate a modified BNJRESTY member.

If the NetView program finds an entry that is not valid in BNJRESTY during activation of the NetView program or when the RTTBL command is invoked, an error message appears on the command facility console and the NetView program uses the Tivoli-supplied resource types.
Chapter 7. Modifying Network Asset Management Command Lists

Network asset management provides a way of collecting inventory data from a subset of hardware and software devices automatically. You can use network asset management to collect vital product data (VPD) such as serial numbers, machine types, and model numbers for hardware products and software information. This information includes version and release levels. However, the NetView program does not verify the returned data from devices supporting network asset management; it only provides a way to collect and log the data.

Reference: Refer to the Tivoli NetView for z/OS Administration Reference for information on the record formats. Refer to the NetView online help for information about NetView-provided command lists.

Any device that supports the REQUEST/REPLY PSID and LPDA-2 architecture can report VPD to the NetView program. An attempt to solicit VPD from a device that does not support the architecture can cause the keyboard to lock or extraneous data to appear on the screen. You may need to press the RESET key or clear the screen, but these actions do not affect the VPD collection in the NetView program.

Reference: Refer to the SNA library for information on the REQUEST/REPLY PSID and LPDA-2 architecture.

The following examples are some physical units (PUs) that support the REQUEST/REPLY PSID architecture:
• 3720/NCP
• 3725/NCP
• 3745/NCP
• 3174 that reports data for itself and many types of attached devices such as various models of 3191, 3192, and 3194 display stations.

Personal computers running OS/2® are required with these products.

Reference: Instructions for entering VPD for a device are located in the user’s guides for that device. Also, refer to the OS/2 library.

The following are examples of data circuit-terminating equipment (DCE) that supports the LPDA-2 modem and line status architecture:
• 586x modems
• 5822 DSU/CSU
• 7825 DSU/CSU
• 786x modems (7861, 7865, 7868).

Network asset management provides the VPDCMD command to solicit VPD from a given device and the VPDLOG command to build and log a record to an external logging facility (SMF for MVS). You can use Service LevelReporter (SLR) to view the data interactively or to generate reports, or the VPDALL command to generate VPDPU and VPDDCE command entries for all devices within a NetView domain. If you have any resources that require switched lines, be sure that the switched lines are active before collecting VPD.
Modifying NAM Command Lists

Network asset management provides the following command lists:

VPDPU
Collects and logs VPD from a single PU and its attached devices. You can enter this command list from an operator’s console or from another command list.

VPDDCE
Solicits and logs VPD from DCEs that are in a direct path between a specified NCP and a specified PU. You can issue this command list from an operator’s console or from another command list.

VPDACT
Is the default name of a command list that the VPDALL command generates when issued with the CREATE option. VPDALL reads a VTAM configuration member in VTAMLST as input and generates a command list called VPDACT (the default). VPDACT contains a list of VPDPU and VPDDCE entries for devices in your domain. You can later issue VPDACT to collect and log VPD from the supported devices in the NetView domain.

VPDLOGC
Is the command list that builds and logs START and END records. A START record is generated for a VPDACT command list at the beginning of a VPD solicitation. An END record is generated for a VPDACT command list at the end of a VPD solicitation. Do not issue this command list from an operator’s console or from a user-written command list.

VPDXDOM
Is a service command list used for VPD solicitation from cross-domain resources. This command list is driven through a NetView automation table. Do not issue this command list from an operator’s console or from a user-written command list.

Reference: Refer to Tivoli NetView for z/OS Administration Reference for the record formats and the NetView online help for descriptions of VPD command lists. Refer to Tivoli NetView for z/OS Automation Guide for additional information.

VPD Collection from a Single PU

The following list describes the procedures for collecting VPD from a single PU and its attached devices:

1. Specify a resource name and issue the VPDPU or VPDDCE command list.
2. The command list issues a VPDCMD command to solicit data from the specified resource, and waits for the response messages.
3. A PU responds with VPD for itself, or for itself and its attached devices.
4. The command list traps the response messages and saves the VPD, such as machine type, model number, and serial numbers, in command list variables.
5. When the completion message is received, the command list builds records and writes them to an external logging facility.
6. If any abnormal events occur before completion, a command list error message is issued and the command list terminates. An abnormal event can be a logging failure, an inactive VPDTASK, or an abend.
VPD Collection from a Single NetView Domain

The following list describes procedures for collecting VPD from a single NetView domain:

1. A NetView operator enters the following command:
   VPDALL CONFIG(ATCCON01),CREATE,CLIST(VPDACT),ADD

2. The VPDALL command list reads the specified nodes from the configuration member (ATCCON01, in this example) in VTAMLST. VPDALL extracts all the resource names from the VTAMLST nodes so that VPD can be collected. VPDALL then builds VPDPDU and VPDDCE entries in a command list called VPDACT. VPDALL does not support dynamic reconfiguration decks (DRDs) or DCEs on switched lines.

   **Note:** To collect data from the entire domain, the configuration member must contain the definitions for all the resources in the domain.

3. You can modify VPDACT by adding or deleting resource names.

4. When the VPDACT command list is executed, VPDLOGC is called to generate a START record. VPDACT then calls the VPDPDU and VPDDCE command lists and, after they are complete, calls the VPDLOGC to generate an END record.

Focal Point VPD Collection

Figure 43 shows a focal point NetView program for VPD.

![Diagram of NetView program]

**Figure 43. VPD Focal Point NetView Program**

The following steps describe the procedures for the collection of VPD for the sample focal point NetView program shown in Figure 43.

1. During installation, NV1 sets the common global variable SMFVPD to 200. NV2 sets the common global variable to 250.

   **Note:** CNMSTYLE sets the common global variable SMFVPD to 37.

2. NV1 is designated as a focal point NetView program for VPD collection. In the NetView automation table (DSITBL01), for NV1 only, uncomment the statement designated to drive the VPDXDOM command list.

   **Reference:** For more information, refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components.

3. Start DSIELTSK from the focal point NetView NV1.
Modifying NAM Command Lists

4. NV1 establishes a direct OST-to-NNT session with NV2 using the START DOMAIN command.
5. NV1 issues START VPDTASK.
6. NV1 issues ROUTE NV2, START VPDTASK.
7. NV1 issues ROUTE NV2, VPDACT. This causes the VPDACT command list in NV2 to run under an NNT.
8. In NV2, VPDACT verifies that it is running under an NNT, and generates the following message:
   
   \[ \text{MSG OPID X\$S VPDLOG 250 '1 STRING1 10 STRING2...'} \]

   where X\$S is a special string recognized by the NetView automation table.
9. When the VPDACT command list in NV2 writes the generated message to the operator in NV1, the message triggers the NetView automation table to execute the VPDXDOM command list in NV1.

Reference: Refer to [Tivoli NetView for z/OS Automation Guide](#) for additional information about the VPXDOM command list.
10. When VPDXDOM is entered, the message string is as follows:

    DSI039I MSG FROM OPID : X\$S VPDLOG 250 1 STRING1...

11. VPDXDOM verifies that NV1 set SMFVPD as a common global variable and changes SMFVPD from 250 (NV2) to 200 (NV1).
12. VPDLOGC logs the data records under NV1’s SMF record number 200.
13. Be sure that the cross-domain session stays active until after the VPD solicitation is completed.

Customization Considerations

You can customize the NetView-provided VPD command lists to suit your requirements.

When modifying network asset management command lists to build different record formats, do not exceed 256 bytes per record. NetView has a command string limitation of 240 characters. You can write a command processor to make full use of the VPD command.

Reference: Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for information about command processors.

If you are changing the SMF record format, you cannot use record number 37. You must globally define the SMF record number within the user-defined range of 128–255. If you are using SLR, you must write the SLR table to match your modified SMF record format.

Reference: Refer to NetView online help and [Tivoli NetView for z/OS Customization](#) Using REXX and the NetView Command List Language for limitations on the use of &WAIT and RESET, and for considerations regarding the issuance of a second network asset management command list and network asset management command while a previous network asset management command list is running.

To improve performance, you can do the following:
- Write a command list that reads in VPDACT to distribute the workload among several autotasks. Dividing the workload among several OSTs or autotasks
allows multiple VPDPU or VPDDCE entries to execute simultaneously. Otherwise, the VPDPU and VPDDCE entries are executed serially.

- Create several configuration members (for example, one member for each major node) or, using VPDALL, create several command lists.
- Make each command list run under several tasks, such as an OST and an autotask.
Modifying NAM Command Lists
Chapter 8. Customizing the Event/Automation Service

The event/automation service (E/AS) lets you manage all network events from the platform of your choice. You can use either the Tivoli Enterprise Console or Tivoli NetView for z/OS (NetView) to see a comprehensive list of events in your network.

The Event/Automation Service: Overview

The event/automation service consists of the following services:

- **Alert adapter service**
  The alert adapter service is an event adapter that converts Tivoli NetView for z/OS alerts to console events and forwards the events to the console event server. The alert adapter service collects filtered SNA alerts directly from the NetView hardware monitor and translates the alerts into appropriate console class or subclass instances. To receive alerts from NetView, the event/automation service registers with the NetView PPI. Filtered alerts from the NetView hardware monitor are sent over the PPI to the alert adapter service. All alerts to be converted will match the formats described in the *IBM Systems Network Architecture Management Services Reference*.

- **Message adapter service**
  The message adapter service is an event adapter that converts any message forwarded from NetView message automation into console events. The resulting events are forwarded to a designated event server. The message adapter collects filtered messages directly from the NetView automation table and translates the messages into appropriate console class or subclass instances. To receive messages from NetView, the event/automation service registers with the NetView PPI. Filtered messages from the NetView message automation table are sent over the PPI to the message adapter.

- **Event receiver service**
  The event receiver service receives events from a console server and converts them into SNA alerts. The converted alerts are then forwarded to the NetView hardware monitor where they are filtered and routed to the NetView automation table.

- **Alert-to-trap service**
  The alert-to-trap service is an SNMP sub-agent that converts Tivoli NetView for z/OS alerts to SNMP traps and forwards the traps to an SNMP agent. The alert-to-trap service collects filtered SNA alerts directly from the NetView hardware monitor and translates the alerts into appropriate SNMP trap instances. To receive alerts from NetView, the event/automation service registers with the NetView PPI. Filtered alerts from the NetView hardware monitor are sent over the PPI to the alert-to-trap service. All alerts to be converted will match the formats described in the *IBM Systems Network Architecture Management Services Reference*.

- **Trap-to-alert service**
  The trap-to-alert service receives events from an SNMP manager and converts them into SNA alerts. The converted alerts are then forwarded to the NetView hardware monitor where they are filtered and routed to the NetView automation table.
For general information about Tivoli event adapters, refer to the Tivoli Enterprise Console Adapters Guide.

**Starting the Event/Automation Service**

The event/automation service (E/AS) can be started from either the MVS system console using a startup procedure, or from the UNIX® System Services command shell using a command file. The sample startup procedure installed with the E/AS is IHSAEVNT. The command file used to start the E/AS from the UNIX System Services command shell is IHSAC000.

The environment that the E/AS is started from (either the MVS system console or the UNIX System Services command shell) determines certain operational characteristics of the E/AS as follows:

- The location of default configuration files.
- Whether certain startup parameters can be specified.
- The default output logs for trace/error data.

All other operational characteristics of the E/AS are the same regardless of the startup environment.

For information on installing and starting the E/AS, refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components.

**Customizing the Initialization of the Event/Automation Service**

The event/automation service (E/AS) has a number of configurable settings. A few must be set by the E/AS administrator in order for the E/AS to successfully initialize. For more information, refer to Tivoli NetView for z/OS Installation: Configuring Additional Components.

Configurable settings can be set by the E/AS administrator using configuration files, startup parameters, and E/AS modification commands. Some configurable settings can be set using more than one of these methods. Configurable settings are set in the following order, from highest priority to lowest:

- E/AS modification commands are issued to the E/AS after initialization. Any E/AS modification commands that affect a configurable setting change that setting for the duration of the current execution of the E/AS only.
- A configurable setting that is specified as an E/AS startup parameter.
- A configurable setting that is specified in a configuration file.
- The default value of the configurable setting.

E/AS modification commands are discussed fully in the Tivoli NetView for z/OS Command Reference.

**Defaults for Configurable Settings**

The following table lists all configurable settings and their defaults:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default</th>
<th>Overridden By</th>
</tr>
</thead>
<tbody>
<tr>
<td>E/AS PPI name</td>
<td>IHSATEC</td>
<td>PPI startup parameter, global initialization file PPI statement</td>
</tr>
<tr>
<td>Configuration File Name</td>
<td>Started with IHSAC000</td>
<td>Global Initialization File Name</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Global initialization file name</td>
<td></td>
<td>/etc/netview/global_init.conf</td>
</tr>
<tr>
<td>Alert adapter configuration file name</td>
<td></td>
<td>/etc/netview/alert_adpt.conf</td>
</tr>
<tr>
<td>Alert-to-trap configuration file name</td>
<td></td>
<td>/etc/netview/alert_trap.conf</td>
</tr>
<tr>
<td>Trap-to-alert configuration file name</td>
<td></td>
<td>/etc/netview/trap_alert.conf</td>
</tr>
<tr>
<td>Message adapter configuration file name</td>
<td></td>
<td>/etc/netview/message_adpt.conf</td>
</tr>
<tr>
<td>Event receiver configuration file name</td>
<td></td>
<td>/etc/netview/event_rcv.conf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Startup Parameter</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>IHSAINIT startup parameter</td>
<td></td>
</tr>
<tr>
<td>ALRTTCFG startup parameter, global initialization file ALRTTCFG statement</td>
<td></td>
</tr>
<tr>
<td>ALRTCFG startup parameter, global initialization file ALRTCFG statement</td>
<td></td>
</tr>
<tr>
<td>MSGCFG startup parameter, global initialization file MSGCFG statement</td>
<td></td>
</tr>
<tr>
<td>ERCVCFG startup parameter, global initialization file ERCVCFG statement</td>
<td></td>
</tr>
<tr>
<td>OUTSIZE startup parameter</td>
<td></td>
</tr>
<tr>
<td>-S startup option</td>
<td></td>
</tr>
<tr>
<td>-P startup option</td>
<td></td>
</tr>
<tr>
<td>-M startup option</td>
<td></td>
</tr>
<tr>
<td>-E startup option</td>
<td></td>
</tr>
<tr>
<td>Global initialization file TRACE statement, TRACE command</td>
<td></td>
</tr>
<tr>
<td>Global initialization file NOSTART statement</td>
<td></td>
</tr>
<tr>
<td>Global initialization file OUTPUT statement, OUTPUT command</td>
<td></td>
</tr>
<tr>
<td>Alert adapter and message adapter configuration file ServerLocation statement</td>
<td></td>
</tr>
<tr>
<td>Alert adapter and message adapter configuration file ServerPort statement</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 8. Customizing the Event/Automation Service 113
<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
<th>Configuration File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert adapter class definition statement (CDS) file name</td>
<td>Started with IHSAEVNT -- IHSAACDS</td>
<td>Alert adapter configuration file AdapterCdsFile statement</td>
</tr>
<tr>
<td></td>
<td>Started with IHSAC000 -- /etc/netview/alert_adpt.cds</td>
<td></td>
</tr>
<tr>
<td>Alert-to-trap adapter class definition statement (CDS) file name</td>
<td>Started with IHSAEVNT -- IHSAALCDS</td>
<td>Alert-to-trap configuration file AdapterCdsFile statement</td>
</tr>
<tr>
<td></td>
<td>Started with IHSAC000 -- /etc/netview/alert_trap.cds</td>
<td></td>
</tr>
<tr>
<td>Trap-to-alert adapter class definition statement (CDS) file name</td>
<td>Started with IHSAEVNT -- IHSATCDS</td>
<td>Trap-to-alert configuration file AdapterCdsFile statement</td>
</tr>
<tr>
<td></td>
<td>Started with IHSAC000 -- /etc/netview/trap_alert.cds</td>
<td></td>
</tr>
<tr>
<td>Event receiver class definition statement file name</td>
<td>Started with IHSAEVNT -- IHSAECDS</td>
<td>Event receiver configuration file AdapterCdsFile statement</td>
</tr>
<tr>
<td></td>
<td>Started with IHSAC000 -- /etc/netview/event_rcv.cds</td>
<td></td>
</tr>
<tr>
<td>Message adapter format file name</td>
<td>Started with IHSAEVNT -- IHSAMFMT</td>
<td>Message adapter configuration file AdapterFmtFile statement</td>
</tr>
<tr>
<td></td>
<td>Started with IHSAC000 -- /etc/netview/message_adpt.fmt</td>
<td></td>
</tr>
<tr>
<td>Maximum event cache size</td>
<td>64KB</td>
<td>Alert adapter and message adapter configuration file BufEvtMaxSize statement</td>
</tr>
<tr>
<td>Event cache HFS path</td>
<td>/etc/Tivoli/tec/cache</td>
<td>Alert adapter and message adapter configuration file BufEvtPath statement</td>
</tr>
<tr>
<td>Maximum event cache retrieval buffer size</td>
<td>64KB</td>
<td>Alert adapter and message adapter configuration file BufEvtRdblkLen statement</td>
</tr>
<tr>
<td>Amount to shrink the event cache</td>
<td>8KB</td>
<td>Alert adapter and message adapter configuration file BufEvtShrinkSize statement</td>
</tr>
<tr>
<td>Enable event buffering</td>
<td>YES</td>
<td>Alert adapter and message adapter configuration file BufferEvents statement</td>
</tr>
<tr>
<td>Rate to flush the event cache</td>
<td>0</td>
<td>Alert adapter and message adapter configuration file BufferFlushRate</td>
</tr>
<tr>
<td>Maximum number of events allowed in the event cache</td>
<td>0</td>
<td>Alert adapter and message adapter configuration file BufferEventsLimit</td>
</tr>
<tr>
<td>Tivoli Enterprise Console server connection mode</td>
<td>connection_less</td>
<td>Alert adapter and message adapter configuration file ConnectionMode statement</td>
</tr>
<tr>
<td>Maximum size of a console event</td>
<td>4096 bytes</td>
<td>Alert adapter and message adapter configuration file EventMaxSize statement</td>
</tr>
<tr>
<td>Console event filtering definitions</td>
<td>No filters defined</td>
<td>Alert adapter and message adapter configuration file Filter statement</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Configuration File/Statement</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>Console event filtering from event cache definitions</td>
<td>No filters defined</td>
<td>Alert adapter and message adapter configuration file FilterCache statement</td>
</tr>
<tr>
<td>Mode of console event filtering</td>
<td>OUT</td>
<td>Alert adapter and message adapter configuration file FilterMode statement</td>
</tr>
<tr>
<td>Broken connection retry interval</td>
<td>120 seconds</td>
<td>Alert adapter and message adapter configuration file RetryInterval statement</td>
</tr>
<tr>
<td>Console event forwarding debug mode</td>
<td>NO</td>
<td>Alert adapter and message adapter configuration file TestMode statement</td>
</tr>
<tr>
<td>Event receiver PPI name</td>
<td>NETVALRT</td>
<td>Event receiver configuration file NetViewAlertReceiver statement</td>
</tr>
<tr>
<td>Event receiver port number</td>
<td>0</td>
<td>Event receiver configuration file PortNumber statement</td>
</tr>
<tr>
<td>Enable PortMapper for the event receiver</td>
<td>YES</td>
<td>Event receiver configuration file UsePortmapper statement</td>
</tr>
<tr>
<td>Alert-to-trap SNMP agent IP location</td>
<td>loopback</td>
<td>Alert-to-trap service configuration file Hostname statement</td>
</tr>
<tr>
<td>Alert-to-trap community name</td>
<td>public</td>
<td>Alert-to-trap service configuration file Community statement</td>
</tr>
<tr>
<td>Alert-to-trap Enterprise Object ID</td>
<td>1.3.6.1.4.1.1.1588.1.3</td>
<td>Alert-to-trap service configuration file Enterpriseoid statement</td>
</tr>
<tr>
<td>Trap-to-alert PPI name</td>
<td>NETVALRT</td>
<td>Trap-to-alert service configuration file NetViewAlertReceiver statement</td>
</tr>
<tr>
<td>Trap-to-alert port number</td>
<td>162</td>
<td>Trap-to-alert service configuration file PortNumber statement</td>
</tr>
</tbody>
</table>

### Customizing the Event/Automation Startup Parameters

Startup parameters can be specified for the IHSAEVNT startup procedure if you are starting the E/AS from the MVS system console, or on the UNIX System Services command line for the IHSAC000 command. Startup parameters follow two general formats:

- parameter=value
- -option[value]

Either format can be used from either startup environment unless otherwise noted below. However, in order to pass the option/value format to the IHSAEVNT startup procedure the list of options and values must be passed in a single parameter/value format. The IHSAEVNT startup procedure provides the following parameter to accomplish this:

**OELINE**

An example of using the OELINE parameter to pass option/value format startup parameters to the IHSAEVNT startup procedure follows:

```
s IHSAEVNT,OELINE='-opt1 value1 -opt2 value2...'
```

Use single quotes to surround the options and values passed with the OELINE parameter.
The option/value format is a case-sensitive format. Ensure you specify the following options exactly as they are described. Values are not translated to uppercase. For some options, only the option is specified. There is no corresponding value associated with the option.

The startup parameters are:

**INITFILE**=**file** or **-i** **file**

This startup parameter specifies the name of the global initialization file in **file**. If you use the **INITFILE**=**file** format, the file is a 1–8 character PDS member name that is associated with the IHSSMP3 data set definition from the IHSAEVNT startup procedure. This format is not valid when starting the E/AS from the UNIX System Services command line. If you use the **-i** **file** format, the file is a full MVS data set or HFS path and file name. Surround MVS data set names with single quotes to make them fully-qualified data sets. For example:

- **INITFILE**=IHSAINIT
- **-i** 'NETVIEW.V5R1M0.SCNMUXCL(IHSAINIT)'
- **-i** /etc/netview/global_init.conf

**MSGCFG**=**file** or **-m** **file**

This startup parameter specifies the name of the message adapter configuration file in **file**. If you use the **MSGCFG**=**file** format, the file is a 1–8 character PDS member name that is associated with the IHSSMP3 data set definition from the IHSAEVNT startup procedure. This format is not allowed when starting the E/AS from the UNIX System Services command line. If you use the **-m** **file** format, the file is a full MVS data set or HFS path and file name. Surround MVS data set names with single quotes to make them fully-qualified data sets. For example:

- **MSGCFG**=IHSAMCFG
- **-m** 'NETVIEW.V5R1M0.SCNMUXCL(IHSAMCFG)'
- **-m** /etc/netview/message_adpt.conf

**ALRTCFCFG**=**file** or **-a** **file**

This startup parameter specifies the name of the alert adapter configuration file in **file**. If you use the **ALRTCFCFG**=**file** format, the file is a 1–8 character PDS member name that is associated with the IHSSMP3 data set definition from the IHSAEVNT startup procedure. This format is not allowed when starting the E/AS from the UNIX System Services command line. If you use the **-a** **file** format, the file is a full MVS data set or HFS path and file name. Surround MVS data set names with single quotes to make them fully-qualified data sets. For example:

- **ALRTCFCFG**=IHSAACFG
- **-a** 'NETVIEW.V5R1M0.SCNMUXCL(IHSAACFG)'
- **-a** /etc/netview/alert_adpt.conf

**ALRTTCFCFG**=**file** or **-a** **file**

This startup parameter specifies the name of the alert-to-trap service configuration file in **file**. If you use the **ALRTTCFCFG**=**file** format, the file is a 1–8 character PDS member name that is associated with the IHSSMP3 data set definition from the IHSAEVNT startup procedure. This format is not allowed when starting the E/AS from the UNIX System Services command line. If you use the **-a** **file** format, the file is a full MVS data set or HFS path and file name. Surround MVS data set names with single quotes to make them fully-qualified data sets. For example:

- **ALRTTCFCFG**=IHSATTCFG
- **-a** 'NETVIEW.V5R1M0.SCNMUXCL(IHSATTCFG)'
- **-a** /etc/netview/alert_trap.conf
TALRTCFG=file or -t file
This startup parameter specifies the name of the trap-to-alert service configuration file in file. If you use the TALRTCFG=file format, the file is a 1–8 character PDS member name that is associated with the IHSSMP3 data set definition from the IHSAEVNT startup procedure. This format is not allowed when starting the E/AS from the UNIX System Services command line. If you use the -t file format, the file is a full MVS data set or HFS path and file name. Surround MVS data set names with single quotes to make them fully-qualified data sets. For example:
TALRTCFG=IHSATCFG
-t 'INETVIEW.V5R1M0.SCNMUXCL(IHSATCFG)'
-t /etc/netview/trap_alert.conf

ERCVCFG=file or -e file
This startup parameter specifies the name of the event receiver configuration file in file. If you use the ERCVCFG=file format, the file is a 1–8 character PDS member name that is associated with the IHSSMP3 data set definition from the IHSAEVNT startup procedure. This format is not allowed when starting the E/AS from the UNIX System Services command line. If you use the -e file format, the file is a full MVS data set or HFS path and file name. Surround MVS data set names with single quotes to make them fully-qualified data sets. For example:
ERCVCFG=IHSAECFG
-e 'INETVIEW.V5R1M0.SCNMUXCL(IHSAECFG)'
-e /etc/netview/event_rcv.conf

PPI=ppiname or -p ppiname
This startup parameter specifies the name of the E/AS PPI mailbox in ppiname. For example:
PPI=IHSATEC
-p IHSATEC

OUTSIZE=size or -O size
This startup parameter enables output log wrapping and specifies the maximum size of the output log file, in kilobytes. If size is specified as 0, output log wrapping is disabled. For more information on E/AS output, refer to "Event/Automation Service Output" on page 118.
OUTSIZE=0
-O 0

-M msgfile
This startup parameter specifies the location of the E/AS messages file. msgfile specifies a full MVS dataset or HFS path and filename. Surround MVS data set names with single quotes to make them fully qualified data sets. For example:
-M 'INETVIEW.V5R1M0.SCNMUXMS(IHSMMSG1)'
-M /usr/lpp/netview/msg/C/ihsamsg1

-P This startup parameter is not allowed when starting the E/AS from the IHSAEVNT startup procedure. It is used to disable the forwarding of MVS system console messages to the UNIX System Services command shell if the E/AS was started under the UNIX System Services command shell. By default, a message that is issued to the MVS system console is also issued at the UNIX System Services command shell.

