IBM Z NetView
Version 6.Release 3

Application Programmer's Guide

SC27-2870-03

Note

Before using this information and the product it supports, read the information in “Notices” on page 155.

This edition applies to version 6, release 3 of IBM Z NetView (product number 5697-NV6) and to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

This edition replaces SC27-2870-02.

Contents

Figures

About this publication

The IBM Z® NetView® product provides advanced capabilities that you can use to maintain the highest degree of availability of your complex, multi-platform, multi-vendor networks and systems from a single point of control. This publication, the *IBM Z NetView Application Programmer's Guide*, is written for programmers working with the NetView product and other related products. Use it to perform the following tasks:

- Use the NetView LU 6.2 transport application programming interfaces (APIs).
- Write programs that send network management vector transport (NMVT) and control point management service unit (CP-MSU) formatted alerts to the hardware monitor for processing; write programs that send data buffers to, or receive data buffers from, other application programs.
- Use common operations services (COS) commands and COS command lists.

Intended audience

This publication is for system programmers. Readers should be familiar with the Systems Network Architecture (SNA) requirements described in *Systems Network Architecture Formats* and in the *SNA/Management Services Alert Implementation Guide*.

Publications

This section lists publications in the IBM Z NetView library and related documents. It also describes how to access NetView publications online and how to order NetView publications.

IBM Z NetView library

The following documents are available in the IBM Z NetView library:

- *Administration Reference*, SC27-2869, describes the NetView program definition statements required for system administration.
- *Application Programmer's Guide*, SC27-2870, describes the NetView program-to-program interface (PPI) and how to use the NetView application programming interfaces (APIs).
- *Automation Guide*, SC27-2846, describes how to use automated operations to improve system and network efficiency and operator productivity.
- *Command Reference Volume 1 (A-N)*, SC27-2847, and *Command Reference Volume 2 (O-Z)*, SC27-2848, describe the NetView commands, which can be used for network and system operation and in command lists and command procedures.