-S This startup parameter is not allowed when starting the E/AS from the IHSAEVNT startup procedure. This startup parameter specifies that the alert adapter service and message adapter service are to be started in
secure mode. For more information on secure mode, refer to "Running Event/Automation Service in Secure Mode" on page 122.

-E path

This startup parameter is not allowed when starting the E/AS from the IHSAEVNT startup procedure. This startup parameter specifies the HFS path of trace/error log files. path specifies an HFS path. For example:

-E /tmp

Customizing the Event/Automation Service Configuration Files

The E/AS uses six configuration files. These files and their default names are:

- The global initialization file
  IHSAINIT or /etc/netview/global_init.conf
- The alert adapter configuration file
  IHSAAACFG or /etc/netview/alert_adpt.conf
- The alert-to-trap service configuration file
  IHSAAATCF or /etc/netview/alert_trap.conf
- The trap-to-alert service configuration file
  IHSATCFG or /etc/netview/trap_alert_trap.conf
- The message adapter configuration file
  IHSAMCFG or /etc/netview/message_adpt.conf
- The event receiver configuration file
  IHSAEFCFG or /etc/netview/event_rcv.conf

The global initialization file is used to change configurable settings that are required by all five services. Each of the other configuration files are used to change configurable settings that are specific to the services. The statements within these files must all be contained on one line. Each of these files can have comments. Comment statements begin with the pound sign (#).

If the E/AS is started from the IHSAEVNT startup procedure, by default the 8-character PDS name specified is used to locate the file. The file must be in a dataset specified by the IHSSMP3 dataset definition statement from the IHSAEVNT startup procedure. If the E/AS is started from the UNIX System Services command shell, by default the HFS name specified is used to locate the file.

Every statement in a configuration file can be a comment. If all configuration file statements are comments, the configuration file will not change any of the configurable settings. Each of the four configuration files must exist for the E/AS to properly initialize, even if the file contains nothing but comments. The E/AS will not initialize if it cannot locate a configuration file.

For more information on the configuration file statements, refer to the Tivoli NetView for z/OS Administration Reference.

Event/Automation Service Output

All event/automation service (E/AS) output can be sent to one or both of two destinations: the generalized trace facility (GTF) and the E/AS output logs. By default, data is sent to the E/AS output logs. The destination of E/AS output can be changed using the OUTPUT command or the OUTPUT statement in the global
There is an output log associated with each of the three services, and an output log associated with the entire E/AS address space. If output log wrapping is disabled, these output logs are physically represented by one system file. If output log wrapping is enabled, these output logs are physically represented by two system files - a primary file and a secondary file.

When wrapping is disabled, all output log data is written to the primary file.

When wrapping is enabled, the wrap size is used to limit the total amount of bytes that can be written to either the primary or the secondary file. When this wrap size is exceeded, the current file being used for output log output (either the primary or secondary file) is closed, and the file that was not previously in use (either the primary or the secondary) is opened for further logging. Whenever an output log is opened, all data that was previously in the log is destroyed. Therefore, the maximum amount of output log data available is 2 times the wrap size (both the primary and secondary files are full), and the minimum amount of output log data available is the wrap size (a switch has just occurred to either the primary or secondary file, destroying all data previously in that file).

For more information on setting output log wrapping, refer to the OUTSIZE parameter on page 117.

**Event/Automation Service Output Log Names**

When the E/AS is started using the IHSAEVNT startup procedure, the names of the output logs are defined by the following data set definition statements within the IHSAEVNT procedure:

- IHSC (primary file) and IHSCS (secondary file) – defines the output log files for the E/AS address space.
- IHSA (primary file) and IHSAS (secondary file) – defines the output log files for the alert adapter service.
- IHSM (primary file) and IHSMS (secondary file) – defines the output log files for the message adapter service.
- IHSE (primary file) and IHSES (secondary file) – defines the output log files for the event receiver service.
- IHSL (primary file) and IHSLS (secondary file) – defines the output log files for an alert-to-trap service.
- IHST (primary file) and IHSTS (secondary file) – defines the output log files for a trap-to-alert service.

If output log wrapping is disabled, the data set definition for the secondary file need not be present in the IHSAEVNT startup procedure, but it is a good practice to leave it in. The data set definition for the primary file must always be present.

By default, the output log files are set to the IHSAEVNT jobs SYSOUT data set. If SYSOUT data sets are used for the output log files, output log wrapping is disabled. If you want to enable output log wrapping, you must change these data set definitions to reference an MVS sequential data set or HFS file.

**Note:** There is no restriction placed on the type of file that you specify in the data set definition statements in the IHSAEVNT startup procedure. However, it is recommended that you do not define a PDS member as an output log file.
due to synchronization problems that may occur when trying to write data to the PDS member. You also should use a different file for each data set definition statement.

Unless you have been instructed to run with tracing enabled by a Tivoli service representative, it is recommended that you use the default SYSOUT data sets that are specified in the sample IHSAEVNT startup procedure and do not enable output log wrapping.

When the E/AS is started using IHSAC000 in the UNIX System Services command shell, the names of the output log files are defined as follows:

- The files must be HFS files. By default, the path of the files is /tmp. This path can be changed using the -E startup option. Refer to this option on page 118.
- controlp.log (primary file) and controls.log (secondary file) are the names of the output log files for the E/AS address space. These names cannot be changed.
- alertp.log (primary file) and alerts.log (secondary file) are the names of the output log files for the alert adapter service. These names cannot be changed.
- alrttrpp.log (primary file) and alrttrps.log (secondary file) are output error log files for the alert-to-trap adapter service.
- trapalrtp.log (primary file) and trapalrts.log (secondary file) are output error log files for the trap-to-alert service.
- messagep.log (primary file) and messages.log (secondary file) are the names of the output log files for the message adapter service. These names cannot be changed.
- eventrcvp.log (primary file) and eventrcvs.log (secondary file) are the names of the output log files for the event receiver service. These names cannot be changed.

The E/AS creates these output log files if they do not exist.

Note: Unless you have been instructed to run with tracing enabled by a Tivoli service representative, it is recommended that you do not enable output log wrapping.

Types of Event/Automation Service Output Data

The E/AS generates two types of output data: trace data and error data.

Trace data is only generated if tracing is enabled. By default, tracing is disabled. To change trace settings, refer to the Tivoli NetView for z/OS Command Reference for information on the TRACE command, and the Tivoli NetView for z/OS Administration Reference for information on the global initialization file TRACE statement.

In general, tracing should only be used if you are requested to do so by a Tivoli service representative.

Error data is composed of MVS system console messages and output log only messages. In general, any error condition detected by the E/AS results in an MVS console message. This console message is also written to E/AS output. To aid in problem determination, additional messages may also be written to E/AS output. These output log only messages that were not issued to the MVS system console may give more detail concerning the problem.
The combination of system console and output log only messages should allow you to resolve most E/AS problems without the aid of a Tivoli service representative.

Not all MVS console messages describe error conditions. There are a number of informational messages that are also issued by the E/AS and sent to E/AS output logs.

**Format of Event/Automation Service Output Data**

When an output log file is initially opened, the first entry in the output log file is composed of the name of the output file followed by a date/time string in the format:

```
day month date time year
```

The following example shows the header for the message adapter service primary output log file, assuming that the E/AS was started from the IHSAEVNT startup procedure:

```
IHSM Fri Feb 20 10:45:55 1998
```

All other E/AS output data is composed of a header followed by the specific data.

The header is composed of:

- A date/time string in the format:
  
  day month date time year

- The module name of the module where the message was issued
- The line number within the module where the message was issued
- The type of message, which can be one of the following:
  - LOW - Specifies this message is issued if the LOW or higher level of tracing has been enabled.
  - NORMAL - Specifies this message is issued if the NORMAL or higher level of tracing has been enabled.
  - VERBOSE - Specifies this message is issued if the VERBOSE level of tracing has been enabled.
  - CONSMMSG - Specifies this is an MVS console message.
  - LOGONLY - Specifies this is a message that accompanies an MVS console message, but is issued only to E/AS output.
  - IP - Specifies this message is issued if IP tracing has been enabled.

An example of an E/AS output entry follows:

```
-----date/time-------- module line msgtype -> specific data
```

In this example, the console message IHS0075I was issued from the reported E/AS module at the specified time and date.

**Note:** Module and line numbers are for use by a Tivoli service representative if additional problem determination is needed.
Customizing Alert and Message Routing from NetView

When NetView is installed, the routing of alert and message data to the event/automation service is by default disabled. NetView automation table statements and hardware monitor filter commands are used to enable the routing of alert and message data to the event/automation service. Refer to the Tivoli NetView for z/OS Automation Guide for complete information on enabling and customizing the routing of alerts and messages from NetView to the E/AS.

Running More Than One Event/Automation Service

Multiple E/AS address spaces can be active at the same time. In most cases, you will only need one E/AS; however, you may need more than one for any of the following reasons. You want:

- A subset of alerts or messages to be translated and sent to a different Tivoli Enterprise Console server.
- Alerts or messages to be translated and sent to more than one console server.
- A subset of console events to be translated and sent to a different NetView alert receiver.
- Console events to be translated and sent to more than one NetView alert receiver.

If you run more than one E/AS, the E/AS PPI mailbox name must be unique for each. All other configurable settings can be shared between the E/AS invocations. However, you should consider changing the following configurable settings between each E/AS invocation:

- If you use more than one event receiver service, only one should register with the PortMapper. Others should specify a port number and disable the use of PortMapper. If more than one event receiver attempts to use the PortMapper, only the last event receiver to access PortMapper will actually be registered; all other registrations for the other event receivers will be lost. A warning message is written to the MVS system console when the event receiver PortMapper registration is overwritten.
- The E/AS output log files should be unique for each E/AS invocation. Otherwise, data from one E/AS will be interleaved in the same output log file as data from another E/AS. If you are using the IHSAEVNT startup procedure to execute the E/AS, and the output log files are to SYSOUT data sets, then these datasets are automatically unique for each E/AS invocation.

Running Event/Automation Service in Secure Mode

The E/AS relies on the Tivoli client framework for secure mode operation. For more information, refer to the Tivoli FrameWork Planning and Installation Guide. For more information on events and secure mode operations, refer to Event Integration Facility (EIF) User’s Guide.

Advanced Customization - Translating Data

In addition to the configuration files that the E/AS uses to define operational characteristics, each E/AS service uses a translation file that contains a set of rules that tell the service how to translate the incoming data into a Tivoli Enterprise Console event or a SNMP trap. Each translation file is a text-readable file that can be customized.
The translation files used by the services of the E/AS have two different formats. The alert adapter, alert-to-trap, trap-to-alert and event receiver services use a class definition statement (CDS) translation file. The message adapter service uses a message format translation file.

To customize these translation files, you should have an understanding of the format of console events, SNMP traps, or both. For more information about console events, refer to the Tivoli Enterprise Console User’s Guide.

For additional information on SNMP traps, refer to the appropriate z/OS documentation for SNMP agent.

Class Definition Statement Files

The class definition statement (CDS) file defines how to construct Tivoli Enterprise Console events from the information that is sent by a data source. For the alert adapter service and the alert-to-trap service, the data source is NetView. For the event receiver service, the data source is a Tivoli Enterprise Console server. For the trap-to-alert service, the data source is an SNMP trap manager. The statements in this file are referred to as class definition statements (CDS’s). Class definition statements are rules that enable the service to map the incoming data that it receives to a console event.

Note: The event receiver service, alert-to-trap service, and trap-to-alert service will further process the console event that is produced using these class definition statements to turn it into an alert or SNMP trap. Refer to “Event Receiver Post-CDS Processing” on page 140 for more information on creating alerts from console servers. Refer to “Alert-to-Trap Post-CDS Processing” on page 158 for more information on creating traps from alerts. Refer to “Trap-to-Alert Post-CDS Processing” on page 157 for more information on creating alerts from SNMP traps.

A CDS file is composed of one or more CDS’s. Each CDS can include a SELECT, FETCH and a MAP segment that specify the rules for mapping data into a Tivoli Enterprise Console event. These rules allow for selecting an event class based on the incoming data, fetching additional data for creating the console event, and mapping the information collected from the incoming event into event attributes for the outgoing console event.

A CDS has the following general format:
CLASS <class_name> SELECT <select_statements> FETCH <fetch_statements> MAP

The CDS file also supports comment lines beginning with the comment sign (#).

The keywords in a CDS provide the following kinds of information:

CLASS
The <class_name> defines the class name that will be used on the outgoing console event if the incoming data matches this CDS.

SELECT
Consists of one or more <select_statement> entries that incoming data must satisfy to match, or select, this CDS. Select statements are evaluated in the order that they appear in the SELECT segment. If all of the <select_statements> of a particular CDS are satisfied, then the incoming data matches the corresponding CDS. Otherwise, the
adapter tries to match the incoming data with the next CDS. If the incoming data cannot be matched with any CDS, it is discarded.

**FETCH**

Consists of zero or more `<fetch_statement>` entries that are used to retrieve additional pieces of data from the incoming data in order to build the event attributes in the map segment. The FETCH segment is used to retrieve data not retrieved by the SELECT segment, or to change the data that was retrieved by the SELECT segment.

**MAP**

Consists of zero or more `<map_statement>` entries that specify how to build the event attributes of the Tivoli Enterprise Console event instance using the service's default data, user-defined constant data and pieces of data retrieved in the SELECT and FETCH segments.

For the alert adapter service, each class of event defined in the `.baroc` file of the service on an event server must match one or more CDS in the CDS file. The CDSs specify how to map incoming data to the class and event attributes of the outgoing Tivoli Enterprise Console event instance. If you change or add classes or event attributes in the CDS file, you must make a corresponding change to the `.baroc` file on the event server.

For the event receiver service, the outgoing Tivoli Enterprise Console event is never sent to an event server; it is a pseudo-event that is processed further to create an alert. Therefore, there is no corresponding `.baroc` file on an event server for any Tivoli Enterprise Console events created from the event receiver's CDS file.

Each CDS is evaluated in the order it appears in the CDS file. An incoming event is mapped to the class specified by the first CDS whose SELECT segment is evaluated successfully. When more than one CDS is provided for a given class of event, the CDS with the most restrictive SELECT segment should appear first in the CDS file.

If the `<class_name>` is equal to `*DISCARD*`, any incoming data matching the SELECT segment should be discarded. Note that data will also be discarded if it does not match any CDS. However, if a given type of incoming data must always be discarded, it is more efficient to define a `*DISCARD*` statement and put it at the beginning of the CDS file rather than letting the adapter evaluate all CDS's before finally discarding the event.

**Encoding Incoming Event Data**

Incoming event data is encoded by the service into name/value pairs. Name/value pairs are also referred to as attributes. For any incoming event, all of the attributes are placed in a list that is then used in the SELECT, FETCH and MAP segments. The service selects which, if not all, of the incoming data to encode into name/value pairs, see the specific service encoding discussion later in this section.

The name part of the attribute is a text string. There are two types of names - *generic* and *keyword*.

*Generic* names are text strings created by the services. A service may create these names internally, or it may create them from information provided in the incoming raw data; in either case, the method used by the service to create attribute names will be discussed with the specific service encoding later in this chapter.
Keywords have the format $\text{keyword}$. Data that is commonly provided in the incoming datastream to the service is usually coded into keywords rather than generic names. The actual keyword name is never derived from the incoming data, but rather is defined by the service.

The main difference between keywords and generic names is how the names are used in processing the CDS file. Keywords provide faster data lookup during CDS file processing. Otherwise, keywords and generic names are nothing more than data tags, with keywords prefaced with ‘$’.

The value part of the attribute is also a text string. Again, the service will assign this text string based on data in the raw event.

**Alert Adapter Service and Alert-to-Trap Service Data Encoding**

The alert adapter and alert-to-trap service uses keyword attributes exclusively for their data encoding. The following table lists each of the keyword attribute names used and how the value field is assigned from the incoming alert data.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ALERT_CDPT}$</td>
<td>A 2-byte hexadecimal value taken from the alert description code field of the generic alert data subvector, or the resolution description code field of the resolution data subvector.</td>
</tr>
<tr>
<td>$\text{ORIGIN}$</td>
<td>A character string with the name/type hierarchy pairs from the Hierarchy Name List or Hierarchy/Resource List subvectors. The string contains the hierarchy in the form: resnam1/typ1,resnam2/typ2,resnam3/typ3,resnam4/typ4,resnam5/typ5 Only the number of pairs in the actual subvector are used.</td>
</tr>
<tr>
<td>$\text{SUB_ORIGIN}$</td>
<td>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: resnamx/typx where $x$ is the number of the last pair in the list.</td>
</tr>
<tr>
<td>$\text{hostname}$</td>
<td>The netid.nau node name of the SNA node where the alert originated. Could be a NetView/390 node, an AS/400® node, etc.</td>
</tr>
<tr>
<td>$\text{ADAPTER_HOST}$</td>
<td>The IP name of the host where the NetView/390 alert adapter resides.</td>
</tr>
<tr>
<td>$\text{DATE}$</td>
<td>The date when the alert was received by the NetView/390 alert adapter. In format: MMM HH:MM:SS, e.g. OCT 10 12:08:30.</td>
</tr>
<tr>
<td>$\text{SEVERITY}$</td>
<td>FATAL, CRITICAL, etc. The alert type field from the Generic Alert Data subvector, or the event type, is used to determine the severity. Refer to Table 18 on page 127.</td>
</tr>
<tr>
<td>$MSG</td>
<td>The Long Error Description:Long Probable Cause message that describes the problem. This message is similar to the ALERT DESCRIPTION:PROBABLE CAUSE message displayed on the NPDA ALERTS-DYNAMIC panel.</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$ADAPTER_HOST_SNANODE</td>
<td>The netid.domainid node name of the NetView that sent the alert to the NetView/390 alert adapter.</td>
</tr>
<tr>
<td>$EVENT_TYPE</td>
<td>E.g. PERMANENT, TEMPORARY, etc.. For Generic Alerts, it is obtained by inspecting the Alert Type byte of Generic Alert Data subvector. It matches the EVENT TYPE displayed on the NPDA EVENT DETAIL panel.</td>
</tr>
<tr>
<td>$ARCH_TYPE</td>
<td>GENERIC_ALERT, GENERIC_RESOLUTION, or NONGENERIC_ALERT. NMVT Alert Major Vectors contain a Generic Alert Data subvector are GENERIC_ALERTs. NMVT Resolution Major Vectors are GENERIC_RESOLUTIONs. All other alerts are NONGENERIC_ALERTs.</td>
</tr>
<tr>
<td>$PRODUCT_ID</td>
<td>The hardware or software product set identifier (PSID) of the alert or event sender. This can be 4, 5, 7, or 9 characters. Pertains to all generic alerts and some non-generic alerts.</td>
</tr>
<tr>
<td>$ALERT_ID</td>
<td>An 8-character hexadecimal value assigned by the sender to designate an individual alert condition. The value will always be 00000000 for resolution alerts. Pertains only to generic alerts (including resolutions).</td>
</tr>
<tr>
<td>$BLOCK_ID</td>
<td>The code used to identify the IBM hardware or software associated with the alert. See the NetView Resource Alerts Reference manual. Pertains only to non-generic alerts.</td>
</tr>
<tr>
<td>$ACTION_CODE</td>
<td>A code that provides an index to predefined screens. Pertains only to non-generic alerts. For non-generic alerts, the combination of the block id and action code uniquely identify the sending product.</td>
</tr>
<tr>
<td>$SELF_DEF_MSG</td>
<td>Text extracted from Self-defining Text Message Sv31.</td>
</tr>
<tr>
<td>$EVENT_CORREL</td>
<td>Correlators extracted from MSU Correlation Sv47. These correlators correlate alerts to other alerts. That is, you may have two or more alerts that pertain to the same underlying problem and such alerts are correlated by Sv47. The tecad_snaevent.rls file on the Tivoli Enterprise Console server contains rules which discard alerts that have already been reported.</td>
</tr>
</tbody>
</table>
Correlators extracted from Incident Identification subvectors. These correlators correlate alerts to resolutions. The tecad_snaevent.rls file on the Tivoli Enterprise Console server contains rules which CLOSE all correlated alerts when a resolution is received.

A correlator that has meaning only to the alert adapter.

Always assigned the string "[N/A]".

Always assigned the string "[N/A]".

Always assigned the string "[N/A]".

Non-keyword attributes can also be assigned by users in the NetView address space. Refer to [Tivoli NetView for z/OS Automation Guide](#) for more information on how to customize alerts forwarded from NetView. Using this method, any attribute name/value pair can be created and used by the CDS file process. The alert adapter and trap-to-alert service do not use generic attributes other than when they are assigned within NetView.

The value for the severity event attribute is determined by mapping an alert type (or event type) to a severity. The table below shows this mapping. The hexadecimal byte is the alert type field from the generic alert data subvector.

**Table 18. Alert Types and Severities**

<table>
<thead>
<tr>
<th>Alert Type</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01, PERMANENT</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>0x02, TEMPORARY</td>
<td>HARMLESS</td>
</tr>
<tr>
<td>0x03, PERFORMANCE</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x04, INTERVENTION REQ'D</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>0xNN, CUSTOMER APPLICATION</td>
<td>MINOR</td>
</tr>
<tr>
<td>0xNN, END USER GENERATED</td>
<td>MINOR</td>
</tr>
<tr>
<td>0xNN, SUMMARY</td>
<td>HARMLESS</td>
</tr>
<tr>
<td>0xNN, INTENSIVE MODE REC</td>
<td>HARMLESS</td>
</tr>
<tr>
<td>0x09, AVAILABILITY</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>0x0A, NOTIFICATION</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x0B, ENVIRONMENT</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>0x0C, INSTALLATION</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x0D, OPERATION/PROCEDURE</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x0E, SECURITY</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>0x0F, DELAYED RECOVERED</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x10, PERMANENT AFFECTED</td>
<td>MINOR</td>
</tr>
<tr>
<td>0x11, IMPENDING PROBLEM</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x12, UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>0xNN, HELD</td>
<td>MINOR</td>
</tr>
<tr>
<td>0x14, BYPASSED</td>
<td>WARNING</td>
</tr>
<tr>
<td>0x15, REDUNDANCY LOST</td>
<td>WARNING</td>
</tr>
</tbody>
</table>
Table 18. Alert Types and Severities (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x16</td>
<td>SITUATION WARNING</td>
</tr>
<tr>
<td>0xNN</td>
<td>RESENT ALERT</td>
</tr>
<tr>
<td>0xNN</td>
<td>RESOLVED PROBLEM</td>
</tr>
<tr>
<td>0xNN</td>
<td>UNSUPPORTED TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alert-to-Trap Service Data Encoding

The alert-to-trap service constructs enterprise traps (type 6). The CDS file enables customization of the specific code field in the trap. This is done by supplying a value for the SPECIFIC keyword in the MAP sections of the CDS file.

The basic approach of the alert-to-trap service is to construct Tivoli Enterprise Console event keyword/value pairs from the alert and then map the keyword/value pairs (other than SPECIFIC) into SNMP OCTET strings to be included as variable bind data in the resulting trap. Both the keyword and the value are included in the resulting OCTET string.

The alert-to-trap service has access to the alert-adapters keyword attributes, and these can be used in SELECT, MAP and FETCH statements. However, not all alert adapter attributes are applicable to SNMP traps.

The CLASS names in class definition statements are not used in the traps built by the alert-to-trap servicer. However, the CLASS name is still required to satisfy CDS syntax rules, and it is useful when you document the trap you are constructing.

Trap-to-Alert Service Data Encoding

The trap-to-alert service receives an SNMP trap as its incoming data. This data is encoded into both keyword attributes and generic attributes.

The following table lists the keyword attributes created by the trap-to-alert service.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ORIGIN_ADDR</td>
<td>The value is a string containing the internet address (in dotted decimal notation) from which the trap came. Note that when the sample datagram forwarding daemon is used, the value is the internet address of the host in which the daemon is running.</td>
</tr>
<tr>
<td>$ORIGIN_PORT</td>
<td>The value is a string containing the number of the port (in decimal) at the origin address from which the trap came. Note that when the sample datagram forwarding daemon is used, the value is the number of the port over which the daemon forwarded the trap.</td>
</tr>
<tr>
<td>$SNMP_VERSION</td>
<td>The value is a string containing the number (in decimal) indicating which SNMP version was implemented at the agent that sent the trap. This determines how the trap was formatted. The value for SNMPv1 is &quot;0&quot;.</td>
</tr>
</tbody>
</table>

The following table lists the generic attributes created by the trap-to-alert service from the SNMP trap data that is not a variable binding. All data is converted to a
character string before assigning it to the generic attribute name.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>community</td>
<td>The value of the SNMP trap community field.</td>
</tr>
<tr>
<td>enterpriseOID</td>
<td>The value of the SNMP trap enterpriseOID field.</td>
</tr>
<tr>
<td>agent_address</td>
<td>The value of the SNMP trap agent address field.</td>
</tr>
<tr>
<td>generic_trap</td>
<td>The value of the SNMP trap generic trap field.</td>
</tr>
<tr>
<td>specific_trap</td>
<td>The value of the SNMP trap specific trap field.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The value of the SNMP trap timestamp field.</td>
</tr>
</tbody>
</table>

The variable binding data is created directly from the variable binding data. The variable binding name becomes the name of the generic attribute, and the variable binding data is converted to a character string if it is not already a character string and assigned to the generic attribute. When more than one variable binding within an SNMP trap contains the same name, the name and index is appended to the name to create the generic attribute name. For example, if the variable binding name 1.3.6.1.4.1.2.2.1.3.1.0 occurred 3 times within the same SNMP trap, the generic attribute names that are created as a result would be as follows:

1.3.6.1.4.1.2.2.1.3.1.0
1.3.6.1.4.1.2.2.1.3.1.0<1>
1.3.6.1.4.1.2.2.1.3.1.0<2>

Event Receiver Service Data Encoding
The event receiver service receives a Tivoli Enterprise Console event as its incoming data. This data is encoded into both keyword attributes and generic attributes. This encoding is very straightforward since the data is already in the name/value form of an attribute. Every event attribute name in the incoming console event becomes the name of a generic attribute in the attribute list, and the corresponding event attribute value becomes the value of the attribute. The className of the event is encoded as the value of the $CLASSNAME keyword attribute. As such, the event receiver creates one keyword attribute, $CLASSNAME, and as many generic attributes as there are event attribute/value pairs in the incoming console event.

SELECT Segment of a Class Definition Statement
The SELECT segment of a CDS is composed of one or more `<select_statement>` entries. Each `<select_statement>` entry has the following format:

```<n>: ATTR(<a_op>, <a_op_value>),
    VALUE(<v_op>, <v_op_value>);```

A `<select_statement>` is satisfied if an attribute is found in the list of attributes provided by the service that fulfills the conditions specified by the `ATTR` and `VALUE` expressions of the `<select_statement>`. An attribute must be found for each
<select_statement> for the SELECT segment to be satisfied. If a SELECT segment is not satisfied, the entire CDS is ignored and processing continues with the next CDS in the CDS file.

<n> Is the identification number of the <select_statement>. n can be any valid integer. Each <select_statement> must have a unique identification number; this identification number is used in further processing of the CDS.

**ATTR**
Specifies the name of an attribute, in <a_op_value> and a modifying condition on the attribute name in <a_op>. The ATTR expression is mandatory in the SELECT statement. The list of attributes created by the service from the incoming data are searched until an attribute is found that has a name field which matches the condition expressed by the ATTR expression.

<a_op>
Modifies the ATTR name and can have one of the following values:

=  Specifies that the attribute name in <a_op_value> must match the name of an attribute in the attribute list.

PREFIX  Specifies that the attribute name in <a_op_value> must be a prefix of the name of an attribute in the attribute list.

SUFFIX  Specifies that the attribute name in <a_op_value> must be a suffix of the name of an attribute in the attribute list.

<a_op_value>
Specifies the name of an attribute. The attribute list is searched sequentially and the ATTR <a_op> expression is applied to each attribute name field until a matching attribute is found.

By default, <a_op_value> is a string. However, <a_op_value> can also be a variable. Variables are described below.

When specified as a string, <a_op_value> must be enclosed in double quotes (") if the string contains a blank character or if it is all digits (0 through 9). The following examples show possible <a_op_value> strings:

```
hello
$ORIGIN
"hello, world"
"12"
```

When specified as a variable, <a_op_value> can contain any of these types of variables:

**Keyword**
A keyword provided by the event adapter, for example, $ORIGIN.

**Name**
Name variables are assigned the value of the name field of an attribute that has satisfied a previous <select_statement>ATTR expression. A name variable is specified as $Nn, where n is the number of the <select_statement> that the desired attribute satisfied (for example, $N2).

**Value**
Value variables are assigned the value of the value field of an attribute that has satisfied a previous <select_statement> VALUE
expression. A value variable is specified as $Vn$, where $n$ is the number of the $<select_statement>$ that the desired attribute satisfied (for example, $V5$).

The following example of an ATTR expression looks for a generic name that is equal to user1. If the service has provided an attribute named user1, the ATTR expression will be satisfied.

 ATTR(=,"user1")

The following example of an ATTR expression looks for a keyword that is equal to $SORIGIN$. If the service has provided an attribute named $SORIGIN$, the ATTR expression will be satisfied.

 ATTR(=,$SORIGIN$)

**VALUE**

This expression is optional. For the attribute in the attribute list that matches the associated ATTR expression, the value of the attribute is subjected to a match based on the information in the VALUE expression.

$<v_{op}>$

Modifies the VALUE expression and can have one of the following values:

$=$ Specifies that the VALUE expression in $<v_{op\_value}>$ must match the value of an attribute in the attribute list.

PREFIX Specifies that the VALUE expression in $<v_{op\_value}>$ must be a prefix of the value of an attribute in the attribute list.

SUFFIX Specifies that the VALUE expression in $<v_{op\_value}>$ must be a suffix of the value of an attribute in the attribute list.

$!= $ Specifies that the VALUE expression in $<v_{op\_value}>$ must not be equal to the value of an attribute in the attribute list.

$<v_{op\_value}>$

Specifies the value of an attribute. By default, $<v_{op\_value}>$ is a string. However, $<v_{op\_value}>$ can also be a variable.

When specified as a string, $<v_{op\_value}>$ must be enclosed in double quotes (") if the string contains a blank character or if it is all digits (0 through 9). The following examples show possible $<v_{op\_value}>$ strings:

```
hello
$SORIGIN
"hello, world"
"12"
```

When specified as a variable, $<v_{op\_value}>$ can contain any of these types of variables:

**Keyword**
The keyword is assigned a constant value (either a string or a number), and the keyword can be used to reference the value.

**Name**
Name variables are assigned the value of the name field of an attribute that has satisfied a previous $<select_statement>$ ATTR expression. A name variable is specified as $Nn$, where $n$ is the number of the $<select_statement>$ that the desired attribute satisfied (for example, $N2$).
**Value**

Value variables are assigned the value of the value field for an attribute that has satisfied a previous `<select_statement>` VALUE expression. A value variable is specified as $Vn$, where $n$ is the number of the `<select_statement>` that the desired attribute satisfied (for example, $V5$).

The following example of a VALUE expression looks for an attribute with a value that is prefixed with `Serial`:

```
VALUE(PREFIX,"Serial")
```

A valid match for this VALUE expression is `Serial1`.

**SELECT Segment Evaluation**

For an entire SELECT segment to be matched, an attribute must be matched for each of the `<select_statement>` expressions in that SELECT segment. More than one attribute in the attribute list may satisfy a `<select_statement>`. The first one in the attribute list that satisfies the statement is used for further CDS processing.

If the SELECT segment is satisfied, the class name of the SELECT segment is used for the outgoing Tivoli Enterprise Console event. Processing of the event continues with the FETCH segment, unless the class is *DISCARD*, in which case the event is discarded. If the incoming data satisfies no SELECT segment of a CDS in the CDS file, the incoming data is discarded.

Each time a `<select_statement>` is evaluated successfully, the two variables $Nn$ and $Vn$ are created. These variables, along with the adapter-provided keywords, can be used in any subsequent SELECT, FETCH or MAP segment.

**FETCH Segment of a Class Definition Statement**

The SELECT segment of a CDS retrieves attribute names and values from the incoming data, but it does not allow for changes to the selected pieces of information. In some circumstances, it is necessary to extract a substring out of an attribute value or to provide user-defined variables. The FETCH segment in a CDS allows you to do this.

The FETCH segment is composed of one or more `<fetch_statement>` expressions. Each `<fetch_statement>` has the following format:

```
<n>: <expression>
```

where

- `<n>` is an identification number of the `<fetch_statement>`. `<n>` can be any valid integer. Each `<fetch_statement>` must have a unique identification number. A `<fetch_statement>` results in the value of `<expression>` being assigned to a new variable, $Fn$, where $n$ is the identification number of the `<fetch_statement>`.

- `<expression>` is one of the following:
  - A string
  - Any output value from the SELECT segment (such as adapter-provided keywords and SELECT segment variables.
  - Any output from a previous `<fetch_statement>`
  - A substring with any combination of strings, SELECT segment output, and `<fetch_statement>` output.
An example of a FETCH segment using substrings is:

1: SUBST ($V2, 1, 5);

This statement uses the value of the variable $V2, as assigned from <select_statement> number 2, and assigns the substring represented by the first 4 characters of $V2 to the variable $F1.

The output of the FETCH segment is the set of fetch variables $Fn.

**MAP Segment of a Class Definition Statement**

The MAP segment of a CDS creates the event attributes and associated values that will be put in the outgoing Tivoli Enterprise Console event.

The MAP segment is composed of one or more <map_statement> expressions. Each <map_statement> has one of the following formats:

- `<slot name> = <string>;`
- `<slot name> = <variable>;`  
- `<slot name> = PRINTF(<format_string>, <var1>, ..., <varn>);`

- `<slot name>`
  - The name of any event attribute. For the alert adapter service, this should be an event attribute that corresponds to an event attribute in the service’s .baroc file on an event server. For the event receiver service, this should be an event attribute that is allowed by the event receivers post-CDS file processing.

- `<string>`
  - Any character string.

- `<variable>`
  - Any variable passed to the MAP segment from the SELECT or FETCH segments, such as adapter-defined keywords or segment variables.

- **PRINTF**
  - Specifies a format that allows the value of the event attribute to be formatted using a C-style printf() format string. This format string currently supports only the %s format specifier.

- `<var>`
  - Can contain either a `<string>` or a `<variable>`.

Here is an example of a MAP segment:

```map
MAP
  origin = $V2;
  hostname = $HOSTNAME;
  msg = PRINTF("The origin is %s", $V2);
```

In this example, the `origin` event attribute would be given the value of the SELECT segment variable $V2. The `hostname` event attribute would be given the value of the $HOSTNAME keyword. Assuming the value of the variable $V2 is `NV390SP/SP`, the `msg` event attribute would be given the value "The origin is NV390SP/SP" (the double quotes are not included in the value).

The output of the map process is a list of event attribute name/value pairs that are used to generate the outgoing Tivoli Enterprise Console event that will either be sent to the event server or used for post CDS-file processing.
MAP_DEFAULT Section of the Class Definition Statement Files

Some event attributes, like source and hostname, will probably have a constant value for all the Tivoli Enterprise Console events generated by a given service. To avoid repeating identical map statements in many CDS’s, the CDS file supports a MAP_DEFAULT section. This section defines event attribute name/value pairs for all CDS’s in the CDS file. The event attributes that are defined in this global definition section can be overridden by specific definitions in a CDS.

Here is an example of a MAP_DEFAULT section:

```plaintext
MAP_DEFAULT
    origin = $ORIGIN;
    sub_origin = $SUB_ORIGIN;
    msg = $MSG;
END
```

In some cases, you may want to put CDSs into more than one CDS file and have them all be used by a service. To enable this, an extension to normal CDS file processing has been added for the E/AS services. The %INCLUDE statement allows additional CDS files to be embedded within the current CDS file. The %INCLUDE keyword cannot be preceded by blank characters, and it must be followed by a separator of one blank character. Following the separator is the file name of the CDS file to be opened. This file name is either a 1 to 8 character PDS member name that is associated with the IHSSMP3 data set definition, or a complete file name that is preceded by the backslash (\) character. The maximum number of CDS file members that can be opened at the same time is 20; this represents the maximum number of nested %INCLUDE statements that are valid.

The following example shows the %INCLUDE statement syntax. Assume that the file named IHSAACD1 contains the single statement:

```
sub_origin = $SUB_ORIGIN;
```

in this example:

```plaintext
MAP_DEFAULT //Statements from IHSAACDS
    source = NW390ALT;
    origin = $ORIGIN;
%INCLUDE IHSAACD1 //New file with sub_origin statement
    hostname = $HOSTNAME;
    adapter_host = $ADAPTER_HOST;
END //Continuation of IHSAACDS
```

For more information about Tivoli Enterprise Console events, refer to the Tivoli Enterprise Console Adapter’s Guide. For an example of using CDS’s, refer to the IHSAACDS or IHSAECDS sample shipped with the event/automation service. These are the default translation files used for the alert adapter and event receiver services, respectively.

Message Format Files

The FMT file defines how the message adapter service constructs Tivoli Enterprise Console events from message information that is sent by NetView. The statements in this file are referred to as format specification statements (FSS). Format specification statements are rules that allow the service to map the incoming message data that it collects from NetView to an outgoing console event.

The following sections describe the syntax of the message adapter service’s format specifications and how format specifications are mapped into events.
Encoding Incoming Event Data

For the message adapter service, the incoming data is a message string. This message text string is matched against format specifications in the FMT file. The primary piece of information, therefore, is the message string itself.

Like a CDS file, the job of the FMT file is to allow the user to customize the outgoing console event based on the incoming message data. This method does not encode the data into attributes; however, there are certain event attribute names that receive default information from the incoming message data.

The table below lists each of the default event attribute names and their corresponding default values. If the value for the event attribute is not actually present in the incoming data, then the default event attribute value will be the null string. ANY event attribute that is listed in the map rules portion of a format specification statement has a default value; if it is not provided in the incoming data, its default value is the null string ("").

<table>
<thead>
<tr>
<th>Event attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>The netid.domainid node name of the NetView where the message originated.</td>
</tr>
<tr>
<td>sub_origin</td>
<td>The job number associated with the message. If a job number is not available for the message, the value defaults to a null string (&quot;&quot;).</td>
</tr>
<tr>
<td>hostname</td>
<td>Same as the origin event attribute.</td>
</tr>
<tr>
<td>adapter_host</td>
<td>The IP name of the host where the Event/Automation Service is running.</td>
</tr>
<tr>
<td>date</td>
<td>The date and time that the message was sent from the NetView automation table. In format: MMM HH:MM:SS, e.g. OCT 10 12:08:30.</td>
</tr>
<tr>
<td>msg_id</td>
<td>The first token of the message. In most cases, this token is the actual message identifier.</td>
</tr>
<tr>
<td>severity</td>
<td>Inferred from the last character of the msg_id. The translation of this character to a value for this event attribute is: A, E, S CRITICAL T FATAL anything else WARNING</td>
</tr>
<tr>
<td>msg</td>
<td>The message text, which includes msg_id as the first token.</td>
</tr>
<tr>
<td>adapter_host_snanode</td>
<td>The netid.domainid node name of the NetView that sent the message to the Message Adapter service.</td>
</tr>
<tr>
<td>multiline_msg</td>
<td>The second and succeeding message lines from the message. If the message is contained in one line, the value of multi_line message is N/A.</td>
</tr>
<tr>
<td>jobname</td>
<td>The jobname associated with the message. If a jobname is not available for the message, the value of jobname defaults to a null string (&quot;&quot;).</td>
</tr>
</tbody>
</table>
Default event attributes and values can also be assigned by users in the NetView address space. Refer to the *Tivoli NetView for z/OS Automation Guide* for more information about customizing messages forwarded from NetView. Using this method, any attribute name/value pair can be created and used by the FMT file process.

**Format Specifications**

The FMT file is made up of 1 or more FSS. An FSS has the following three parts:

- The format header has the keyword `FORMAT` followed by the class name. This is optionally followed by the `FOLLOWS` keyword and a previously defined `FORMAT` class name. If the incoming message matches this FSS, the class name following the `FORMAT` keyword will be used on the outgoing Tivoli Enterprise Console event.

- The format content has a format string optionally followed by a list of map rules. The format string performs a function similar to the `SELECT` segment of a CDS file; that is, it matches the incoming message to a particular FSS. The map rules perform a function similar to the `MAP` segment in the CDS file; that is, they assign values to event attributes.

- The `END` keyword completes the FSS.

The format header, the format string, each map rule, and the `END` keyword must begin on a new line.

The `FOLLOWS` relationship is used to enable a specific FSS to be built from more generic ones. When format B follows format A, B inherits all of the map rules (but not the format string) from A. Format B can define any additional map rules, but any map rules redefined by B are not inherited from A. Format B can override inherited map rules by redefining them.

Messages that are forwarded by NetView typically have a common format consisting of a message identifier and message-specific text. These message components can be represented in the format string using a component specifier notation that is very similar to the C-style `printf()` notation. This `printf()` notation is similar to the notation used in CDS files.

The following format string describes the entire class of messages that are produced by the NetView automation table:

```
%s*
```

Input messages are tokenized into constants and blanks. A constant is any consecutive string of non-blank characters. Component specifiers allow the constants and blanks to be grouped into more complex “tokens” when trying to match an FSS against a specific message. The current allowable component specifiers are:

- `%s` Matches one constant in the input message
- `%s*` Matches zero or more constants in the input message
- `%s+` Matches one or more constants in the input message

The format string `DSI%s %s*` is taken from the default message adapter FMT file shipped with the E/AS, and is used in the following discussion to demonstrate the usage of format strings.

As an example of matching a message to the `DSI%s %s*` format specification, consider the following NetView message:

```
DSI002I INVALID COMMAND: 'BADCOMMAND'
```
The component specifiers and matches are as follows:

\[
\text{DSI DSI} \\
\%s 002I \\
\%s* INVALID COMMAND: 'BADCOMMAND'
\]

The DSI002I message has some constant parts and some variable parts. That is, certain parts of the message (constant parts) will be the same for any DSI002I message that is generated. The constant parts of the message are:

\[
\text{DSI002I INVALID COMMAND: ' '}
\]

The variable part of the message is:

\[
\text{BADCOMMAND}
\]

Note that the first constant part of the message goes all the way to the first single quote ('') in the message. The second single quote is the beginning of the second constant part of the message, which also happens to be the last character in the message. The data inside of the single quotes is all variable.

The following message is an example of another DSI002I message with different variable parts:

\[
\text{DSI002I INVALID COMMAND: 'WORSE COMMAND'}
\]

In this case, the variable part is composed of two words and a space -- WORSE COMMAND.

The format string \texttt{DSI}\%s %s* can be specialized for the DSI002I message as follows:

\[
\text{DSI %s INVALID COMMAND: '%s*'}
\]

Using the DSI002I message above, the component specifiers and matches are as follows:

\[
\text{DSI DSI} \\
\%s 002I \\
\%s* INVALID COMMAND: 'INVALID COMMAND: ' \\
\%s* WORSE COMMAND \\
' '
\]

The blank characters that separate the words of a message must also be present in the format string. A single space character in the format string will match any number of blank characters in the message.

Suppose the space between the colon (:) and the quote ('') is deleted in the specialized DSI002I format string given above:

\[
\text{DSI %s INVALID COMMAND: '%s*'}
\]

In this example, the format string would no longer match DSI002I messages. However, in the following example, the NetView message \texttt{would} match the format specification, since all consecutive blanks from both the input message and the format specification are boiled down to a single blank character:

\[
\text{DSI %s INVALID COMMAND: ' %s*'}
\]

Care should be taken when using arbitrary length repeater component specifiers (%s* and %s+). The following format string does not make much sense:

\[
\text{This is not a good format %s %s*}
\]
The first %s* will match everything through the end of the message, and the second %s* will never match anything. It may appear that this does not matter, but the importance becomes apparent when map rules are discussed in "Map Rules".

The following format string, however, is meaningful:
This is a good format %s* : %s*
The first %s* will match everything up to the first colon (:) and the second %s* will match everything through the end of the message.

From the above examples, you can see that you can specialize a generic format to match a more specific event by either replacing component specifiers with constants or by restricting the arbitrary length repeater specifiers to a fixed length by using constants to terminate the specifier.

Map Rules

The service translates incoming message data into an event class with event attribute name/value pairs, and sends this information to an event server. As with the alert adapter service, a .baroc file at the event server must be present to match the outgoing Tivoli Enterprise Console events created by the message adapter service.

The event class is determined by matching an input message to an FSS as described previously. However, once the class is determined. Values must be assigned to the event attribute names. These values can come from a variety of places, such as from the message itself, from default event attributes provided by the service, or from specifications within the FMT file. Map rules define how event attributes are assigned values.

The map rule portion of the format string consists of zero or more lines that contain a .baroc file event attribute name followed by a value specifier. The value specifiers are one of four types:

- $i$, where $i$ indicates the position of a component specifier in a format string. Each component specifier is numbered from 1 to the maximum number of component specifiers in the format string. For example, in the specialized format specification for the DSI002I message given above, the %s* component specifier would be referred to in the map rules as $2. The value of a $i value specifier, also referred to as a variable value specifier, is the portion of the input message that was consumed by the component specifier. These variables are very similar to the variables output from the SELECT and FETCH segments in the CDS file.

- A constant string. The value of the event attribute is the specified string. If the string is a single constant, it can be specified without surrounding double quotes ("). Otherwise, double quotes must be used.

- A PRINTF statement. This mechanism allows you to compose more complex event attributes from other event attributes. The PRINTF statement consists of the keyword PRINTF followed by a C-style printf() format string and a list of event attribute names. The printf() format string currently only supports the %s conversion specifier. The values of the event attributes that are used in the PRINTF statement must also have been derived from either the $i value specification or a constant string value specification. They cannot be derived from another PRINTF value specification. The value of the argument event attributes will be used to compose a new constant string according to the
printf() format string. This constant string becomes the value of the event attribute. This value specifier is very similar to the PRINTF MAP segment format in the CDS file.

- DEFAULT. This keyword indicates that the adapter should use its internal logic to derive the value of the indicated event attribute. For example, the incoming message data contains the hostname (netid.nau) where the message originated. If the hostname event attribute is therefore set to the value DEFAULT, netid.nau will be the value of the hostname event attribute. This is similar to the use of keywords in the alert adapter service.

Since DEFAULT is a keyword, a constant map whose value is the string DEFAULT must be specified in double quotes (" ").

You should specify only one map rule for each .baroc file event attribute in any one format specification. The map rule can be inherited from a more generic format specification (using the FOLLOWS keyword), or it can be explicitly defined on the format specification that directly matches the input message. Since the service does not have access to the .baroc file, which resides on the event server, care must be taken to make sure that the format specifications agree with the corresponding .baroc file definitions. If an event attribute name is misspelled in a map rule, for example, the service will not report any error and will send the event to the event server as usual. However, the event will be meaningless to the event server.

There can be attributes in the incoming message that do not directly correspond to any .baroc file event attributes. However, the service might need to use these values to compose PRINTF style constant strings. This data needs to be assigned to temporary event attributes, which can then be used in the PRINTF value specification but does not allow the event attribute to be sent over to the event server as an independent event attribute name/event attribute value pair. Temporary event attributes are designated with a minus sign (-) immediately preceding the event attribute name in the map rule. These temporary event attributes are not .baroc file event attributes. Do not use the minus sign (-) when referring to the temporary event attribute in the PRINTF specification.

%INCLUDE Statements

The %INCLUDE statement allows additional FMT files to be imbedded within the current FMT file. The %INCLUDE keyword cannot be preceded by blank characters, and it must be followed by a separator of one blank character. Following the separator is the file name of the FMT file to be opened. This file name is either a 1 to 8 character PDS member name that is associated with the IHSSMP3 dataset definition, or a complete file name that is preceded by the backslash (\) character. The maximum number of FMT file members that can be opened at the same time is 20; this represents the maximum number of nested %INCLUDE statements that are allowed.

Format File Example

The following sample will be used to demonstrate the concepts discussed above; this example was taken (and modified somewhat) from the message adapter services default message format file (IHSAMFMT):

```
FORMAT NV390MSG_Event
%INCLUDE
source NV390MSG
origin DEFAULT
desctext "This string will be overridden"
END
```
FORMAT NV390MSG_NetView_NCCF Follows NV390MSG_Event

DSI%s %s*
sub_source "NetView NCCF"
msgnumber $1
temp1 $2
desctext PRINTF("Got a DSI message: %s", temp1)
END

%Include morefmts

Using this format file, assume that the following message is received by the service:

DSI002I INVALID COMMAND: 'A BAD COMMAND'

This message will match the NV390MSG_NetView_NCCF format specification defined above if the additional format statements included in MOREFMTS do not specify another format specification that this message can match on. Remember, matches on the FSS in the FMT file begin with the last FSS in the file and progress toward the first FSS until a match occurs.

With this match, the source event attribute will be assigned the string value NV390MSG. The origin event attribute will be assigned whatever default the event adapter associates with this event attribute. The desctext event attribute will be assigned the string This string will be overridden initially. These event attributes are all assigned with the more generic NV390MSG_Event FSS, from which the NV390MSG_NetView_NCCF FSS follows.

The sub_source event attribute will be assigned the value of NetView NCCF. The msgnumber event attribute will be assigned the value 002I (which was dissected from the input message on the first %s* specification). The -temp1 temporary event attribute will be assigned the string INVALID COMMAND: 'A BAD COMMAND' (which was dissected from the input message on the second %s* specification). This temporary variable is then used with the PRINTF value specifier to override the desctext event attribute with the string Got a DSI message: INVALID COMMAND: 'A BAD COMMAND'.

All of the event attributes, with the exception of the -temp1 event attribute, will be used to build the outgoing Tivoli Enterprise Console event. The classname for the event will be NV390MSG_NetView_NCCF, the name of the most specifically matched FSS.

For more information about Tivoli Enterprise Console events, refer to the Tivoli Enterprise Console Adapter's Guide. For an example of using FSS, refer to the IHSAMFMT sample shipped with the event/automation service.

Event Receiver Post-CDS Processing

For the alert adapter service and message adapter service, translation files are used to translate incoming service specific data into a Tivoli Enterprise Console event. For the event receiver, a CDS file will be used to go in the opposite direction (translate an event into a NetView alert).

To do this, the processing of the CDS file by the event receiver will be modified slightly from the processing that is done on the file by the alert adapter service. Syntactically, all of the information that is discussed in "Class Definition Statement Files" section above is still true for the event receiver CDS file. The event receiver treats the event that is output by the CDS file process as a pseudo event; that is,
the event is not meant to be sent to a Tivoli Enterprise Console server, but rather is parsed for certain specific event attributes that are encoded into the NMVT.

The Input Attribute List

The incoming Tivoli Enterprise Console event is encoded into an attribute list as described in the service specific encoding section later in this chapter. In addition to the $CLASSNAME keyword created when the incoming event is parsed, there are additional keywords created for the input attribute list by the event receiver. The following list describes the additional keywords:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NMVT_TYPE</td>
<td>The type of the NMVT to be created (alert or resolution). This keyword is modified by the NMVT_TYPE event attribute. The NMVT_TYPE event attribute can have a value of ALERT or RESOLVE.</td>
<td>ALERT</td>
</tr>
<tr>
<td>$CDS_GROUP</td>
<td>This keyword contains values in the set &quot;GROUP001&quot;, GROUP002&quot;, ... &quot;GROUP999&quot;. The value of the CONTINUE event attribute is used to set the value of this keyword. For more information on the $CDS_GROUP keyword and the CONTINUE event attribute, see the section &quot;Matching Multiple CDS’s to Create the Pseudo Event&quot; later in this chapter.</td>
<td>GROUP001</td>
</tr>
<tr>
<td>$BUILD_SV31LIST</td>
<td>Assigned the value of the BUILD_SV31LIST event attribute. This event attribute can have a value of NO or YES. When the alert is built, the value of this keyword is used to determine whether subvector 31s are to be added for each event attribute/value pair in the original Tivoli Enterprise Console event. For more information about the $BUILD_SV31LIST keyword and the BUILD_SV31LIST event attribute, see the section &quot;Building the SV 31s containing the Original Event&quot; later in this section.</td>
<td>YES</td>
</tr>
</tbody>
</table>

The Output Pseudo Event

Like any Tivoli Enterprise Console event, this pseudo event contains a class name, followed by event attribute/value pairs. Note that because this event will never be
sent to a console, there is no `.baroc` file on any console server that corresponds to these events. In general, a CDS file enables any event attribute/value pair and any class name to be put into the pseudo event. Even though any class name and event attribute/value pair can be placed in the pseudo event, the event receiver only uses certain predefined event attribute names to translate the event into an alert. Any other event attributes are ignored.

The Pseudo Event Class name
The event receiver does not use the pseudo event class name for translating the Tivoli Enterprise Console event. All of the CDSs in the event receiver CDS file can have the same name; however, for ease of organizing the various CDSs and debugging, it is recommended that you use a different class name for each CDS in the CDS file. The convention used in the sample CDS file shipped with the E/AS is to group the CDSs that are associated with producing a particular subvector within the NMVT together and prefacing them with a common character string. The end of the class name can then have some unique designation to make it unique.

An example:
```
CLASS SV05_1
  ...
END
CLASS SV05_2
  ...
END
CLASS SV05_3
  ...
END
...
```

In this example, the SELECT segments (not shown) in each CDS statement will cause a different subvector 05 to be built. The class name for the SV 05 that is eventually built will have a unique name that identifies it as an SV 05. Again, this information is only used only for visual organization and debugging.

The NMVT_TYPE event attribute
You can specify the type of NMVT, whether it is an alert or a resolution, by coding the NMVT_TYPE event attribute in the MAP segment of a CDS. There are two valid values for this event attribute: `RESOLVE` and `ALERT`. The value of this event attribute is copied to the `$NMVT_TYPE` keyword.

The SV event attribute
This event attribute is the main vehicle for creating the subvectors that are to be placed into the NMVT.

The event attribute name must be prefixed with `SV`; the rest of the event attribute name can be any character string. `SV05`, `SVAA` and `SVNONSENSE` are all recognized as SV event attributes. Again, for clarity and debugging, it is recommended that the event attribute names contain the number of the subvector being created -- `SV05`, `SV92`, `SV05_1`.

An SV event attribute value contains the full subvector (including the length and subvector key). The values that are assigned to SV event attributes in the MAP segment of a CDS are interpreted as character strings; the event receiver will
decode the numeric character string into the hexadecimal values that are to be used in the alert. An example of a subvector event attribute from the sample CDS file:

```
SV05 = "080509100004E3C5C30040";
```

The value in the SV05 is a character string with hexadecimal characters. The event receiver translates this character string into true numeric format for inclusion in the NMVT. The event receiver does not validate this subvector. The subvector that is placed into the NMVT is similar to the following:

```
080509100004E3C5C30040
```

Following the general CDS file syntax, if the event attribute value contains only the digits in the range of 0–9, the value must be enclosed within double quotations to be interpreted as a string. The previous example has alphabetic characters (representing the hexadecimal values A-F) in it, so it was not necessary to enclose the event attribute value within quotes. It is a good habit, though, to enclose SV event attributes within double quotations.

**Disabling Hexadecimal String Translation**

In some cases, you may want to add a character string that is *not* a hexadecimal value to the subvector string. As previously described, by default the event receiver attempts to translate the event attribute value hexadecimal string into numeric format under the assumption that the string is a sequence of hexadecimal characters (0–9, A–F). In the previous example, the hexadecimal string E3C5C3 is, in EBCDIC, TEC.

To specify the string TEC directly within the event attribute value, enclose the string within <> braces. The braces must have escape characters preceding them; the escape character is #. Using this convention, for example, the string is as follows:

```
SV05 = "080509100004#<TEC#>0040"
```

This event attribute value would produce exactly the same NMVT subvector as the first example, as follows:

```
080509100004E3C5C30040
```

The braces indicate to the event receiver that the data enclosed within the braces is not a hexadecimal string number that needs to be converted, but the string is to be placed directly into the NMVT.

**Using Attribute List Data in the Output Subvector**

Event attributes can be assigned the value of a CDS variable ($V, $N, $F variables), the value of a keyword, or generic attribute from the attribute list. When using these variables, it is likely that the value of the variable should not be converted. Also, it is likely that these variables do not contain the entire coded subvector entirely within the variable. To handle this, the PRINTF style of MAP statement assignment is useful.

Extending the SV 05 example introduced above, assume that the string TEC is the value of the $V2 variable generated by a SELECT segment. To produce an identical SV 05 for the NMVT, enter the following:

```
SV05 = PRINTF("080509100004#<%s#>0040", $V2);
```

Using the PRINTF syntax, the %s format specifier is substituted with the value of the $V2 variable, which is TEC. The escaped braces tell the event receiver not to translate the TEC string into numeric format, and again the following subvector produced is identical to that produced in the first two examples:
Any time you need to assign data that came from the original Tivoli Enterprise Console event to the output subvector, you will likely need to use the PRINTF syntax with string translation disabled. However, it is possible that the incoming event has, as an event attribute value, the string E3C5C3 instead of the string TEC. In this case, use the following string to produce the desired NMVT subvector:

```c
SV05 = PRINTF("0B0509100004%E3C5C30040", $V2);
```

If you continued to disable the hexadecimal string translation, your output subvector is similar to the following:

```
0B0509100004C5F3C3F5C3F30040
```

Each of the six characters E, 3, C, 5, C and 3 is left in their character state (C5, F3, C3, F5, C3 and F3).

**Automatic Subvector/Subfield Length Calculation**

In the initial SV 05 example

```
SV05 = "0B0509100004E3C5C30040"
```

the length of the subvector was coded directly into the string. Because there is no variable information in the subvector, the length is coded directly into the event attribute value within the CDS MAP segment. The length of the subvector might not be known when the CDS file is created if variable data is used.

Consider the following example that inserts attribute list data into the subvector:

```
SV05 = PRINTF("0B0509100004#<%s#>0040", $V2);
```

In this example, the value of the $V2 variable was TEC; therefore, it has a length of 3. This was used to calculate the total subvector length (0B), the subfield 10 length (09), and the resource name length (04). In reality, the length of the value of the $V2 variable will be unknown until the event arrives.

To enable the event receiver to calculate the length of a portion of the subvector string, use curly braces {} around that portion of the string. The curly braces must be escaped with the escape character #. The curly braces are removed from the string when the length is calculated, but the opening curly brace is the place holder in the subvector string for the length field.

Modify the previous example as follows:

```
SV05 = PRINTF("#(05#(1000#(#<%s#>#)0040#)#)", $V2);
```

Following is a step-by-step translation of this event attribute. The PRINTF substitution is first as follows:

```
SV05 = "#(05#(1000#(#TEC#)0040#)#)"
```

At this stage, the output subvector is similar to the following:

```
...E3C5C3...
```

Where the ellipsis represent all data yet to be translated into the subvector. Next, the segment #[TEC#] is used to calculate the length of the resource name entry.

The output subvector is as follows:

```
...04E3C5C3...
```
The first #{ is replaced with the length of the segment, the matching #{ is removed. Next, the segment #{100004TEC0040#} is used to calculate the length of the subfield 10 entry.

The output subvector is as follows:
09100004E3C5C30040

Again, the #{ is replaced with the length of the segment, the matching #{ is removed. Finally, the segment #{05091000100004TEC0040#} is used to calculate the length of the entire subvector 05.

The final output subvector is as follows:
080509100004E3C5C30040

The **BUILD_SV31LIST Event Attribute**
The entire original Tivoli Enterprise Console event is, by default, coded into SV 31s and attached to the NMVT. The class name, each event attribute/value pair, and the END designator are coded into separate SV 31s. The `BUILD_SV31LIST` event attribute enables the user to control whether this list of SV 31s is to be added to the NMVT. When the pseudo event is completed, if a `BUILD_SV31LIST` event attribute is present in the event AND has a value of `NO`, the SV 31 list is excluded. Otherwise, the SV 31 list is included.

The **CONTINUE Slot**
This event attribute is used to enable the matching of multiple CDSs to create a single pseudo event. A full description of this multiple pass process on the CDS file is given in "Matching Multiple CDSs to Create the Pseudo Event". This event attribute can have a value of either `NEXT` or `GROUP xxx`, where `xxx` is a value in the range of 000–999.

The value of this event attribute is used to update the value in the `$CDS_GROUP` keyword. This keyword defaults to a value of `GROUP001`. If the value of a `CONTINUE` event attribute is `NEXT`, `$CDS_GROUP` is updated by adding a 1 to the three numeric digits at the end of the value. If the current value of `$CDS_GROUP` is `GROUP001`, and a `CONTINUE` event attribute with a value of `NEXT` is encountered in a MAP segment, the new value of the `$CDS_GROUP` keyword will be `GROUP001`.

If the value of the `CONTINUE` event attribute is `GROUP xxx`, this value is used to replace the `$CDS_GROUP` value only if the numeric digits in the event attribute value are greater than the numeric digits in the current `$CDS_GROUP` value.

The **SF21 Slot**
This event attribute is used to override the code point in any Subfield 21s that are in the SV 31s used to send the original Tivoli Enterprise Console event. The value of this event attribute must be as follows:

```
attributeName=codepoint
```

Where `attributeName` is the name of any generic attribute in the input attribute list, and `codepoint` is a 2-digit hexadecimal string that defines the value to be placed in the SF 21 that is associated with the SV31 for the named generic attribute.

Like the SV event attribute, the SF21 must only be prefixed with the string SF21; any characters after this prefix are ignored.
Matching Multiple CDSs to Create the Pseudo Event

A major difference between the way that CDS files are processed by the event adapters and how the CDS file is processed by the event receiver is the number of CDSs that can be matched to produce a single Tivoli Enterprise Console event (or pseudo event, in the case of the event receiver).

The One-Pass Method

The event adapters will run through all of the statements in a CDS file until either one statement is matched or the end of the file is reached without a match. The MAP segment of that single matching CDS is then used to create the event attribute/value pairs that will go into the outgoing Tivoli Enterprise Console event.

Although this same one-pass process could be used to create any of the pseudo events that will be translated into an alert, it could result in a cumbersome CDS file. To illustrate this, consider the following example.

From an incoming event, create an alert that has various combinations of SV 05s and SV 92s based on event attribute/value pairs in the event. For the SV 05 creation, you look for the presence of two event attributes -- resource1 and resource2. The following four CDSs map the SV 05:

CLASS SV05_1
  SELECT
    1: ATTR(=,resource1);
    2: ATTR(=,resource2);
  MAP
    SV05 = PRINTF("#{05#{1000#{#<%s#>#}0084#{#<%s#>#}0040#}#}", $V1, $V2);
END

CLASS SV05_2
  SELECT
    1: ATTR(=,resource1);
  MAP
    SV05 = PRINTF("#{05#{1000#{#<%s#>#}0040#}#}", $V1);
END

CLASS SV05_3
  SELECT
    1: ATTR(=,resource2);
  MAP
    SV05 = PRINTF("#{05#{1000#{#<%s#>#}0084#}#}", $V1);
END

CLASS SV05_4
  SELECT
    1: ATTR(=,$CLASSNAME);
  MAP
    SV05 = "#{05#{1000#{#<NONE#>#}0084#}#}";
END

To produce the four different event attributes, different SELECT segments must be used to inspect for the presence of these event attributes; therefore, there will be 4 different CDSs in the CDS file. Only one of these SV 05s will be in the pseudo event. The last CDS uses the $CLASSNAME keyword as a default. This keyword will always be present, so the last CDS will be selected if none of the other CDSs are matched.

The SV 92 subvector depends on value of another event attribute, severity. There are three different values for the severity event attribute that could result in
different SV 92s, and a fourth SV 92 that is created if the severity event attribute contains none of these values. These CDSs are as follows:

CLASS SV92_1
SELECT
  1: ATTR(=, severity), VALUE(=, FATAL);
MAP
  SV92 = "0892010001FE0300000000"
END

CLASS SV92_2
SELECT
  1: ATTR(=, severity), VALUE(=, WARNING);
MAP
  SV92 = "0892010011FE0300000000"
END

CLASS SV92_3
SELECT
  1: ATTR(=, severity), VALUE(=, HARMLESS);
MAP
  SV92 = "0892010002FE0300000000"
END

CLASS SV92_4
SELECT
  1: ATTR(=, $CLASSNAME);
MAP
  SV92 = "0892010002FE0300000000"
END

Again, this would require 4 different CDSs to produce one and only one of these 4 different event attributes.

To produce a single pseudo event that could have any combination of the above SV 05s and SV 92s using one pass through the CDS file would require 16 different CDS statements. The multiplication of the 4 statements needed to produce a unique SV05 and the 4 statements needed to produce a unique SV 92. Each of the 16 MAP segments has a single SV 05 and SV 92, representing all of the combinations that could occur. The four CDSs that represent both resources in combination with the various SV 92s are:

CLASS SVBOTH_1
SELECT
  1: ATTR(=, resource1);
  2: ATTR(=, resource2);
  3: ATTR(=, severity), VALUE(=, FATAL);
MAP
  SV05 = PRINTF("#{05#{1000#{<%s#>#}0084#{<%s#>#}0040#}#}", $V1, $V2);
  SV92 = "0892010001FE0300000000"
END

CLASS SVBOTH_2
SELECT
  1: ATTR(=, resource1);
  2: ATTR(=, resource2);
  3: ATTR(=, severity), VALUE(=, WARNING);
MAP
  SV05 = PRINTF("#{05#{1000#{<%s#>#}0084#{<%s#>#}0040#}#}", $V1, $V2);
  SV92 = "0892010011FE0300000000"
END

CLASS SVBOTH_3
SELECT
  1: ATTR(=, resource1);
  2: ATTR(=, resource2);
When other subvectors that need to be placed in the same output NMVT are added, the number of needed CDSs and the duplication of event attribute mappings in the MAP segment grows considerably.

**The Multiple-Pass Method**

To alleviate this problem, the event receiver makes multiple passes though the CDS file and collects separate mappings from each segment that it matches for the one pseudo event that is created. The $CDS\_GROUP$ keyword and the CONTINUE event attribute are used to control the multiple pass method.

Each pass starts at the beginning of the CDS file. If a CDS is matched that contains a valid CONTINUE event attribute, at least one more pass will be made through the CDS file. If a CDS is matched that does not have a CONTINUE statement, or no CDS is matched, that pass will be the last pass through the CDS file and all of the event attributes collected to this point are used to create the pseudo event.

**EVERY** CDS SELECT segment **MUST** have one statement that looks for the $CDS\_GROUP$ keyword to be equal to a string in the range of GROUP001–GROUP999. By default, the initial value of the $CDS\_GROUP$ keyword is GROUP001, so the first CDS statement matched must look for this keyword to be equal to GROUP001.

When a CDS is matched, the CONTINUE event attribute definition in the MAP segment of that CDS controls whether another pass will be made to match another CDS. The CONTINUE event attribute will cause the value of the $CDS\_GROUP$ keyword to change to a specific value (CONTINUE = GROUP004) or to the next numeric value (CONTINUE = NEXT). If a specific value is given, it must be greater than the current value of the $CDS\_GROUP$ keyword.

To illustrate the usage of the $CDS\_GROUP$ keyword and the CONTINUE event attribute, using the previous example, fill in the keyword and event attribute as follows:

```plaintext
CLASS SV05_1
SELECT
  1: ATTR(=,$CDS\_GROUP), VALUE(=,GROUP001);
  2: ATTR(=,resource1);
  3: ATTR(=,resource2);
MAP
  SV05 = PRINTF("#(05#(1000#(#<%s##>)#0084#(#<%s##>)#0040#)#)", $V1, $V2);
  SV92 = "0B92010012FE0300000000"
END

CLASS SV05_2
```

148  Tivoli NetView for z/OS Customization Guide
When a Tivoli Enterprise Console event arrives to be translated, the first subvector created is the SV 05 subvector. Because the initial value of the $CDS_GROUP keyword is GROUP001, the SELECT segments for all of the CDSs that create the SV 05 will look for this value. If none of the first three CDSs in this group are selected, the fourth will be selected by default. Because these CDSs define a
CONTINUE event attribute with a value of NEXT, the value of the $CDS_GROUP keyword will be updated to GROUP002, and another pass will be made through the CDSs to attempt to match on another CDS.

All of the SV 05 CDSs will now be ignored, because the $CDS_GROUP keyword is another value. Without this gate, the same SV 05 CDS would continue to be matched indefinitely. An SV 92 CDS will be matched next. The GROUP002 value for the $CDS_GROUP keyword determines this. Because none of the SV 92 CDSs have a CONTINUE event attribute, this will be the last pass made through the CDS file.

Using the previous CDSs, if an event arrives with event attributes, as follows:

```
resource1=FIRSTRES
resource2=SECNDRES
severity=WARNING
```

The following two subvectors will be produced:

```
1B0519100009C6C9D9E2E3D9C5E2008409E2C5C3D5C4D9C5E20040
0B92010011FE0300000000
```

**Building the NMVT**

When the pseudo event has been created, the NMVT will be built from data in the event attributes and keywords.

**Building the SV 31s Containing the Original Event**

The $BUILD_SV31LIST keyword indicates whether the SV 31s that contain the original Tivoli Enterprise Console event data will be built. These SV 31s are added to the NMVT first. The value of this keyword is modified by the contents of the BUILD_SV31LIST event attribute.

Each SV 31 contains an element of the original event: the class name, an event attribute/value pair, or the END designator. Formatted on an NPDA screen, a simple CDS example follows (assuming that the original event had a class name of SAMPLE):

```
ORIGINAL T/EC EVENT:
SAMPLE;
    resource1=FIRSTRES;
    resource2=SECNDRES;
    severity=WARNING;
END
```

**Overriding the SF21 Codepoint**

Each SV 31 contains an SF 21 subfield. By default, the codepoint associated with this subfield is X'00'. Two codepoints allow the SV 31 to be associated with the alert description and probable causes: codepoint X'21' to probable causes, and codepoint X'22' to alert description. By default, the SV 31 associated with a severity event attribute is assigned a X'21' codepoint, and the SV 31 associated with a msg event attribute is assigned a X'22' codepoint.

You can change which SV 31 is associated with the alert description or probable causes using the SF21 event attribute. This event attribute contains the name of an attribute from the input attribute list (which must be an event attribute value from the incoming Tivoli Enterprise Console event), followed by an equal (=) sign, followed by a one byte hexadecimal codepoint. For example, if you want to associate an event attribute called eventdetail from the incoming event with the alert description, code the following CDS:
The SF21_1 event attribute value follows:

eventdetail=21

When the SV 31 list is built, the data in the event attribute/value pair named by eventdetail will be associated with the alert description.

This SF 21 override only has an effect if the $BUILD_SV31LIST keyword indicates that the SV 31 list will be built; if the list is not to be built, this event attribute is ignored.

**Alert or Resolve**
The value of the $NMVT_TYPE keyword indicates whether the NMVT will be an alert NMVT (type 0000) or a resolve NMVT (type 0002). This keyword defaults to an alert NMVT. If the NMVT_TYPE event attribute is set within any matched CDS, the value of the $NMVT_TYPE keyword is set to this event attribute.

**Adding the User Subvectors**
After the SV 31s are added and the NMVT type is determined, the user subvectors created from CDS MAP segments are added to the NMVT. As previously mentioned, any event attribute can be assigned a value in the MAP segment of a CDS statement. The only event attributes that will be used to build user subvectors, however, must be prefixed with `SV`.

If the same event attribute name is used more than once, the value of the last one is used as the value of the event attribute. Therefore, if you need multiple subvectors of the same type, name the event attributes with this subvector data uniquely. Using `SV10` as the event attribute name for more than one SV 10 is not valid, because all preceding event attributes will be overwritten in the event attribute list. Use unique names such as `SV10_1`, `SV10_2`, and so forth.

The names for subvector event attributes do not necessarily correspond to the subvector. The value of an event attribute that you name as `SV10_1` can contain data for a completely different subvector. The value of the subvector event attribute determines the subvector type, not the name of the event attribute.

The value of a subvector event attribute is decoded as previously described. Subvectors are added to the NMVT in the order that their defining event attributes are encountered in the MAP segments.

**Calculating the AlertID for SV 92**
Because the alert ID field must be calculated for the subvector at the time that NMVT is built, the event receiver will calculate the value for this field of SV 92. However, you must specify an alert ID place holder in any SV 92 event attributes that you code in a CDS file. You can put any 4 bytes there; they will be overwritten by the event receiver. It is recommended that you code four bytes of zero (00000000) as the place holder.

The event receiver calculates the alert ID as described in *SNA Formats*. 

---

CLASS SF21_1
SELECT
  1: ATTR(*,$CDS_GROUP), VALUE(*,"GROUP001");
  2: ATTR(*,eventdetail);
MAP
  SF21_1 = PRINTF("%s=21",$N2);
END
An Example
The following example uses the default event receiver service CDS file (IHSAECDS) provided in the event/automation service.

Assume the following Tivoli Enterprise Console event has been received by the event receiver:

```
SNA_Performance_Degraded;source=NV390ALT;origin=B3088P2;
sub_origin=TX12/DEV;hostname=USIBMNT.NTVED;adapter_host=NMPIPL06;date=OCT 29 16:32:52;
severity=WARNING;msg=PERFORMANCE DEGRADED:CONTROLLER;adapter_host_snanode=USIBMNT.NTVED;
```

The previous event was an alert that was changed into an event by the alert adapter. All of the event attribute/value pairs are first coded into generic attributes for the input attribute list; the $CLASSNAME keyword attribute is assigned the value SNA_Performance_Degraded.

The first group in the CDS file is GROUP001; these CDSs determine the NMVT type. Because there is not a status event attribute in the incoming Tivoli Enterprise Console event, the NMVT_TYPE event attribute and the $NMVT_TYPE keyword are set to the value ALERT. Because CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP002.

The next group in the CDS file defines the SV 93. None of the information in the original event determines the value of the SV 93; the value of this subvector is as follows:

\[0493FE03\]

CONTINUE=NEXT is specified in the MAP segment. The $CDS_GROUP keyword is set to GROUP003.

The next group in the CDS file defines the SV 05. The example event will match on the class SV05_4, it has a host name, origin, and source event attribute, but not a probe event attribute. After PRINTF and translation, the value of this subvector follows:

\[2A052B1000EE42C9C2D05E34B5E3E5C5C408408C2F6F8DF200F50905E5F3F9F0C1D3E3004\]

CONTINUE=NEXT is specified in the MAP segment. The $CDS_GROUP keyword is set to GROUP004.

The next group in the CDS file defines the SV 10. None of the information in the original event determines the value of the SV 10; the value of this subvector follows:

\[1C1001911040506C7C5040908F5F6F7CF2BF3080F3E3C9E5D03C9\]

CONTINUE=NEXT is specified in the MAP segment. The $CDS_GROUP keyword is set to GROUP005.

The next group in the CDS file defines the SV 92. The example event will match on the class SV92_4, it has severity=WARNING and the $NMVT_TYPE is set to ALERT. The value of this subvector follows:

\[089201001FE0300000000\]
The alert ID portion of this subvector (the last 4 bytes) will be calculated and filled in by the event receiver. CONTINUE=NEXT is specified in the MAP segment. The $CDS_GROUP keyword is set to GROUP006.

The next group in the CDS file defines the SV 97. The example event will match on the class SV97_1, the $NMVT_TYPE is set to ALERT. The value of this subvector follows:

0A970881200035003000

CONTINUE=NEXT is specified in the MAP segment. The $CDS_GROUP keyword is set to GROUP007.

The next group in the CDS file defines an SF 21. The example event will match on the one and only CDS for this group, the $NMVT_TYPE is set to ALERT. The value of this subfield override follows:

msg=21

CONTINUE=NEXT is specified in the MAP segment. The $CDS_GROUP keyword is set to GROUP008.

The last group in the CDS file defines another SF 21. The example event will match on this last CDS, the severity event attribute is present in the event. The value of this subfield override follows:

severity=22

The $BUILD_SV31LIST keyword is still set to YES. The NMVT built from the previous process follows:

Translating ASCII Text Data

SNMP agents send up data (whether in variable bindings or other parts of the trap) that is essentially ASCII text data, but the data type in the encoding trap

Chapter 8. Customizing the Event/Automation Service 153
indicates an octet string. Since the data type is an octet string, the trap-to-alert data encoding process treats each byte of data as raw hexadecimal data rather than an encoded character. As a result, the parsing done by the trap-to-alert conversion task merely turns this data into a character representation of the hex data bytes for in SELECT criteria in the CDS file. For example, assume the character string ABC appears in a variable binding value with a type of octet string. Since the data is an octet string, the data is converted to the character string $414243$ and assigned to the generic keyword associated with the variable binding name.

If you want to use the original ASCII string value of the generic keyword in the outgoing alert, the ASCII string $414243$ needs to be converted back to the character string ABC and changed to EBCDIC. The $[$ and $]$ escape sequence has been provided to allow for conversion of the EBCDIC character string $414243$ back to the EBCDIC character string ABC.

Within the value encoding, inside the double quotes for the value of the subvector event attribute (whether in a PRINTF or not), this escape set is used to delimit data that is considered to be the character representation of hex data that, in turn, is ASCII character data. Data delimited in this way is turned into EBCDIC character data and placed within the value of the subvector event attribute. For example, if you had the following event attribute assignment in a Class Definition Statement:

$$SV05 = "0B0509100004#\[414243#\]0040"$$

The encoding of this event attribute value into an actual hexadecimal alert subvector would produce:

$$0B0509100004C1C2C30040$$

If data within the range delimited by the escape sequences turns out not to be character representations of hex data that are ASCII characters, then the conversion to EBCDIC will fail, and the translation of the trap (and thus, building of the alert/resolve) is terminated and the trap is discarded. Note that if other escape sequences occur following "#" and before "#]" is encountered, they are simply treated as characters that are put into the subvector, which would later fail conversion to hex then EBCDIC, because they aren’t character representations of hex digits. Also, if 
"[#" or "]#" occur following the "#<" escape sequence, which "turns off" translation of character representations of hex digits to hex data in the subvector, and before "]>", which "restores" that translation mode, then "]#" and "]#" are simply treated as untranslated character data, not escape sequences.

**Translating SNMP Non-String Data Types**

Some attributes used in CDS selection are assigned names based upon the places in the trap from which their values are extracted, while other names are adapted directly from the trap (for example, variable names, which are object identifiers, in the variable bindings). The encoded values are all string data, displayable forms of the data within the trap, and the formats of these strings depend upon the data types assigned to these pieces of data in the trap.

As an example, suppose that the data type of a value in the trap was found to be that of an internet address. The trap-to-alert conversion task would turn this into a string which was the internet address in dotted decimal notation. The following data types can be assigned to data in an SNMP trap, and the corresponding string to which it is translated.

**integer**

signed decimal number string. The integer 30 becomes the EBCDIC string "30"
null a pair of single quotes in EBCDIC. This becomes the EBCDIC string ""

octet string hexadecimal data string. The hex string 313233 becomes the EBCDIC string "313233".

object identifier ASN.1 data in dotted decimal notation format. The object 2C010306 becomes the EBCDIC string "1.4.1.3.6".

printable string an EBCDIC string

visible string an EBCDIC string

general string an EBCDIC string

internet address dotted decimal notation format. The address 09080706 becomes the EBCDIC string "9.8.7.6".

counter unsigned decimal number string. The number 05 becomes the EBCDIC string "5".

gauge unsigned decimal number string. The number 50 becomes the EBCDIC string "50".

ticks unsigned decimal number string. The number 132 becomes the EBCDIC string "132".

When the value is not of a data type listed above, then that value is treated as if it had a data type of octet string. Also, if the data type of the value in the binding is a complex structure like SEQUENCE OF (something that should not happen), then the value is treated as if it had the null data type.

The following example uses the default trap-to-alert service CDS file (IHSATCDS) supplied with the event/automation service. Assume the following trap data is received by the trap-to-alert conversion task (words separated for readability).

303B0201 00040670 75626C69 63A42E06 0C2B0601 14011203 01020101 03400449 B5203F02 01050201 00430100 300F300D 06082B06 01120108 07000201 30

Also assume that the IP address and port associated with the agent originating the trap is 9.50.20.8 and 161, respectively.

The trap data is first coded into corresponding keyword and generic attributes for the input attribute list. The encoded string attributes are:

$ORIGIN_ADDR 9.50.20.8
$ORIGIN_PORT 161
$SNMP_VERSION 0
community public
enterpriseOID 1.3.6.1.20.1.18.3.1.3.1.1.3
agent_address 73.181.32.63
generic_trap 5
specific_trap 0
timestamp 0
1.3.6.1.18.1.8.7.0 30
The first group in the CDS file is GROUP001; this CDS determines the NMVT type and BUILD_SV31LIST setting. Since this trap is not a Multi-System Manager trap, the generic formatting done by the CDS file IHSATALL is used. The NMVT_TYPE event attribute (and therefore, the $NMVT_TYPE keyword) is set to the value ALERT. The BUILD_SV31LIST event attribute (and therefore, the $BUILD_SV31LIST keyword) is set to the value YES. Since CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP002.

The next group in the CDS file defines the SV 92. The value of this subvector is:

0892080012F0000000000000

The Alert ID portion of this subvector (the last 4 bytes) will be calculated and filled in by the event receiver. CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP003.

The next group in the CDS file defines the SV 05. After PRINTF and translation, the value of this subvector is:

22050E100009F7F34BF1F8F14BF3008112110000F7F34BF1F8F14BF31F8F1F8F30081

CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP004.

The next group in the CDS file defines the SV 10. The value of this subvector is:

5A10000211030300002200EE261F3F9F0407E1998E1993983859340C595A3859997
9989A2550E28599A58992F111040804F0F1F0F3F0F0B06E899A596994940D5
85A359895A4D869940D6E261F3F9F00908F5F6F7C2F8F2

CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP005.

The next group in the CDS file defines another SV 10, which contains information about the resource reporting the trap. The value of this subvector is:

2C10000F11090300000091A4959929596A6951A111C0E02F0F0F0F0F0
F0F0F009064A959929596A695

CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP006.

The next group in the CDS file defines the SV 93 and SV 97. The value of these subvectors are:

0493FE000
A97040121004810000

CONTINUE=NEXT is specified in the MAP segment, the $CDS_GROUP keyword is set to GROUP007.

The last group in the CDS file defines the SV 98. The enterpriseOID, specific trap, and generic trap values are added as information in this subvector. The value of this subvector is:

severity=22

The $BUILD_SV31LIST keyword is still set to YES, the actual NMVT built from the previous process is:

0278000029310602028000000512C505E4032100193006D9C9C7C9D056D1C4C997EF94B
F9F74BF8F04BFF85E231060208000000512C505E403210013300609C7C9D056D07D6
D9E377EF1FF3F45E213060208000000512C505E4032100133006054D76E5C509E2C9
The trap-to-alert service post-CDS processing is nearly identical to that used by the event receiver post-CDS processing. The differences are:

- There is no $CLASSNAME keyword created by the trap-to-alert service since the incoming data was not a Tivoli Enterprise Console event.
- An additional escape sequence set $[ and $] is available to aid in translating variable binding data that are ASCII octet strings.
- Unlike Tivoli Enterprise Console event data, SNMP trap data can have a data type other than a character string.

Advanced Customization - Trap-to-Alert Forwarding Daemon

The way the event/automation service trap-to-alert conversion task receives traps is through a datagram socket which is bound to a port that you define in the configuration file (sample member name IHSATCFG). The conventional trap manager data port number, 162, is the default port.

Since port 162 is a "well-known" port for SNMP managers, and there may be multiple SNMP manager applications that are interested in trap data, this sort of port assignment can cause a conflict. To help resolve any conflicts, there is also a sample datagram forwarding daemon, IHSAUFWD, and an associated sample configuration file, IHSAUCFG, that are shipped with the event/automation service. The daemon receives data on a datagram socket and forwards that data to the destinations given in the configuration file.

Most SNMP agents are set to forward traps to the trap manager at port 162. IHSAUFWD can use this port to receive the trap data for all interested managers and then forward this data to the managers. These managers can be on the local system or at any IP address on the network.

The IHSAUFWD daemon uses a sample configuration file (IHSAUCFG) to specify the SNMP managers that are to receive the data. A description of the contents of this configuration file follows:

- **comments**
  Comments may be formed by beginning a line with the pound sign (number sign), "#", or the exclamation point, "!".

- **host internet address and port**
  The syntax for coding a destination for the datagram forwarding daemon, IHSAUFWD, and an associated sample configuration file, IHSAUCFG, that are shipped with the event/automation service. The daemon receives data on a datagram socket and forwards that data to the destinations given in the configuration file.
• white space (one or more blanks)
• port number, in decimal

An example of a lone coded like this would be:
137.45.110.2  6001

For more information on how to use and customize the forwarding daemon, refer to the comments in the IHSAUFWD sample.

### Alert-to-Trap Post-CDS Processing

The alert-to-trap service post CDS processing converts the Tivoli Enterprise Console event that is produced from the CDS process into an SNMP trap.

All non-variable binding information in the trap is put into the constructed trap by the alert-to-trap service directly, without the opportunity to customize it using the CDS file. The only exception to this is the specific trap value.

The alert-to-trap adapter sets the non-variable binding fields as follows:

- **version**: 0
- **community**: the value of the community statement from the alert-to-trap configuration file (IHSAATCF)
- **enterpriseOID**: the value of the enterpriseOID statement from the alert-to-trap configuration file
- **IP address**: the local host IP address
- **generic type**: 6
- **timestamp**: 0

The specific type is taken from the value of the specific event attribute that is created by the CDS processing.

All other event attribute/value pairs are encoded into variable bindings on the trap. The object id used for each of the variable bindings is the Enterprise OID from the alert-to-trap configuration file. The value of the binding has the form:

```
slot=value
```

where slot is the name of the event attribute from the CDS processing, and value is the value of that event attribute. The specific event attribute is not included in the variable bindings since its value is put in the specific type field of the trap.
Chapter 9. NetView Instrumentation

NetView instrumentation consists of subsystems. The topology display subsystem is available if you have NetView management console (NMC) or Tivoli Global Enterprise Manager (Tivoli GEM). For any other subsystem, including the event flow subsystem, Tivoli GEM must be installed.

Considerations

Instrumentation uses REXX CLISTs that invoke API functions. The API consists of compiled REXX. The REXX run time libraries are included in this release of NetView. However, if you are using a previous version, the REXX run time libraries must be included in the NetView steplib.

Events carrying management information to the topology server start as messages containing keyword/value pairs. These messages issued by the API are BNH351I, BNH352I, BNH353I, and BNH354I. These messages will be converted and forwarded to a topology server.

For earlier versions of NetView, instrumentation support (issuing and forwarding the BNH351I - BNH354I messages) requires the following APAR numbers to be available on your system:

- For NetView Version 2 Release 4 — OW30165
- For NetView Version 3 — OW31603
- For TME® 10 NetView for OS/390 Version 1 Release 1 — OW31604

Customization

The following samples were updated for application management instrumentation. You may need to customize them for your environment.

- **CNMSTYLE**
  
  Use CNMSTYLE to add automation table DSIAMIAT and autotask AUTOAMI.
  
  Also, uncomment the TOWER for AMI statement:
  
  ```
  TOWER = *SA *AON *MSM *Graphics MVScmdMgt NPDA +TARA NLDM +AMI
  ```

- **DSIAMIAT** - in sample DSIPARM
  
  A separate automation table for application management instrumentation. You need to uncomment one of the following includes:
  
  - `%INCLUDE DSIAMIR - to route the BNH351-BNH354 messages to another NetView. Use this for NetView Version 2 Release 4 and Version 3 Release 1.
  - `%INCLUDE DSIAMIT - to route the BNH351-BNH354 messages to a message adapter (the event automation service should be started). You may need to modify the PPI receiver ID of your event automation service message adapter (default is IHSATEC). The message adapter converts and sends the messages to the Tivoli Enterprise Console. The console rules format and send the converted messages to the appropriate topology servers.

  Configure the message adapter by including IHSAAPMF in the message adapter format file. Refer to the [Tivoli NetView for z/OS Installation: Configuring Additional Components](#) for more information.
Configure the Tivoli Enterprise Console by importing the files interapp.baroc and interapp.rls to your rules base and entering the address of your topology server in the file ihsttec.cfg. Refer to the Tivoli GEM Installation Guide for more information.

- `%INCLUDE DSIAMIN` - to route the BHN351-BNH354 messages directly to a topology server across NETCONV (this is the default)
- **DSIAMII** - in sample DSIPARM

Application management instrumentation member

- On the focal point NetView (the NetView that routes messages to the topology server or message adapter), code the NetView domain of all remote NetViews (if any) with the RMTLU=luname keyword.

- Customize the monitor default threshold specifications and polling intervals as appropriate for your environment. Note that the defaults defined here apply to all instances of a component or connection type. You can change threshold specifications and polling intervals for a specific instance by invoking the set threshold or set polling interval tasks.

You can define multiple threshold specifications. Each one consists of three values. The first value is the threshold value, the second value is the operator, the third value is the severity of the threshold event. For example:

```
BEGIN_THRESHOLD
  SS=Tivoli;TME10VCNMTAMEL;1.2
  MONITOR=('STATE'UP,6,0,DOWN,6,5 MVR=CNMETDMV 10)
  MONITOR=('IPC QUEUE' 25,8,2)
  MONITOR=('VIEWMGR QUEUE' 25,8,2)
  MONITOR=('VSTATMGR QUEUE' 25,8,2)
END_THRESHOLD
```

In the example, for the IPC QUEUE monitor, when the current value crosses above (operator 8) 25, a WARNING (2) threshold event is sent.

The meaning of each value is:

1. The threshold value against which the current monitor value is compared.
2. The comparison operator used to compare the current monitor value against the threshold value:
   - 0 = greater than
   - 1 = greater than or equal
   - 2 = less than
   - 3 = less than or equal
   - 4 = equal
   - 5 = not equal
   - 6 = changes to
   - 7 = changes from
   - 8 = crosses above
   - 9 = crosses below
   - 10 = matches
   - 11 = does not match
3. The severity of the threshold event to be sent if a match occurs follows:
   - 0 = "NORMAL"
   - 1 = "INFORMATIONAL"
   - 2 = "WARNING"
   - 3 = "SEVERE"
   - 4 = "CRITICAL"
   - 5 = "FATAL"
The following list details what you can customize in DSIAMII to activate one or all of the components.

- Hardware monitor component
  
  INIT=CNME3016(60)
  TERM=CNME3017()

  The parameter for CNME3016 is the heartbeat_interval.

- Event/automation service components (message adapter, alert adapter, event receiver)
  
  INIT=CNME9503(60 IHSAEVENT.IHSATEC)
  TERM=CNME9531()

  Change the INIT=CNME9503 statement to include the proname and PPI receiver ID of your adapters.

- MSM agent instrumentation
  
  INIT=FLCAPMIN(60)
  TERM=FLCAPMTR()

  The parameter for FLCAPMIN is the heartbeat_interval.

- Topology display subsystem components. These DSIAMII members have multiple statements for instrumentation initialization. The statements are as follows:
  
  INIT=CNMETDIN(HBEAT,60)
  INIT=CNMETDIN(QDEPTH,10)
  INIT=CNMETDIN(GMFHS,CNMSJH10.C)
  INIT=CNMETDIN(GPARM,DOMAIN=CNM01)
  INIT=CNMETDIN(RODM,EKGXRODM.X)
  INIT=CNMETDIN(COLDPARM,TYPE=COLD,INIT=EKLISLM)
  INIT=CNMETDIN(WARMPARM,TYPE=WARM)
  INIT=CNMETDIN(COMplete)

  The parameters are:
  
  - HBEAT specifies the heartbeat. It is required.
  - QDEPTH specifies the queue depth. It is required.
  - GMFHS specifies the GMFHS start-up procedure and its alias. It is required.
  - GPARM specifies the parameters to be used with the GMFHS start-up procedure. It is not required but if the domain value is not specified here, GMFHS will look to find the domain in the GMFHS initialization member DUIGNIT or in the specified GMFHS start-up procedure.
  - RODM specifies the RODM start-up procedure and its alias. It is required.
  - COLDPARM specifies the parameters for a RODM start-up procedure when a user chooses to do a RODM cold start. It is not required.
  - WARMMPARM specifies the parameters for a RODM start-up procedure when a user chooses to do a RODM warm start. It is not required.

If you create GEM instrumentation, you should modify DSIAMII to add default threshold specifications and calls to instrumentation initialization and termination routines. Refer to the Tivoli Global Enterprise Manager Instrumentation Guide for API descriptions.

Refer to the Tivoli Global Enterprise Manager CICSPlex SM Instrumentation User’s Guide for information about customizing DSIAMII.
Starting and Stopping Instrumentation

To start instrumentation, issue the INITAMI command on the focal point NetView (the NetView that routes messages to the message adapter). INITAMI is automatically issued on NetViews defined as remote in DSIAMII. The INITAMI command starts the AUTOAMI on the focal point NetView (if not already started). The console id for AUTOAMI is set to AMIxxxxx where xxxxx is the five rightmost characters of the NetView domain. Therefore, the console ID will be unique within a sysplex, and the commands issued from the autotask will correlate.

Instrumentation is not, however, forced to run on AUTOAMI. Therefore, in environments with multiple NetViews in a system, or in a sysplex, the INITAMI command should be issued on autotask AUTOAMI.

The INITAMI command also establishes a RMTCMD session with any NetView whose domain name is coded on the RMTLU statement in DSIAMII. This will log on the AUTOAMI autotask on that NetView.

To stop instrumentation, issue the TERMAMI command. TERMAMI is automatically issued on NetViews defined as remote in DSIAMII. In addition, stop the AUTOAMI autotask on the focal point NetView. This ends the RMTCMD sessions established by INITAMI.

The topology server may issue instrumentation related commands after issuing the TERMAMI command. However, the AUTOAMI autotask must be started for this to work.

Customizing the Tivoli Enterprise Console

If you route the instrumentation messages to the Tivoli Enterprise Console through the event automation service message adapter, you will need to customize the console.

ACB Monitor Customization

The application control block (ACB) Monitor focal point receives status updates for ACBs from the focal point Virtual Telecommunications Access Method (VTAM) and entry point VTAMs. If used in conjunction with Tivoli Business System Manager (TBSM), the ACB Monitor discovers the following:

- generic resources
- user-specified applications,
- applications matching user-specified models

The ACB Monitor also monitors the following:

- ACB status
- session count
- persistent recovery events for ACB applications

If used in conjunction with TBSM or with NMC TN3270 management, the ACB Monitor discovers TN3270 servers and clients. Optionally, ACB data can be saved in a DB2 database.
Define one ACB Monitor focal point for each System complex (or sysplex, the set of MVS and or z/OS systems). To fully enable instrumentation of application dynamics in a sysplex environment, define all other images in the sysplex to be entry points of that focal point.

Saving ACB data in DB2 enables you to query telnet clients by IP address, host name, or application name (using the Locate TN3270 Client TBSM tasks). This also enables you to change your list of critical TN3270 client resources without restarting the ACB Monitor.

Notes:
1. To save ACB data to DB2, DB2 must be operational on the ACB Monitor focal point, and the NetView SQL pipe stage must be enabled.
2. The AMI must be enabled on the ACB Monitor focal point to enable the ACB Monitor instrumentation.

Parts

The parts that are shipped as part of the ACB Monitor are listed in Table 19.

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Language</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TN3270.BSDF</td>
<td>MIF</td>
<td>TN3270 business system description file</td>
</tr>
<tr>
<td>TN3270.BCDF</td>
<td>MIF</td>
<td>TN3270 business component description file</td>
</tr>
<tr>
<td>TN3270.BMDF</td>
<td>MIF</td>
<td>TN3270 business mapping description file</td>
</tr>
<tr>
<td>TN3270.CDF</td>
<td>MIF</td>
<td>TN3270 component definition file</td>
</tr>
<tr>
<td>Ltn3270loc.ddf</td>
<td>DDF</td>
<td>Locate TN3270 client local dialog definition</td>
</tr>
<tr>
<td>Ltn3270glob.ddf</td>
<td>DDF</td>
<td>Locate TN3270 client global dialog definition</td>
</tr>
<tr>
<td>TN3270.html</td>
<td>HTML</td>
<td>Help file</td>
</tr>
<tr>
<td>GENRSC.BSDF</td>
<td>MIF</td>
<td>Generic Resources business system description file</td>
</tr>
<tr>
<td>GENRSC.BCDF</td>
<td>MIF</td>
<td>Generic Resources business component description file</td>
</tr>
<tr>
<td>GENRSC.BMDF</td>
<td>MIF</td>
<td>Generic Resources business mapping description file</td>
</tr>
<tr>
<td>GENRSC.CDF</td>
<td>MIF</td>
<td>Generic Resources component definition file</td>
</tr>
<tr>
<td>GENRSC.html</td>
<td>HTML</td>
<td>Help file</td>
</tr>
<tr>
<td>VTAMAPPL.BSDF</td>
<td>MIF</td>
<td>VTAM Application business system description file</td>
</tr>
<tr>
<td>VTAMAPPL.BCDF</td>
<td>MIF</td>
<td>VTAM Application business component description file</td>
</tr>
<tr>
<td>VTAMAPPL.BMDF</td>
<td>MIF</td>
<td>VTAM Application business mapping description file</td>
</tr>
<tr>
<td>VTAMAPPL.CDF</td>
<td>MIF</td>
<td>VTAM Application component definition file</td>
</tr>
<tr>
<td>VTAMAPPL.html</td>
<td>HTML</td>
<td>Help file</td>
</tr>
</tbody>
</table>

Defining a Focal Point

To define an ACB Monitor focal point, perform the following steps:
1. Customize the automation table in sample DSIAMIAT. Uncomment the following include: %INCLUDE CNMSVTFT.
2. Customize the AMI configuration member in sample DSIAMII using the following steps:
   a. Code the NetView domain name of each ACB Monitor entry point on AMONLU=keyword.
   b. Do you want to save ACB data to DB2?
      • If yes, perform steps 2c and 2d.
      • If no, go to step 2e.
   c. Code AMONDB2=Y.
   d. Code the DB2 volume on DB2VOL=keyword.
   e. Code the DB2 volume catalog on DB2VCAT=keyword.
   f. Code the DB2 buffer pool on DB2BUFFERPOOL=keyword for each predefined VTAM Application to be monitored. An icon will be added to the GEM VTAM Applications view for each Application.
3. Customize the list of VTAM applications and models to be discovered in sample DSIAMII as follows:
   a. Code the application name on APPLCOMPONENT=applname for each predefined VTAM application to be monitored. An icon will be added to the TBSM VTAM applications.
   b. Code the model name on MODELCOMPONENT=modelname for each VTAM model to be monitored. An icon will be added to the TBSM VTAM Applications view for each Application cloned from that model name.
4. Do you want to save ACB data to DB2?
   • If no, go to step 6.
   • If yes, customize the DB2 parameters in sample DSIAMII by completing the following steps:
      a. Code AMONDB2=Y.
      b. Code the DB2 volume on DB2VOL=keyword.
      c. Code the DB2 volume catalog on DB2VCAT=keyword.
      d. Code the DB2 buffer pool on DB2BUFFERPOOL=keyword.
5. Customize the default thresholds in sample DSIAMII. You can customize any of the following:
   • when threshold events are issued for the ACB status monitor
   • the severity of the events issued for the ACB status monitor
   • the session count monitor
   • the persistent recovery monitor

   Customization in DSIAMII defines default thresholds. You can also customize thresholds for each instance (icon) with the set threshold task.

   For example, if you want to change the threshold severity of CONCT and RESET states to SEVERE (3) rather than INFORMATIONAL (1) for APPLCOMPONENT and MODELCOMPONENT Applications, change the following line:
   \texttt{ACT,6,0,CONCT,6,1,RESET,6,1,INACT,6,2,UNKNOWN,6,2,PINACT,6,4,PACT,6,4}

   To:
   \texttt{ACT,6,0,CONCT,6,3,RESET,6,3,INACT,6,2,UNKNOWN,6,2,PINACT,6,4,PACT,6,4}

   Or, if you want a WARNING threshold event to be issued when session counts exceed 999, and a NORMAL threshold event when session counts fall below 1000, change the following line:
To:

MONITOR=('SESSION COUNT' 999,8,2,1000,9,0 EVENT)

6. Install the ACB Monitor VTAM exit. Link edit CSECT CNMIETMN into load module ISTIETMN in the VTAMLIB DD for VTAM.

**Defining An Entry Point**

To define an ACB Monitor entry point, perform the following steps.

**Step 1.** Customize the automation table in sample DSIAMIAT. Uncomment the following include:

```
%INCLUDE CNMSVTET
```

**Step 2.** Install the ACB Monitor VTAM exit. Linked CSECT CNMIETMN into load module ISTIETMN in the VTAMLIB DD of VTAM.

**Starting the VTAM ACB Monitor**

Start the AMI by issuing the `INITAMI` command on the focal point NetView to enable instrumentation for:

- generic resource
- TN3270 servers
- APPLCOMPONENT VTAM applications
- MODELCOMPONENT VTAM applications

To start the VTAM ACB Monitor, issue the `INITAMON` command on the focal point NetView. The focal point and all entry points identified on the `AMONLU=keyword` will be activated.

After the VTAM ACB Monitor has been activated, issue the `INITAMON entry_point` command, to activate an additional entry point, where `entry_point` is the NetView domain name of the entry point.

**Recovering a VTAM ACB Monitor Entry Point**

When the RMTCMD LU 6.2 session between an entry point and the focal point fails, the entry point is automatically stopped. When the error that caused the communication failure between the two NetViews has been corrected, issue the `INITAMON entry_point` command on the focal point to recover the entry point.

**Stopping the VTAM ACB Monitor**

To stop the VTAM ACB Monitor, issue the `TERMAMON` command on the focal point NetView. The focal point and all active entry points will be deactivated. All GEM components will be removed for:

- generic resource
- TN3270 instrumentation
- APPLCOMPONENT instrumentation
- MODELCOMPONENT instrumentation.

To stop a specific entry point, issue the `TERMAMON entry_point` command, where `entry_point` is the NetView domain name of the entry point. Status for all of the applications on the VTAM associated with that NetView will be removed from the database, and all GEM components which represent resources on that VTAM will be removed for:

- generic resource
• TN3270 instrumentation
• MODELCOMPONENT instrumentation.
IBM eNetwork Host-On-Demand provides a host access class library to enable users to write Java™ applications to automate NetView 3270 management console (NMC-3270) sessions. These automation applications can be used to interact with the NMC-3270-provided sessions for routine tasks. The automation applications can also be used from GEM or the NMC.

Providing a Host Access class library application for the NMC-3270 involves the following steps:

- Writing the application
- Launching the application either at NMC-3270 startup or from a dialog.

Refer to the IBM eNetwork Host Access Class for more information. Information is also available from the NetView 3270 management console menu bar (select Books).

### Writing a NetView 3270 Management Console Host Access Class Library Application

The Host Access Class Library (HACL) classes associated with a particular NetView session can be obtained from an instance of ECLSession.

To do this, the application must implement FLB_NVAppInterface. This interface provides the methods for passing the active session to the application which is done through the init method of the interface.

After getting a session object in the init method, you can gain access to the presentation space and interact with it. The presentation space is encapsulated in the ECLPS class, and an instance of it can be obtained using the GetPS() method on ECLSession. ECLPS provides methods that:

- Manipulate text
- Perform searches
- Send keystrokes to the host
- Work with the cursor.

The following sample gets an instance of ECLPS from the session described above.

```java
public void init(ECLSession session)
{
    ps = session.GetPS();
    oia = session.GetOIA();
}
```

When an instance of ECLPS is established, you can register as a com.ibm.eNetwork.ECL.event.ECLPSListener to receive notification of presentation space changes. Registered listeners are notified when the presentation space is changed. This event notification model is the primary mechanism used by an application to drive interactions with the presentation space.
The sample code below registers the current class with the instance of ECLPS.

```java
try {
    ps.RegisterPSEvent(this);
} catch(ECLErr e) { System.out.println(e.GetMsgText()); }
```

Even though you are registered for presentation space events, you still need to implement `com.ibm.eNetwork.ECL.event.ECLPSListener` interface.

The ECLPSListener interface is comprised of three methods which handle different kinds of events occurring within the presentation space. The PSNotifyEvent() method handles normal, non-error events and is the main method for receiving and handling events. The PSNotifyStop() method handles stop events, and the PSNotifyError() method handles errors which occur during event generation.

The following sample defines a PSNotifyEvent() method which prints out screen updates.

```java
public void PSNotifyEvent(ECLPSEvent evt)
{
    try
    {
        char[] temp = new char[1921]; // Screen size is assumed to be 24x80
        ps.GetScreen(temp, 1920, 1, 1920, ps.TEXT_PLANE);
        System.out.println(new String(temp));
        ps.UnregisterPSEvent(this);
    }
    catch (Exception ECLErr)
    {
        System.out.println("ECLErr Exception --> " + ECLErr.toString());
    }
}
```

FLB_NVApplInterface also requires you to implement three other methods (addListener, removeListener, and closeDown) which allow the application to listen for a closeDown event from the NMC-3270. The closeDown event is triggered when the NMC-3270 is ready to close the session. The closeDown method allows the user to clean up before the session ends. The removeListener method stops the application from listening for closeDown events from the NMC-3270.

The following is a sample implementation:

```java
public void addListener( FLB_NVApplListener listener ){
    this.listener = listener ;
}
public void closeDown(){
    //Your clean up code goes here.
}
```

```java
public void removeListener( FLB_NVApplListener listener ){
    this.listener = null;
}
```

Finally, to notify the NMC-3270 of application termination you must code following in your termination or finalize routine:

```java
listener.applClosing(this);
```
Building Host Access Class Library Applications

This section describes how to build a Java application which uses the Host Access Class Library (HACL). The source code preparation and compiling requirements are described.

Source Code Preparation

Programs that use HACL classes must import the HACL package to obtain class definitions and other compile-time information. The HACL package can be imported into a Java source file using the following statement:

```java
import com.ibm.eNetwork.ECL.*;
import com.ibm.eNetwork.ECL.event.*;
import com.ibm.eNetwork.beans.HOD.*;
import src.ibmflb.*;
```

Compilation

To compile the new Java source file, the CLASSPATH must be updated to include the directory containing the HACL package. If HACL was installed in a Windows® environment, the CLASSPATH should already be updated. If HACL was not installed in a Windows environment, you will have to either update the CLASSPATH environment variable manually or use the ‘-classpath’ parameter of the Java compiler, javac, to specify the location of HACL.

Running the HACL Application

The NetView 3270 management console provides two ways to load and run a user-defined application. A user application can be launched from the Execute HACL App dialog or as an Initial HACL App when a session is started. The Run Application dialog can be displayed from the toolbar. The dialog prompts for the name of a user-defined class (fully-qualified class name), constructs an instance of the class using the default constructor, and gives the class access to the current NetView session in the init method. If you do not know the class name you can use the find button to locate the class. In this instance the class will be freshly loaded. The Initial HACL App can be specified in the session configuration window.

Note: When an application that uses HACL is run, the directory containing the HACL package must be found in the path specified by the CLASSPATH environment variable.

Helper Class

The NetView 3270 management console provides a helper class to help an application writer get ECLSessions for various console sessions (for example, the hardware monitor and session monitor).

The following is a sample:

```java
ECLSession session = FLB_HACLhelper.getSession("NPDA");
ECLPS ps = session.GetPS();
ECLOIA oia = session.GetOIA();
```
Sample Applications

The following two sample applications are shipped with the NetView 3270 management console (in the examples subdirectory):

- FLB_HACLsample.java
  This sample enters the command ald in the input area of the screen.
- FLB_ScreenSearchSample.java
  This sample searches for the word alert and responds with an ALD command.
Chapter 11. Designing HTML Files for the NetView Web Server

NetView provides a full-function Web application server that accepts commands through a Web browser interface, displays TCP/IP trace and SNMP data, and includes full online help. You can design HTML files for your own Web page.

Referencing Files and Commands

Most of the HTML for accessing NetView from the Web browser is dynamically generated by the NetView program and sent to the Web application server. Static elements such as style sheets are installed with the servlet code of the Web application server. This eliminates the need for network connections between the Web application server and the NetView host to retrieve frequently-accessed elements of the HTML.

The HTML (including user-written HTML) may be divided between the Web application server (workstation) and the NetView for z/OS host. This can include the following situations:

- Referencing host files from host HTML
- Referencing workstation files from host HTML
- Referencing host files from workstation HTML.

Understanding the Base URL

The following is a typical URL when browsing the NetView program:

http://httphost.example.com/netview/cnm01/

Therefore, the above is considered to be the base URL. If the URL contained a question mark, any remaining data is considered query data and is not considered to be part of the base URL.

Referencing Workstation Files on the Web Application Server

References to other sources should be relative to the base URL. For files on the Web application server, use ../ to back up to the /netview/ directory.

Referencing NetView Host Files

When you design your own HTML files, use the following format:

DD name

DD/ddname/membername&DSIWBSESSION.

DSN name

DSN/hq/mq/lq/membername&DSIWBSESSION.

Where:

hq High-level data set qualifier name
mq Middle-level data set qualifier name
lq Low-level data set qualifier name

NetView supports the file type in the DD and DSN name. For these files, add .type to the end of the file name, for example:

DD/ddname/membername.gif
Typically, the base URL is as follows:
http://httphost.example.com/netview/cnm01/

Therefore, the DD and DSN names are appended to the base URL. For example, the complete URL becomes as follows:
http://httphost.example.com/netview/cnm01/DD/DSIPARM/userhtm

The supported file types are:

<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ext</td>
<td>File type</td>
</tr>
<tr>
<td>.gif</td>
<td>GIF image</td>
</tr>
<tr>
<td>.jpeg</td>
<td>JPEG image</td>
</tr>
<tr>
<td>.jpg</td>
<td>JPEG image</td>
</tr>
<tr>
<td>.class</td>
<td>applet, element, or object</td>
</tr>
<tr>
<td>.mpeg</td>
<td>MPEG video</td>
</tr>
<tr>
<td>.mpe</td>
<td>MPEG video</td>
</tr>
<tr>
<td>.mpg</td>
<td>MPEG video</td>
</tr>
<tr>
<td>.js</td>
<td>Java script</td>
</tr>
</tbody>
</table>

For more examples, refer to FKXWPORF. Also, refer to FKXWB000 for examples of the frameset, help mapping, and title frame rewriter script.

Referencing NetView Commands

&DSIWBSESSION.?DSICMDS+=+command

Specifies the NetView command to be issued. Any blanks in the command must be specified as a plus sign (+) so that the command will be correctly parsed. The NetView Web server (DSIWBTSK) changes the plus signs to blanks before issuing the command. Refer to CNME2011 as an example.

Adding Links to the Portfolio

You can customize the NetView-supplied HTML file FKXWPORF to add links to your pages.

Note: If you change this NetView-supplied HTML file, use a %INCLUDE statement and place any new information in a separate file. Use the %INCLUDE statement for ease in migrating from release to release.

To add links to the portfolio, add a new section (for example, TCP/IP Support) to FKXWPORF similar to the following example:

```html
<TR>
<TD nowrap COLSPAN=2>
<DIV class="weight">
<B>TCP/IP Support</B>
</DIV>
</TD>
</TR>
```

To add a new link to display in the main frame, insert the following:

```html
<TR>
<TD nowrap>&tdsp;&tdsp;&tdsp;&tdsp;</TD>
</TR>
```
Loading Panels

To load a panel into the main frame of the Web browser, specify
\texttt{target="mainFrame"} in the link. You must also change the title of the frame by
adding the following:
\begin{verbatim}
<body ONLOAD="top.newtitle('New Title of Panel')">
\end{verbatim}

If you want the HTML to run both inside and outside the frameset, code the
following instead of as in the above example:
\begin{verbatim}
<script language="JavaScript">
  <!--
  if (top.window.name != window.name)
    top.newtitle('New Title of Panel');
  //-->
</script>
\end{verbatim}

If you chose to place the added link inside the frameset, perform the following
step if you want to add context-sensitive help. In the FXXWB000 file, add a case
statement similar to the following for your help file:
\begin{verbatim}
case "Title of Your Panel": curhelp="yourhelp.html"
  bottomFrame.location.reload(true)
  break
\end{verbatim}

The case statement above is based on the exact text of your title. While the .html
help file is located on the server, use other FXX* files to serve as examples of
how to code the help file. Place your new file in the directory where the FXX* files
are located and then reference the file in the copied case statement above.

History Log

Each time a command is issued, the command and the corresponding response can
be placed in the history log. To place the information in the log, you must invoke
FKXEWLOG. Before invoking FKXEWLOG, create a named safe called
\texttt{MYWEBLOG}.

In \texttt{MYWEBLOG}, the following conventions are required:
\begin{itemize}
  \item Place the command issued and the results into \texttt{MYWEBLOG}, separated by a
        blank line. This will cause the provided information to be formatted similar to
        the output of the TCP/IP PING command.
  \item If no formatting is required or if preformatting (non-HTML formatting) has
        already occurred, then set the first line of \texttt{MYWEBLOG} to \texttt{<PRE>}. Named safes
        containing \texttt{<PRE>} will not be required to contain a blank line because all
        formatting must be provided by the calling function. HTML tags are not
        recommended within the \texttt{<PRE>} tags.
\end{itemize}
Using REXX to Generate HTML

NetView supports an interface similar to the Common Gateway Interface (CGI) for REXX procedures. Use the REXX function CGI() to determine whether your procedure was invoked by the NetView Web server. If CGI() returns 1, the procedure can create a dynamic Web page by ensuring that the beginning characters of the first line of output are either:

- `<HTML`
- `<!DOCTYPE`

Note: HTML and DOCTYPE must be in uppercase.

In this case, the NetView program does not modify or add to the output. It is recommended that you create output using the pipe stage CONSOLE ONLY to prevent the logging and automation of the HTML output. Refer to REXX part FKXEIPTR or HTML part FKXWI100 as examples for writing an HTML page.

Note: The CGI function is the preferred method to provide customization.

To improve performance, you can place static HTML or binary files on the Web application server.

Security Token

Use the security token &DSIWBSSESSION to track the browser session and perform security. This token is passed between the browser and the Web application server.

The security token will be substituted by the NetView program with the actual security token that is in use between the current browser and the Web application server. This token should only be used in HTML that is generated in, or that resides in, the NetView program. This token should not be coded in static HTML that is installed at the Web application server.

The security token is only used when URL rewriting is used for session tracking. Refer to the Tivoli NetView for z/OS Security Reference for more information about session tracking.

If you only use cookies for session tracking in your environment, then this token is not required. When cookies are used and this token is coded, then the token will be removed by the NetView program. If any web browsers have cookies disabled, then the token is required to be coded, or web pages will not be displayed properly to users who have cookies disabled.

Migrating Web Browser Support

In NetView V1R3 and V1R4, in order for the NetView program to track the browser session and perform security, the token &DSIWBSSESIDF was required in any HTML sent to the browser from NetView. For example:

```
DD&D$IWBSSESIDF /ddname/memname
```

This token is no longer required because NetView no longer tracks the browser sessions. Tracking of browser sessions has been moved to the Web application server. If the NetView program encounters this token, it will be removed and the HTML will be treated as if the token did not exist.
A new security token is now used; however, this new token is not an exact replacement for the old token. The new token is &DSIWBSESSION, and is only required if you are using URL rewriting for session tracking between the browser and the Web application server. The new token is placed in different locations, depending on the reference type. See the following examples.

The following example is a command reference on the host:

```
<A target="mainFrame" href="&DSIWBSESSION.?DSICMDS++%25SP.FKXEDVST">DVIPA Status</A>
```

The following example is a reference to a server-side application:

```
<A target="mainFrame" href="/snmpcliserlet/&DSIWBSESSION.?znvDomain=NTV74">SNMP Commands</A>
```

Both of these references are based off the base URL.

*For more information about this new token and URL rewriting, refer to the [Tivoli NetView for z/OS Security Reference](#).* Refer to the FKXWPORF file for examples of opening full screen windows outside of the frameset.

**Note:** If you are using session cookies or referencing server files (for example, HTML or CSS file types), &DSIWBSESSION. is not required.

When referencing the source of a command on host NetView, the security token is listed first. For example:

```
<...src="&DSIWBSESSION.?DSICMDS++%25SP.FKXEIGEN+FKXWPORF">
```

When referencing a panel on host NetView, the security token is listed last. For example:

```
<...src="DD/CNMPNL1/FKXWELCM&DSIWBSESSION.">
```

Submissions of forms also require the security token. The `<FORM>` attribute must be coded with the following action keyword:

```
<Form METHOD="POST" action="&DSIWBSESSION." NAME="fxxwcust" class="monotypeFont">
```

Form data is sent to the host through the server as a set of name-type pairs. For example, a pair of radio buttons would be coded as follows:

```
<form METHOD="POST" action="&DSIWBSESSION." NAME="FKXWMYCD">
  <input TYPE=HIDDEN VALUE="FKXETEST" NAME="COMMAND">
  <input TYPE=HIDDEN VALUE="FKXETEST" NAME="COMMAND">
  <input TYPE=RADIO SELECTED NAME="HAPPY" VALUE="NO">
  Yes:
  No:<input TYPE=RADIO NAME="HAPPY" VALUE="YES">
  <input TYPE="SUBMIT" value="Yes">
</form>
```

If the form containing the above statements is submitted to the host, the command FXXETEST would be invoked. In this example, FXXETEST is coded in REXX and would execute the following REXX PIPE:

```
'PIPE (NAME GRABPARM)',
'SAFE *',
'SEPARATE',
'EDIT /=* 1 1 * N',
'VARLOAD'
```

In this example, the REXX local variable HAPPY will be set with a value of either YES (the default) or NO, depending on if or which radio button was selected.
Appendix A. Color Maps for Hardware Monitor Panels

Table 20 lists the panel name, panel number, and color map for hardware monitor panels. See "Chapter 6. Customizing Hardware Monitor Displayed Data" on page 75 for more information on color maps.

Note: Color maps for hardware monitor help and command description panels are available only in prior releases of NetView. Also, color maps beginning with BNJMP1 are no longer supported.

Table 20. Color Maps for Hardware Monitor Panels

<table>
<thead>
<tr>
<th>Panel Name</th>
<th>Panel Number</th>
<th>Color Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerts-Dynamic</td>
<td>NPDA-30A</td>
<td>BNJMP30A</td>
</tr>
<tr>
<td>Alerts-History</td>
<td>NPDA-31A</td>
<td>BNJMP31A</td>
</tr>
<tr>
<td>Alerts-Static</td>
<td>NPDA-30B</td>
<td>BNJMP30A</td>
</tr>
<tr>
<td>Common Format Glossary</td>
<td>NPDA-02C</td>
<td>BNJMP2C1</td>
</tr>
<tr>
<td>Controller Information Display</td>
<td>NPDA-02E</td>
<td>BNJMP02E</td>
</tr>
<tr>
<td>Controller (CTRL) Selection Menu</td>
<td>NPDA-CTRL</td>
<td>BNJMPCTL</td>
</tr>
<tr>
<td>Downstream Member of Token-Ring LAN Fault Domain</td>
<td>NPDA-44B</td>
<td>BNJMP4BH</td>
</tr>
<tr>
<td>DSU/CSU and Line Status DSU/CSU and Line Parameters Link Segment Level n</td>
<td>NPDA-22C, page 1</td>
<td>BNJMPDL1</td>
</tr>
<tr>
<td>DSU/CSU and Line Status Remote DSU/CSU Interface-Remote Device Status-Link Segment Level n</td>
<td>NPDA-22C, page 2</td>
<td>BNJMPDL2</td>
</tr>
<tr>
<td>DSU/CSU and Line Status Configuration Summary, Link Segment Level n</td>
<td>NPDA-22C, page 3</td>
<td>BNJMPDL3</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43B</td>
<td>BNJMP43B</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43M</td>
<td>BNJMP43M</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43N, 43Q</td>
<td>BNJMP43N</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43C</td>
<td>BNJMP43C</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43T</td>
<td>BNJMP43T</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43A</td>
<td>BNJMP43A</td>
</tr>
<tr>
<td>Event Detail</td>
<td>NPDA-43P NPDA-43S</td>
<td>BNJMP43P BNJMP43S</td>
</tr>
<tr>
<td>Event Detail, alternate</td>
<td>NPDA-43T NPDA-43S</td>
<td>BNJMP434 BNJMP433</td>
</tr>
<tr>
<td>Event Detail for BSC Line</td>
<td>NPDA-43T NPDA-43T</td>
<td>BNJMP43T BNJMP43T</td>
</tr>
<tr>
<td>Event Detail for BSC Station</td>
<td>NPDA-43B NPDA-43B</td>
<td>BNJMP43B BNJMP43B</td>
</tr>
<tr>
<td>Event Detail for BSC/SS Line</td>
<td>NPDA-43B</td>
<td>BNJMP43B</td>
</tr>
<tr>
<td>Event Detail for BSC/SS Station</td>
<td>NPDA-43B</td>
<td>BNJMP43B</td>
</tr>
<tr>
<td>Event Detail for Channel-Attached Station</td>
<td>NPDA-43B NPDA-43J</td>
<td>BNJMP43B BNJMP43J</td>
</tr>
<tr>
<td>Event Detail for Channel Link</td>
<td>NPDA-43B</td>
<td>BNJMP43B</td>
</tr>
<tr>
<td>Event Detail for Instruction Exception</td>
<td>NPDA-43K</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Miscellaneous Interrupts</td>
<td>NPDA-43G</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Scanner-Type 1/4</td>
<td>NPDA-43H</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Scanner-Type 2/3</td>
<td>NPDA-43I</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Scanner-Type 1</td>
<td>NPDA-43D</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Scanner-Type 2</td>
<td>NPDA-43E NPDA-43F</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Scanner-Type 3</td>
<td>NPDA-43I NPDA-43P</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for Scanner-Type 4</td>
<td>NPDA-43D</td>
<td>BNJMP43D</td>
</tr>
<tr>
<td>Event Detail for SDLC Line</td>
<td>NPDA-43B</td>
<td>BNJMP43B</td>
</tr>
</tbody>
</table>
### Color Maps

Table 20. Color Maps for Hardware Monitor Panels (continued)

<table>
<thead>
<tr>
<th>Panel Name</th>
<th>Panel Number</th>
<th>Color Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Detail for SDLC Line</td>
<td>NPDA-43T NPDA-43B</td>
<td>BNJMP43T BNJMP43B</td>
</tr>
<tr>
<td>Event Detail for SDLC Station</td>
<td>NPDA-43R NPDA-43L</td>
<td>BNJMP43T BNJMP43L</td>
</tr>
<tr>
<td>Event Detail for SDLC Station</td>
<td>NPDA-43R</td>
<td>BNJMP43R</td>
</tr>
<tr>
<td>Event Detail for 3270 Non-SNA Controller</td>
<td>NPDA-43R</td>
<td>BNJMP43R</td>
</tr>
<tr>
<td>Event Detail Menu Event Detail Menu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event Detail Menu, alternate</td>
<td>NPDA-43R NPDA-43T</td>
<td>BNJMP434 BNJMP43R</td>
</tr>
<tr>
<td>Event Detail Menu for BSC Line</td>
<td>NPDA-43R NPDA-43T</td>
<td>BNJMP434 BNJMP43R</td>
</tr>
<tr>
<td>Event Detail Menu for BSC Line, alternate</td>
<td>NPDA-43R NPDA-43T</td>
<td>BNJMP434</td>
</tr>
<tr>
<td>Event Detail Menu for SDLC Line Event</td>
<td>NPDA-43R NPDA-43T</td>
<td>BNJMP434</td>
</tr>
<tr>
<td>Event Detail Menu for SDLC Line, alternate</td>
<td>NPDA-43R NPDA-43T</td>
<td>BNJMP434</td>
</tr>
<tr>
<td>Event Detail Menu for SDLC Station</td>
<td>NPDA-42A</td>
<td>BNJMP42A</td>
</tr>
<tr>
<td>Event Summary</td>
<td>NPDA-42B</td>
<td>BNJMP42B</td>
</tr>
<tr>
<td>Event Summary</td>
<td>NPDA-42C (many displays) NPDA-44C</td>
<td>BNJMPGLO</td>
</tr>
<tr>
<td>Glossary displays</td>
<td>NPDA-02B</td>
<td>BNJMP44C</td>
</tr>
<tr>
<td>HELP Menu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hexadecimal Display of Error Record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line Analysis-Link Segment Level ( n )</td>
<td>NPDA-24B</td>
<td>BNJMP4LNA</td>
</tr>
<tr>
<td>Link Configuration</td>
<td>NPDA-44A1</td>
<td>BNJMP441</td>
</tr>
<tr>
<td>Link Configuration</td>
<td>NPDA-44A2</td>
<td>BNJMP442</td>
</tr>
<tr>
<td>Link Configuration, alternate</td>
<td>NPDA-44A1</td>
<td>BNJMP443</td>
</tr>
<tr>
<td>Link Configuration Summary-Level Selection</td>
<td>NPDA-LSLS</td>
<td>BNJMP4LSL</td>
</tr>
<tr>
<td>Link Data for SNA Controller</td>
<td>NPDA-23A</td>
<td>BNJMP23A</td>
</tr>
<tr>
<td>Link Problem Determination Aid (LPDA-1) Data</td>
<td>NPDA-52A</td>
<td>BNJMP52A</td>
</tr>
<tr>
<td>Link Problem Determination Aid (LPDA-1) LDM Data</td>
<td>NPDA-52AL</td>
<td>BNJMP52L</td>
</tr>
<tr>
<td>(LPDA-2) Data Link Segment Level ( 1 )</td>
<td>NPDA-52B</td>
<td>BNJMP52B</td>
</tr>
<tr>
<td>(LPDA-2) Data Link Segment Level ( 1 ), alternate</td>
<td>NPDA-52B</td>
<td>BNJMP522</td>
</tr>
<tr>
<td>(LPDA) Data Link Segment Level 2</td>
<td>NPDA-52C</td>
<td>BNJMP52B</td>
</tr>
<tr>
<td>Link Status and Test Results</td>
<td>NPDA-24A</td>
<td>BNJMP24A</td>
</tr>
<tr>
<td>Link Status and Test Results for LDM</td>
<td>NPDA-24AL</td>
<td>BNJMP24L</td>
</tr>
<tr>
<td>LPDA-1 Command Menu</td>
<td>NPDA-LPDA1</td>
<td>BNJMP4LPL1</td>
</tr>
<tr>
<td>LPDA-2 Command Menu</td>
<td>NPDA-LPDA2</td>
<td>BNJMP01A</td>
</tr>
<tr>
<td>Menu</td>
<td>NPDA-01A</td>
<td></td>
</tr>
<tr>
<td>Modem and Line Status Modem and Line Parameters</td>
<td>NPDA-22B, page 1</td>
<td>BNJMPML1</td>
</tr>
<tr>
<td>Link Segment Level ( n )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modem and Line Status Remote Modem Interface-Remote Device Status-Link Segment Level ( n )</td>
<td>NPDA-22B, page 2</td>
<td>BNJMPML2</td>
</tr>
<tr>
<td>Modem and Line Status Configuration Summary, Link Segment Level ( n )</td>
<td>NPDA-22B, page 3</td>
<td>BNJMPML3</td>
</tr>
<tr>
<td>Most Recent Events</td>
<td>NPDA-41A</td>
<td>BNJMP41A</td>
</tr>
<tr>
<td>Most Recent Statistical Data</td>
<td>NPDA-51E NPDA-51F</td>
<td>BNJMP51E BNJMP51F</td>
</tr>
<tr>
<td>Most Recent Statistical Data</td>
<td>NPDA-51G</td>
<td>BNJMP51G</td>
</tr>
<tr>
<td>Most Recent Statistical Data</td>
<td>NPDA-51H NPDA-51I</td>
<td>BNJMP51H BNJMP51I</td>
</tr>
<tr>
<td>Panel Name</td>
<td>Panel Number</td>
<td>Color Map</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Most Recent Statistical Data</td>
<td>NPDA-51B</td>
<td>BNJMP51B</td>
</tr>
<tr>
<td>Most Recent Statistical Data for Printer</td>
<td>NPDA-51D</td>
<td>BNJMP51B</td>
</tr>
<tr>
<td>Most Recent Statistical Data for Tape</td>
<td>NPDA-51C</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Statistics</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Stats for BSC/SS Station</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Stats for BSC STA. w/LPDA</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Stats for Channel Attached STA</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Stats for Local CTRL</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Stats for SDLC Station</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Most Recent Traffic Stats for SDLC STA. w/LPDA</td>
<td>NPDA-70A (all displays)</td>
<td>BNJMP70A</td>
</tr>
<tr>
<td>Multiple Entries for Selected Resource</td>
<td>NPDA-51A</td>
<td>BNJMP51A</td>
</tr>
<tr>
<td>Overwrite Map</td>
<td>BNJMPOVERW</td>
<td></td>
</tr>
<tr>
<td>Recommended Action for Selected Event</td>
<td>NPDA-BN1xxxxyy</td>
<td>BNJMP45A</td>
</tr>
<tr>
<td>Recording and Viewing Filter Status</td>
<td>NPDA-20A,20B</td>
<td>BNJMP20A</td>
</tr>
<tr>
<td>Release Level for SNA Controller</td>
<td>NPDA-21A</td>
<td>BNJMP21A</td>
</tr>
<tr>
<td>Remote DTE Interface Status</td>
<td>NPDA-25A</td>
<td>BNJMP25A</td>
</tr>
<tr>
<td>Remote DTE Interface Status for LDM</td>
<td>NPDA-25AL</td>
<td>BNJMP25A</td>
</tr>
<tr>
<td>Remote Self-Test Results</td>
<td>NPDA-22A</td>
<td>BNJMP22A</td>
</tr>
<tr>
<td>Remote Self-Test Results for LDM</td>
<td>NPDA-22AL</td>
<td>BNJMP22L</td>
</tr>
<tr>
<td>Reported Resource Hardware</td>
<td>NPDA-44B</td>
<td>BNJMP44B</td>
</tr>
<tr>
<td>Reported Resource Software Product</td>
<td>NPDA-02A, page 1</td>
<td>BNJMP2A1</td>
</tr>
<tr>
<td>Screen Control/Help</td>
<td>NPDA-02A, page 2</td>
<td>BNJMP2A2</td>
</tr>
<tr>
<td>Screen Control/Help</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender Hardware Product ID</td>
<td>NPDA-44B</td>
<td>BNJMP44B</td>
</tr>
<tr>
<td>Sender Software Product ID</td>
<td>NPDA-54D</td>
<td>BNJMP44B</td>
</tr>
<tr>
<td>Statistical Counter Detail Display, page 1</td>
<td>NPDA-53E</td>
<td>BNJMP53E</td>
</tr>
<tr>
<td>Statistical Counter Detail Display, page n</td>
<td>NPDA-53F</td>
<td>BNJMP53F</td>
</tr>
<tr>
<td>Statistical Detail</td>
<td>NPDA-53K</td>
<td>BNJMP53K</td>
</tr>
<tr>
<td>Statistical Detail</td>
<td>NPDA-53R</td>
<td>BNJMP43R</td>
</tr>
<tr>
<td>Statistical Detail Display for Ethernet</td>
<td>NPDA-40A</td>
<td>BNJMP40A</td>
</tr>
<tr>
<td>Statistical Detail Menu</td>
<td>NPDA-50A</td>
<td>BNJMP50A</td>
</tr>
<tr>
<td>Menu for BSC</td>
<td>NPDA-53R</td>
<td>BNJMP43R</td>
</tr>
<tr>
<td>Statistical Detail Menu for SDLC</td>
<td>NPDA-02D</td>
<td>BNJMP02D</td>
</tr>
<tr>
<td>TEST Information Display</td>
<td>NPDA-40A</td>
<td>BNJMP40A</td>
</tr>
<tr>
<td>Transmit Receive</td>
<td>NPDA-25B</td>
<td>BNJMPTRRT</td>
</tr>
<tr>
<td>Test-Link Segment Level n</td>
<td>NPDA-44B</td>
<td>BNJMP44B</td>
</tr>
<tr>
<td>Upstream Member of Token-Ring Fault Domain</td>
<td>NPDA-50A</td>
<td>BNJMP50A</td>
</tr>
</tbody>
</table>
Color Maps
Appendix B. NetView Macros and Control Blocks

The macros and control blocks identified in this appendix are provided by the NetView program as programming interfaces for customers.

Attention: Do not use as programming interfaces any NetView macros other than those identified in this appendix.

General-Use Programming Interface Control Blocks and Include Files

The following control blocks and include files are provided as general-use programming interfaces.

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIBC</td>
<td>NetView Bridge HLL C include file</td>
</tr>
<tr>
<td>DSIBCCALL</td>
<td>NetView Bridge HLL C service routine definition</td>
</tr>
<tr>
<td>DSIBCNNM</td>
<td>NetView Bridge HLL C return codes</td>
</tr>
<tr>
<td>DSIBCHLB</td>
<td>NetView Bridge HLL C mapping of DSIHLB</td>
</tr>
<tr>
<td>DSIBPCNM</td>
<td>NetView Bridge HLL PL/I return codes</td>
</tr>
<tr>
<td>DSIBPHLB</td>
<td>NetView Bridge HLL PL/I mapping of DSIHLB</td>
</tr>
<tr>
<td>DSIBPHLS</td>
<td>NetView Bridge HLL PL/I service routine definitions</td>
</tr>
<tr>
<td>DSIBPLI</td>
<td>NetView Bridge HLL PL/I include file</td>
</tr>
<tr>
<td>DSIC</td>
<td>Main HLL C include file</td>
</tr>
<tr>
<td>DSICALL</td>
<td>HLL C service routine definitions</td>
</tr>
<tr>
<td>DSICCNM</td>
<td>HLL C return codes</td>
</tr>
<tr>
<td>DSICCONS</td>
<td>HLL C constants</td>
</tr>
<tr>
<td>DSICHLB</td>
<td>HLL C mapping of DSIHLB</td>
</tr>
<tr>
<td>DSICORIG</td>
<td>HLL C origin block mapping</td>
</tr>
<tr>
<td>DSICPRM</td>
<td>HLL C NetView bridge parameter block</td>
</tr>
<tr>
<td>DSICVARC</td>
<td>HLL C varying length character strings</td>
</tr>
<tr>
<td>DSIPCNM</td>
<td>HLL PL/I return codes</td>
</tr>
<tr>
<td>DSIPCONS</td>
<td>HLL PL/I constants</td>
</tr>
<tr>
<td>DSIPHLB</td>
<td>HLL PL/I mapping of DSIHLB</td>
</tr>
<tr>
<td>DSIPHLLS</td>
<td>PL/I definitions for HLL service routines</td>
</tr>
<tr>
<td>DSIPILI</td>
<td>Main HLL PL/I include file</td>
</tr>
<tr>
<td>DSIPORIG</td>
<td>HLL PL/I origin block mapping</td>
</tr>
<tr>
<td>DSIPPRM</td>
<td>HLL PL/I NetView bridge parameter block</td>
</tr>
<tr>
<td>EKG1ACCB</td>
<td>PL/I RODM access block</td>
</tr>
<tr>
<td>EKG1ENTB</td>
<td>PL/I RODM entity access information block</td>
</tr>
<tr>
<td>EKG1Fldb</td>
<td>PL/I RODM field access information block</td>
</tr>
<tr>
<td>EKG1IADT</td>
<td>PL/I abstract data types</td>
</tr>
<tr>
<td>EKG1IEEP</td>
<td>PL/I external entry point declaration</td>
</tr>
<tr>
<td>EKG1INIC</td>
<td>PL/I include statements</td>
</tr>
<tr>
<td>EKG1LOGT</td>
<td>PL/I log record type definitions</td>
</tr>
<tr>
<td>EKG1TRAB</td>
<td>PL/I RODM transaction information block</td>
</tr>
<tr>
<td>EKG11101</td>
<td>PL/I function block for EKG_Connect</td>
</tr>
<tr>
<td>EKG11102</td>
<td>PL/I function block for EKG_Disconnect</td>
</tr>
<tr>
<td>EKG11201</td>
<td>PL/I function block for EKG_Checkpoint</td>
</tr>
<tr>
<td>EKG11202</td>
<td>PL/I function block for EKG_Stop</td>
</tr>
<tr>
<td>EKG11302</td>
<td>PL/I function block for EKG_CreateClass</td>
</tr>
<tr>
<td>EKG11303</td>
<td>PL/I function block for EKG_DeleteClass</td>
</tr>
<tr>
<td>EKG11304</td>
<td>PL/I function block for EKG_CreateField</td>
</tr>
<tr>
<td>EKG11305</td>
<td>PL/I function block for EKG_DeleteField</td>
</tr>
</tbody>
</table>

181
<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG11306</td>
<td>PL/I function block for EKG_CreateSubfield</td>
</tr>
<tr>
<td>EKG11307</td>
<td>PL/I function block for EKG_DeleteSubfield</td>
</tr>
<tr>
<td>EKG11401</td>
<td>PL/I function block for EKG_ChangeField</td>
</tr>
<tr>
<td>EKG11402</td>
<td>PL/I function block for EKG_SwapField</td>
</tr>
<tr>
<td>EKG11403</td>
<td>PL/I function block for EKG_ChangeSubfield</td>
</tr>
<tr>
<td>EKG11404</td>
<td>PL/I function block for EKG_SwapSubfield</td>
</tr>
<tr>
<td>EKG11405</td>
<td>PL/I function block for EKG_LinkTrigger</td>
</tr>
<tr>
<td>EKG11406</td>
<td>PL/I function block for EKG_LinkNoTrigger</td>
</tr>
<tr>
<td>EKG11407</td>
<td>PL/I function block for EKG_UnLinkTrigger</td>
</tr>
<tr>
<td>EKG11408</td>
<td>PL/I function block for EKG_UnLinkNoTrigger</td>
</tr>
<tr>
<td>EKG11409</td>
<td>PL/I function block for EKG_CreateObject</td>
</tr>
<tr>
<td>EKG11410</td>
<td>PL/I function block for EKG_DeleteObject</td>
</tr>
<tr>
<td>EKG11411</td>
<td>PL/I function block for EKG_RevertToInherited</td>
</tr>
<tr>
<td>EKG11412</td>
<td>PL/I function block for EKG_AddNotifySubscription</td>
</tr>
<tr>
<td>EKG11413</td>
<td>PL/I function block for EKG_DeleteNotifySubscription</td>
</tr>
<tr>
<td>EKG11415</td>
<td>PL/I function block for EKG_TriggerNamedMethod</td>
</tr>
<tr>
<td>EKG11416</td>
<td>PL/I function block for EKG_TriggerOIMethod</td>
</tr>
<tr>
<td>EKG11417</td>
<td>PL/I add object deletion notification subscription</td>
</tr>
<tr>
<td>EKG11418</td>
<td>PL/I delete object deletion notification subscription</td>
</tr>
<tr>
<td>EKG11501</td>
<td>PL/I function block for EKG_QueryField</td>
</tr>
<tr>
<td>EKG11502</td>
<td>PL/I function block for EKG_QuerySubfield</td>
</tr>
<tr>
<td>EKG11503</td>
<td>PL/I function block for EKG_QueryEntityStructure</td>
</tr>
<tr>
<td>EKG11504</td>
<td>PL/I function block for EKG_QueryFieldStructure</td>
</tr>
<tr>
<td>EKG11505</td>
<td>PL/I function block for EKG_QueryFieldID</td>
</tr>
<tr>
<td>EKG11506</td>
<td>PL/I function block for EKG_QueryFieldName</td>
</tr>
<tr>
<td>EKG11507</td>
<td>PL/I function block for EKG_QueryNotifyQueue</td>
</tr>
<tr>
<td>EKG11508</td>
<td>PL/I query multiple subfields</td>
</tr>
<tr>
<td>EKG11509</td>
<td>PL/I locate</td>
</tr>
<tr>
<td>EKG11510</td>
<td>PL/I function block for EKG_QueryResponseBlockOverflow</td>
</tr>
<tr>
<td>EKG11600</td>
<td>PL/I function block for EKG_ExecuteFunctionList</td>
</tr>
<tr>
<td>EKG12001</td>
<td>PL/I function block for EKG_QueryFunctionBlockContents</td>
</tr>
<tr>
<td>EKG12002</td>
<td>PL/I function block for EKG_LockObjectList</td>
</tr>
<tr>
<td>EKG12003</td>
<td>PL/I function block for EKG_UnlockAll</td>
</tr>
<tr>
<td>EKG12004</td>
<td>PL/I function block for EKG_ResponseBlock</td>
</tr>
<tr>
<td>EKG12005</td>
<td>PL/I function block for EKG_SendNotification</td>
</tr>
<tr>
<td>EKG12006</td>
<td>PL/I function block for EKG_SetReturnCode</td>
</tr>
<tr>
<td>EKG12007</td>
<td>PL/I function block for EKG_WHEREAM1</td>
</tr>
<tr>
<td>EKG12008</td>
<td>PL/I function block for EKG_OutputToLog</td>
</tr>
<tr>
<td>EKG12009</td>
<td>PL/I function block for EKG_MessageTriggeredAction</td>
</tr>
<tr>
<td>EKG12011</td>
<td>PL/I function block for EKG_QueryObjectName</td>
</tr>
<tr>
<td>EKG21415</td>
<td>PL/I response block for EKG_TriggerNamedMethod</td>
</tr>
<tr>
<td>EKG21416</td>
<td>PL/I response block for EKG_TriggerOIMethod</td>
</tr>
<tr>
<td>EKG21501</td>
<td>PL/I response block for EKG_QueryField</td>
</tr>
<tr>
<td>EKG21502</td>
<td>PL/I response block for EKG_QuerySubfield</td>
</tr>
<tr>
<td>EKG21503</td>
<td>PL/I response block for EKG_QueryEntityStructure</td>
</tr>
<tr>
<td>EKG21504</td>
<td>PL/I response block for EKG_QueryFieldStructure</td>
</tr>
<tr>
<td>EKG21505</td>
<td>PL/I response block for EKG_QueryFieldID</td>
</tr>
<tr>
<td>EKG21506</td>
<td>PL/I response block for EKG_QueryFieldName</td>
</tr>
<tr>
<td>EKG21507</td>
<td>PL/I response block for EKG_QueryNotifyQueue</td>
</tr>
<tr>
<td>EKG21508</td>
<td>PL/I query multiple subfields</td>
</tr>
<tr>
<td>EKG21509</td>
<td>PL/I locate</td>
</tr>
<tr>
<td>EKG21510</td>
<td>PL/I response block for EKG_QueryResponseBlockOverflow</td>
</tr>
<tr>
<td>EKG22001</td>
<td>PL/I response block for EKG_QueryFunctionBlockContents</td>
</tr>
<tr>
<td>EKG22007</td>
<td>PL/I response block for EKG_WHEREAM1</td>
</tr>
</tbody>
</table>
## NetView Macros and Control Blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG22011</td>
<td>PL/I response block for EKG_QueryObjectName</td>
</tr>
<tr>
<td>EKG3ACCB</td>
<td>C/370™ RODM access block</td>
</tr>
<tr>
<td>EKG3CADT</td>
<td>C/370 RODM abstract data types</td>
</tr>
<tr>
<td>EKG3CEEP</td>
<td>C/370 external entry point declaration</td>
</tr>
<tr>
<td>EKG3CINC</td>
<td>C/370 include statements</td>
</tr>
<tr>
<td>EKG3CLOG</td>
<td>C/370 log record definitions</td>
</tr>
<tr>
<td>EKG3ENTB</td>
<td>C/370 RODM entity access information block</td>
</tr>
<tr>
<td>EKG3FLDB</td>
<td>C/370 RODM field access information block</td>
</tr>
<tr>
<td>EKG3TRAB</td>
<td>C/370 RODM transaction information block</td>
</tr>
<tr>
<td>EKG31101</td>
<td>C/370 function block for EKG_Connect</td>
</tr>
<tr>
<td>EKG31102</td>
<td>C/370 function block for EKG_Disconnect</td>
</tr>
<tr>
<td>EKG31201</td>
<td>C/370 function block for EKG_Checkpoint</td>
</tr>
<tr>
<td>EKG31202</td>
<td>C/370 function block for EKG_Stop</td>
</tr>
<tr>
<td>EKG31302</td>
<td>C/370 function block for EKG_CreateClass</td>
</tr>
<tr>
<td>EKG31303</td>
<td>C/370 function block for EKG_DeleteClass</td>
</tr>
<tr>
<td>EKG31304</td>
<td>C/370 function block for EKG_CreateField</td>
</tr>
<tr>
<td>EKG31305</td>
<td>C/370 function block for EKG_DeleteField</td>
</tr>
<tr>
<td>EKG31306</td>
<td>C/370 function block for EKG_CreateSubfield</td>
</tr>
<tr>
<td>EKG31307</td>
<td>C/370 function block for EKG_DeleteSubfield</td>
</tr>
<tr>
<td>EKG31401</td>
<td>C/370 function block for EKG_ChangeField</td>
</tr>
<tr>
<td>EKG31402</td>
<td>C/370 function block for EKG_SwapField</td>
</tr>
<tr>
<td>EKG31403</td>
<td>C/370 function block for EKG_ChangeSubfield</td>
</tr>
<tr>
<td>EKG31404</td>
<td>C/370 function block for EKG_SwapSubfield</td>
</tr>
<tr>
<td>EKG31405</td>
<td>C/370 function block for EKG_LinkTrigger</td>
</tr>
<tr>
<td>EKG31406</td>
<td>C/370 function block for EKG_LinkNoTrigger</td>
</tr>
<tr>
<td>EKG31407</td>
<td>C/370 function block for EKG_UnLinkTrigger</td>
</tr>
<tr>
<td>EKG31408</td>
<td>C/370 function block for EKG_UnLinkNoTrigger</td>
</tr>
<tr>
<td>EKG31409</td>
<td>C/370 function block for EKG_CreateObject</td>
</tr>
<tr>
<td>EKG31410</td>
<td>C/370 function block for EKG_DeleteObject</td>
</tr>
<tr>
<td>EKG31411</td>
<td>C/370 function block for EKG_RevertToInherited</td>
</tr>
<tr>
<td>EKG31412</td>
<td>C/370 function block for EKG_AddNotifySubscription</td>
</tr>
<tr>
<td>EKG31413</td>
<td>C/370 function block for EKG_DeleteNotifySubscription</td>
</tr>
<tr>
<td>EKG31415</td>
<td>C/370 function block for EKG_TriggerNamedMethod</td>
</tr>
<tr>
<td>EKG31416</td>
<td>C/370 function block for EKG_TriggerOfMethod</td>
</tr>
<tr>
<td>EKG31417</td>
<td>C/370 add object deletion notification subscription</td>
</tr>
<tr>
<td>EKG31418</td>
<td>C/370 delete object deletion notification subscription</td>
</tr>
<tr>
<td>EKG31501</td>
<td>C/370 function block for EKG_QueryField</td>
</tr>
<tr>
<td>EKG31502</td>
<td>C/370 function block for EKG_QuerySubfield</td>
</tr>
<tr>
<td>EKG31503</td>
<td>C/370 function block for EKG_QueryEntityStructure</td>
</tr>
<tr>
<td>EKG31504</td>
<td>C/370 function block for EKG_QueryFieldStructure</td>
</tr>
<tr>
<td>EKG31505</td>
<td>C/370 function block for EKG_QueryFieldID</td>
</tr>
<tr>
<td>EKG31506</td>
<td>C/370 function block for EKG_QueryFieldName</td>
</tr>
<tr>
<td>EKG31507</td>
<td>C/370 function block for EKG_QueryNotifyQueue</td>
</tr>
<tr>
<td>EKG31508</td>
<td>C/370 query multiple subfields</td>
</tr>
<tr>
<td>EKG31509</td>
<td>C/370 locate</td>
</tr>
<tr>
<td>EKG31510</td>
<td>C/370 function block for EKG_QueryResponseBlockOverflow</td>
</tr>
<tr>
<td>EKG31600</td>
<td>C/370 function block for EKG_ExecuteFunctionList</td>
</tr>
<tr>
<td>EKG32001</td>
<td>C/370 function block for EKG_QueryFunctionBlockContents</td>
</tr>
<tr>
<td>EKG32002</td>
<td>C/370 function block for EKG_LockObjectList</td>
</tr>
<tr>
<td>EKG32003</td>
<td>C/370 function block for EKG_UnlockAll</td>
</tr>
<tr>
<td>EKG32004</td>
<td>C/370 function block for EKG_ResponseBlock</td>
</tr>
<tr>
<td>EKG32005</td>
<td>C/370 function block for EKG_SendNotification</td>
</tr>
<tr>
<td>EKG32006</td>
<td>C/370 function block for EKG_SetReturnCode</td>
</tr>
<tr>
<td>EKG32007</td>
<td>C/370 function block for EKG_WhereAmI</td>
</tr>
</tbody>
</table>
NetView Macros and Control Blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKG32008</td>
<td>C/370 function block for EKG_OutputToLog</td>
</tr>
<tr>
<td>EKG32009</td>
<td>C/370 function block for EKG_MessageTriggeredAction</td>
</tr>
<tr>
<td>EKG32011</td>
<td>C/370 function block for EKG_QueryObjectName</td>
</tr>
<tr>
<td>EKG41415</td>
<td>C/370 response block for EKG_TriggerNamedMethod</td>
</tr>
<tr>
<td>EKG41416</td>
<td>C/370 response block for EKG_TriggerOIMethod</td>
</tr>
<tr>
<td>EKG41501</td>
<td>C/370 response block for EKG_QueryField</td>
</tr>
<tr>
<td>EKG41502</td>
<td>C/370 response block for EKG_QuerySubfield</td>
</tr>
<tr>
<td>EKG41503</td>
<td>C/370 response block for EKG_QueryEntityStructure</td>
</tr>
<tr>
<td>EKG41504</td>
<td>C/370 response block for EKG_QueryFieldStructure</td>
</tr>
<tr>
<td>EKG41505</td>
<td>C/370 response block for EKG_QueryFieldID</td>
</tr>
<tr>
<td>EKG41506</td>
<td>C/370 response block for EKG_QueryFieldName</td>
</tr>
<tr>
<td>EKG41507</td>
<td>C/370 response block for EKG_QueryNotifyQueue</td>
</tr>
<tr>
<td>EKG41508</td>
<td>C/370 query multiple subfields</td>
</tr>
<tr>
<td>EKG41509</td>
<td>C/370 locate</td>
</tr>
<tr>
<td>EKG41510</td>
<td>C/370 response block for EKG_QueryResponseBlockOverflow</td>
</tr>
<tr>
<td>EKG42001</td>
<td>C/370 response block for EKG_QueryFunctionBlockContents</td>
</tr>
<tr>
<td>EKG42007</td>
<td>C/370 response block for EKG_WhereAmI</td>
</tr>
<tr>
<td>EKG42011</td>
<td>C/370 response block for EKG_QueryObjectName</td>
</tr>
<tr>
<td>FLBTREM</td>
<td>C/370 exception view update parameter structure</td>
</tr>
<tr>
<td>FLBTRSM</td>
<td>C/370 status change parameter structure</td>
</tr>
</tbody>
</table>

The following macros are provided as general-use programming interfaces.

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNMALTDATA</td>
<td>Alter data on a queue</td>
</tr>
<tr>
<td>CNMAUTOTAB</td>
<td>Invoke automation table</td>
</tr>
<tr>
<td>CNMCLOS MEM</td>
<td>Close NetView partitioned data set</td>
</tr>
<tr>
<td>CNMCODE2TXT</td>
<td>Code point translation</td>
</tr>
<tr>
<td>CNMCOMMAND</td>
<td>Invoke NetView commands</td>
</tr>
<tr>
<td>CNMCPYSTR</td>
<td>Copy storage</td>
</tr>
<tr>
<td>CNMETINIT</td>
<td>Initialize the server support</td>
</tr>
<tr>
<td>CNMETNEXT</td>
<td>Get next transaction request</td>
</tr>
<tr>
<td>CNMETQUIESCE</td>
<td>Quiesce the database</td>
</tr>
<tr>
<td>CNMETREADY</td>
<td>Ready for next transaction</td>
</tr>
<tr>
<td>CNMETRPARM</td>
<td>Get transaction request</td>
</tr>
<tr>
<td>CNMETTERM</td>
<td>Terminate the Server support</td>
</tr>
<tr>
<td>CNMETWAIT</td>
<td>Wait for a transaction request</td>
</tr>
<tr>
<td>CNMGETATTR</td>
<td>Query message attributes</td>
</tr>
<tr>
<td>CNMGETDATA</td>
<td>Data queue manipulation</td>
</tr>
<tr>
<td>CNMGETTPARM</td>
<td>Get transaction reply parameters</td>
</tr>
<tr>
<td>CNMHREGIST</td>
<td>High performance transport application registration</td>
</tr>
<tr>
<td>CNMHSENDMU</td>
<td>Send high performance message unit</td>
</tr>
<tr>
<td>CNMI</td>
<td>CNMI access under a DST</td>
</tr>
<tr>
<td>CNMINFOC</td>
<td>Query NetView character information</td>
</tr>
<tr>
<td>CNMINFOI</td>
<td>Query NetView integer information</td>
</tr>
<tr>
<td>CNMKEYIO</td>
<td>Keyed file access under a DST</td>
</tr>
<tr>
<td>CNMLOCK</td>
<td>Control a lock</td>
</tr>
<tr>
<td>CNMNAMESTR</td>
<td>Named storage</td>
</tr>
<tr>
<td>CNMOPENMEM</td>
<td>Open NetView partitioned data set</td>
</tr>
<tr>
<td>CNMOPREP</td>
<td>Resource object data manager</td>
</tr>
<tr>
<td>CNMPRSMDB</td>
<td>Process message data block</td>
</tr>
<tr>
<td>CNMRREADMEM</td>
<td>Read NetView partitioned data set</td>
</tr>
<tr>
<td>CNMRREGIST</td>
<td>Application registration</td>
</tr>
<tr>
<td>CNMSCOPECK</td>
<td>Check command authorization for security</td>
</tr>
</tbody>
</table>
Product-Sensitive Programming Interfaces

The following control blocks are provided as product-sensitive programming interfaces.

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAUTISAW</td>
<td>Internal session awareness record</td>
</tr>
<tr>
<td>AAUTLOGR</td>
<td>Structure map for NetView SMF log record</td>
</tr>
<tr>
<td>BNJTBRF</td>
<td>Batch record format table</td>
</tr>
<tr>
<td>DSIAIFRO</td>
<td>Automation internal function request object extension vector</td>
</tr>
<tr>
<td>DSIASYPN</td>
<td>Asynchronous panel parameter list</td>
</tr>
<tr>
<td>DSICBH</td>
<td>Control block header</td>
</tr>
<tr>
<td>DSICWB</td>
<td>Command work block</td>
</tr>
<tr>
<td>DSIDSB</td>
<td>Data services block</td>
</tr>
<tr>
<td>DSIDSRB</td>
<td>Data services request block</td>
</tr>
<tr>
<td>DSIDTR</td>
<td>Data transport Request block</td>
</tr>
<tr>
<td>DSIELB</td>
<td>External logging block</td>
</tr>
<tr>
<td>DSIID</td>
<td>NetView level identifier</td>
</tr>
<tr>
<td>DSIIFR</td>
<td>Internal function request</td>
</tr>
<tr>
<td>DSILogDS</td>
<td>NetView log DSECT</td>
</tr>
<tr>
<td>DSIMVT</td>
<td>Main vector table</td>
</tr>
<tr>
<td>DSIPDB</td>
<td>Parse descriptor block</td>
</tr>
<tr>
<td>DSISCE</td>
<td>System command entry</td>
</tr>
<tr>
<td>DSISCT</td>
<td>System command table (include only)</td>
</tr>
<tr>
<td>DSISVL</td>
<td>Service routine vector list (include only)</td>
</tr>
<tr>
<td>DSISWB</td>
<td>Service work block</td>
</tr>
<tr>
<td>DSITECBR</td>
<td>Branch table of ECB processor load module</td>
</tr>
<tr>
<td>DSITIB</td>
<td>Task information block</td>
</tr>
<tr>
<td>DSITVB</td>
<td>Task vector block</td>
</tr>
<tr>
<td>DSIUUE</td>
<td>Installation exit parameter list</td>
</tr>
<tr>
<td>DUITRXCM</td>
<td>RSM resource command support for XITCM exit</td>
</tr>
<tr>
<td>DUITSTAT</td>
<td>RSM resource status information for XITST exit</td>
</tr>
</tbody>
</table>

The following macros are provided as product-sensitive programming interfaces.

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIAUTO</td>
<td>Automation services</td>
</tr>
<tr>
<td>DSIBAM</td>
<td>Build automation message</td>
</tr>
<tr>
<td>DSIBAMKW</td>
<td>Build automation message keyword</td>
</tr>
<tr>
<td>DSICBS</td>
<td>Control block services</td>
</tr>
<tr>
<td>DSICES</td>
<td>Command entry services</td>
</tr>
<tr>
<td>DSICVTHE</td>
<td>Convert to hex</td>
</tr>
<tr>
<td>DSIC2T</td>
<td>Translate alert code point to text</td>
</tr>
</tbody>
</table>
## NetView Macros and Control Blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIDATIM</td>
<td>Date and time</td>
</tr>
<tr>
<td>DSIDEL</td>
<td>Delete user-defined module</td>
</tr>
<tr>
<td>DSIDKS</td>
<td>Disk services</td>
</tr>
<tr>
<td>DSIFIND</td>
<td>Find long-running command storage</td>
</tr>
<tr>
<td>DSIFRE</td>
<td>Free storage</td>
</tr>
<tr>
<td>DSIFREBS</td>
<td>Free buffer structure service</td>
</tr>
<tr>
<td>DSIGET</td>
<td>Get storage</td>
</tr>
<tr>
<td>DSIGETDS</td>
<td>Retrieve messages</td>
</tr>
<tr>
<td>DSIHREGS</td>
<td>High-performance registration</td>
</tr>
<tr>
<td>DSIHSNDS</td>
<td>High-performance send</td>
</tr>
<tr>
<td>DSIKVS</td>
<td>Keyword/value services</td>
</tr>
<tr>
<td>DSIILCS</td>
<td>Obtain/release control blocks</td>
</tr>
<tr>
<td>DSILOD</td>
<td>Load user-defined module</td>
</tr>
<tr>
<td>DSIMBS</td>
<td>Message buffer services</td>
</tr>
<tr>
<td>DSIMDS</td>
<td>Message definition services</td>
</tr>
<tr>
<td>DSIMMDBS</td>
<td>Message data block service</td>
</tr>
<tr>
<td>DSIMQS</td>
<td>Message queuing services</td>
</tr>
<tr>
<td>DSINOR</td>
<td>Resource object data manager d</td>
</tr>
<tr>
<td>DSIPAS</td>
<td>Parameter/alias services</td>
</tr>
<tr>
<td>DSIPOP</td>
<td>Remove long-running command</td>
</tr>
<tr>
<td>DSIPPOS</td>
<td>ECB post services</td>
</tr>
<tr>
<td>DSIPRS</td>
<td>Parsing services</td>
</tr>
<tr>
<td>DSIPSS</td>
<td>Presentation services</td>
</tr>
<tr>
<td>DSIPUSH</td>
<td>Establish long-running command</td>
</tr>
<tr>
<td>DSIQOS</td>
<td>Query operator services</td>
</tr>
<tr>
<td>DSIQRS</td>
<td>Query resource services</td>
</tr>
<tr>
<td>DSIRDS</td>
<td>Resource definition services</td>
</tr>
<tr>
<td>DSIRXCOM</td>
<td>Access REXX variables (VM only)</td>
</tr>
<tr>
<td>DSIRXEB5</td>
<td>Get an EVALBLOK</td>
</tr>
<tr>
<td>DSISRCMV</td>
<td>Search for subvector/subfield</td>
</tr>
<tr>
<td>DSISYS</td>
<td>Operating system indicator</td>
</tr>
<tr>
<td>DSITECBS</td>
<td>Manage a dynamic ECB list for DSTs</td>
</tr>
<tr>
<td>DSIVARS</td>
<td>Global Variable Access</td>
</tr>
<tr>
<td>DSIWAT</td>
<td>ECB wait services</td>
</tr>
<tr>
<td>DSIWCS</td>
<td>Write console services</td>
</tr>
<tr>
<td>DSIWLS</td>
<td>Write log services</td>
</tr>
<tr>
<td>DSIZCSMS</td>
<td>CNM data services</td>
</tr>
<tr>
<td>DSIZVSMJ</td>
<td>VSAM data services</td>
</tr>
<tr>
<td>DSIFREGS</td>
<td>Registration services</td>
</tr>
<tr>
<td>DSIFSNDS</td>
<td>Send services</td>
</tr>
</tbody>
</table>
Index

Special Characters
&CGLOBAL 40
&CUR 32, 54
&SUPPCHAR 46
&TGLOBAL 40
&VIEWAID 47
&VIEWCOLS 49
&VIEWCURROW 47
&VIEWICCOL 47
&VIEWICROW 47
&VIEWROWS 49
&WAIT 108

A
access method 5, 12
accessibility information ix
ACTION command list 81, 85
ACTION statement, SCRNFMT 26
activate screen format definition 25
actual panel name
adding 79
changing panel text 78
adding functions 3
adding links
Web application server 172
AID (attention identification) information 47
alert adapter service
event/automation service 111
alert-to-trap service
event/automation service 111
alerts
description 1, 91
generic
build panel 92
description 91
modify 75
NMVT 90
recommended action code point 83
record 75
reference documentation, table 4
sample record 92
Tivoli-supplied alert table 92
message 75
nongeneric
messages 80
migration purposes 90
modify 75
sender 85
user-defined 90, 91
Alerts-Dynamic panel 80
Alerts-History panel 75, 80
Alerts-Static panel 75, 80
alias names
definition 1
reference documentation, table 4
alias panel name
adding 79
determining 75, 76
application, performance-critical 13
application management
instrumentation 159
APPLID NetView control variable 41
assembled command procedure 13
attribute
symbols 35
variables 37
audible alarm 85
automated operations
definition 1
NetView automation 1
automation table
setting message color and highlighting 28
VPDXDOM command list 106, 107
autotask 109
B
BGNESS FLSCN command 43
block ID 76
BNJALxxx sample table 78
BNJBLKID sample table 77
BNJDNUMB 82
BNJDSERV task 103
BNJPNL2 DD statement 84
BNJPNL2 definition statement 102
BNJPROMP (prompt highlight token table) 89
BNJRESTY member 102
BNJwwwww code point members 84
books
feedback viii
online viii
ordering viii
BROWSE command, view help 64
code point (continued)
detail data (BNJ92UTB) 99
failure cause (BNJ96UTB) 99
install cause (BNJ95UTB) 99
probable cause (BNJ93UTB) 99
recommended action (BNJ81UTB) 84, 99
user cause (BNJ94UTB) 99
color, panel text 29
color and highlighting fields, control 35
color buffer 86, 89
color maps
BNJOVERW 86
hardware monitor panel 177
list 177
map element 86
repitition factor option 88
repitition map element 88
sample 87
variable row 88
column headings, NCCF panel
control tags, PREFIX and NOPREFIX statements 26
customizing, COLUMNNHEAD statement 26
COLUMNNHEAD statement,
SCRNFMT 26
command area, NCCF panel 28
command buffers 9
command entry indicator, NCCF panel 28
command facility console 103
command facility panel, customizing 25
command facility panel attributes 25
command help
copying 64
locating source files 63
modifying 67
storing 68
command line 54
command lists
error message 106
modifying 33
variable 6, 32
writing 33
command procedure, issuing 43
command processor, interface 9, 11
commands
data services 10
immediate 9
long-running 9
compiled language 13
compound symbols in source panels 42
concatenated user library 91
control blocks
access 13
general-use 181
product-sensitive 181
control program text title 99
control variable 40, 41
CREATE option 106
customizing hardware monitor displayed data (continued)
  using NMVT support for user-written programming (continued)
  modifying generic code point tables 99
  table format 99

D
  data file 6
  data services task (DST) subtask 8
  DCE (data communication equipment) 105, 106
  DEFAULTS command, activate screen format definition 25
  designing functions
    choosing languages
      introduction 12
      logging 15
      performance 13
    identifying conceptual component
      adding optional task 8
      collecting data 4
      data file 5
      data storage and record 6
      defining transaction 10
      exit and command 10
      installation exit 5
      operator command and message 6
      operator presentation 6
      service routine 5
      task structure 7
  designing HTML files
    Web application server 171
    detail data code point 92
    direct NNT session 108
    direct OST session 108
    directory list of panel names 76
    disabling information ix
    displayed data, hardware monitor 75
    displaying
      special attributes 36
    documentation for customizing 3
    domain id area, NCCF panel 26
    DRD (dynamic reconfiguration deck) 107
    DSIAMAT 159
    DSIAMII 160
    DSIELT 107
    DSIMDS macro 81, 91
    DSIPOP 46
    DSPUSH 43, 46
    DST (data services task) subtask 8
    dynamic reconfiguration deck (DRD) 107

E
  e-mail contact ix
  embed flag 102
  END record 106, 107
  event/automation service (continued)
    overview 111
    starting 112
  event detail panel 75, 76, 80
  event receiver service
    event/automation service 111
  exit, installation 5
  exit routine, installation 13

F
  feedback about publications ix
  filter
    definition 1
    hardware monitor 1
    messages 1
    reference documentation, table 4
  focal point VPD collection 107
  full-screen panel, display 29
  functional extension 7
  functions, design and implement 1

G
  GENALERT command 92
  general-use programming interfaces 181
  generic alert code point 75
  generic alert record 75
  global variable 40, 57
  GLOBALV 40
  GO command 14
  group control system 7

H
  HALT subroutine 46
  hardware monitor panels
    altering text
      color 85
      highlighting 85
      intensity 85
    audible alarm 85
    displayed data 75
    displays, list 177
    message
      System job title 129
      user interface 36
      Web application server 173
    history log
      Web application server 173
      hierarchy complete 95
      highlight fields, control color 35
      highlight panel text 29
    using NMVT support for user-written programming
      adding or modifying resource type 102
      building generic alert panel 92
    overlays
      Recommended Action panel 81
    hardware product identifier 83
    held and action message area, NCCF panel 27
    held messages, NCCF panel warning 27
    HELD statement, SCRNFMT 26
    HELPDESK, changing 67
    HELPDESK, searching 68
    history log
      Web application server 173
      hierarchy complete 95
      highlight fields, control color 35
      highlight panel text 29

 188 Tivoli NetView for z/OS Customization Guide
IEBUPDTE utility 79
IEHPROGM utility 79
IHSAEVNT 112
IMDAREA statement, SCRNFMT 28
immediate message area, NCCF panel 28
immediate message line, customizing 23
indent, NCCF panel 28
INDENT statement, SCRNFMT 28
information, accessibility ix
information, disability ix
INITAMI 162, 165
INITAMON 165
input-capable fields 49
INPUT 54
variable 47
input field 52
INPUT keyword 47
input value 33
installation exit interface 4
programs 10
routine 5
routine 13
setting message color and highlighting 28
instrumentation considerations 159
customizing 159
messages 159
starting 162
instrumentation, stopping 162, 165
inventory data, collecting 105
keyboard, shortcut keys ix
languages, choosing 12
LASTLINE statement, SCRNFMT 28
limitations
background message color, 3270 28
customizing NCCF panel 25
displaying held messages 27
NORMQMAX statement value 27
setting message default colors 26
link-edit load module name 91
LOADCL command 13
loading panels
Web application server 173
local variable, REXX 41
lock/unlock indicator, NCCF panel 28
LOCKIND statement, SCRNFMT 28
logging facilities 6
logging method 15
LPDA-2 architecture 105
macros, product-sensitive 185
managing additional component 3
managing online help 63
National Language Support (NLS) kanji feature 2
message help 64
message translations 2
reference documentation, table 4
NCCF panel, customizing 25
NetView automation table 106, 107
NetView component, definition 44
log 40
panel library 91
NetView 3270 management console host access class library 167
NetView command facility panel 25
network log 15
management data 5
qualified procedure correlation identifier 99
network asset management (NAM) command list modifying 108
VPDAct command list 106
VPDCE command list 106
VPDLOGC command list 106
VPDPU command list 106
VPDxDOM command list 106
new management function 3
new online help creating 67
storing 68
structuring conventions 67
NMVT (network management vector transport) 90
NOINPUT option creating rollable components 44
definition 33
displaying online help panels 33
return command line input 54
NOMSG option 34
nongeneric alerts 90
NOPREFIX statement, SCRNFMT 26
NORMAL statement, SCRNFMT 26
NORMQMAX statement, SCRNFMT 26
extreme value, calling attention to 27
minimum value 27
OST-NNT cross-domain sessions 27
printers 27
queueing messages for later display 27
specifying infinite queues 27
values 27
online help copying 64
creating new help 67
highlighting attributes 64
locating source files 63
modifying command help 67
procedure 64
regular help 67
source 67
naming 68
organization 63
source 64
store procedures 68
writing 67
online help panels
color attributes 36
highlighting attributes 36
online publications viii
operator command 6
operator command interface 44
operator control and security command authorization 2
reference documentation, table 4
span of control 2
operator id area, NCCF panel 26
OPID NetView control variable 40, 41
OPT (optional) subtask 8
OPT task, adding 12
ordering publications viii
output area, NCCF panel 26
OVERRIDE command, activate screen format definition 25
overwrite global variable 41

P
panel
data stream 68
definition, using with VIEW 29
definition statement 40
hardware monitor 75
partitioned data set 63
record length 67
variables 37
panels, loading
Web application server 173
partial command, predefining 54
PAUSE command 14
performance 13
PF keys, customizing 23
PF keys, using with VIEW 55
physical unit (PU) 105, 106
PREFIX statement, SCRNFMT 26
preload
NetView command list. 13
REXX command list 13
probable cause code point 91
product-sensitive
control blocks 181
macros 185
product-set identification (PSID) 82
program function keys, using with VIEW 55
programming interfaces
general-use 181
product-sensitive 185
PROMOTE option, UNIQUE command 46
prompt highlight token table 89
PSID (product-set identification) 82
publications
feedback viii
online viii
ordering viii

Q
queueing commands 44

R
recommended action number 81
Recommended Action panel 75, 76
record filters 1
record format, building 108
referencing commands
Web application server 171
referencing files
Web application server 171
regular help panel 64, 67, 68
repetition factor option 88
repetition map element 88
REQUEST/REPLY PSID architecture 105
RESIDYN command list output example 58
RESET command 108
RESOURCE command 57
resource type
adding 102
modifying 102
return codes 33, 34, 45, 47
REXX function CGI
Web application server 174
REXX-generated HTML
Web application server 174
REXX programming language, local variable 41
ROLL command 43
roll group 43, 46
rollable component
creating 44
REXX command procedure that drives 51

S
screen format definition (SCRNFMT)
command facility panel attributes 25
customizable fields
CUMLNHEAD line 26
column area 28
column entry indicator 28
current date 26
domain identifier 26
held and action message area 27
immediate message area 28
indentation 28
lock/unlock indicator 28
operator identifier 26
output area 26
separator line 28
system states 26
time of last display 26
title area 26
message color default value 25
SCRNFMT statements
ACTION 26
CMDLINE 28
COLUMNS 26
HELD 26
HOLDPCNT 27
HOLDWARN 27
IMDAREA 28
INDENT 28
LASTLINE 28
LOCINDEX 28
MLINDEX 28
NOPEN 26
NORM 26
NORMMAX 26
PREFIX 26
TITLE 26
TITLEDATE 26
TITLEDOMID 26
TITLEOID 26
TITLES 26
TITLETE 26
secondary extent 67, 85
security token
Web application server 174
sense code descriptions, customizing 71
separator line, NCCF panel 28
sequential data set 68
sequential logging
definition 2
reference documentation, table 4
service level reporter (SLR) 108
serviceable component identifier 83
session monitor data
definition 2
performance classes 2
reference documentation, table 4
response time monitor (RTM) 2
shortcut keys, keyboard ix
SHOWCODE command list 34
SMF log 5
SMF logging failure 106
SMF record format, changing 108
SMF record number 108
source, help
building 67
definition 64
locating 63
modifying 67
structure 67
viewing 64
source, helps
sample panel 30
specialized disk service 5, 12
START DOMAIN command 108
START record 106, 107
START VPDTASK 108
STARTCNM NPDA 103
status area, NCCF panel 26
STOP TASK 103

190  Tivoli NetView for z/OS Customization Guide
storing new or modified help 68
symbols, compound 42
system allocation 5, 12
system interface 7

T

task, operator station (OST) 7
task variable 14
TERMAMI 162
TERMAMON 165
tilde definition 54
time area, NCCF panel 26
title area, NCCF panel 26
TITLE statement, SCRFMT 26
TITLEDATE statement, SCRFMT 26
TITLEDOMID statement, SCRFMT 26
TITLEOPID statement, SCRFMT 26
TITLETIME statement, SCRFMT 26
Tivoli Customer Support ix
Tivoli Enterprise Console
customizing 162
transaction program
command processor 10
installation exit 10
trap-to-alert service
event/automation service 111

U

UNIQUE command 33, 45
UPPER command 44
user-defined alert
generic 91
nongeneric 90
user interface
BNJDNUMB 82
BNJnwww 84
user subtask, writing 12
user table, defining
BNJ81UTB 99
BNJ82UTB 99
BNJ92UTB 99
BNJ93UTB 99
BNJ94UTB 99
BNJ95UTB 99
BNJ96UTB 99
sample 101
user-written functions
definition 2
reference documentation, table 4

V

variable row placement option 88
variables, compound 42
vector transport, network management
(NMVT) 90
VIEW command, using 29
VIEW command processor
attribute definition 35
code 32
creating rollable components 44
definition statement 40

Web application server (continued)
displaying error messages 34
displaying return codes 34
displaying variables in source panels 40
dynamic update capability 56
finding global variables 40
full-screen input capability 47
global variable 40
INPUT option 33
input value 33
issuing from command procedure 43
managing command lines 59
managing PF keys 59
message data 33
MSG option 56
NOINPUT option 33
panel definition
attribute symbol 35
attribute variable 37
controlling color 35
controlling highlighting 35
return code 33
return command line input 54
subcommands 55
using 29
using PF keys 55
using SHOWCODE command list 34
using UNIQUE command 45
using UPPER command 44
VIEWAID variable 49
view filters 1
VIEWAID variable 49
VIEWCOLS variable 49
VIEWCURCOL variable 47
VIEWCURROW variable 47
VIEWICCOL variable 47, 48
VIEWICROW variable 47, 48
VIEWROWS variable 49
vital product data (VPD), definition 105
VPD command 106, 108
VPDACT command 106
VPDALL command 107
VPDDCE command entry 107
VPDLOGC command list 106, 107
VPDPU command entry 107
VPDTASK 106
VPDXDOM command list 106, 107
VSAM data service 5, 12
VTAM ACB Monitor
starting 165
VTAM CNAME 5
VTAM configuration member in
VTAMLST 106, 107
VTAMLST 106

W

Web application server
adding links 172
designing HTML files 171
history log 173
including references 175
loading panels 173
migration 174
referencing files 171
REXX function CGI 174

X

XVAR 31, 43

Index 191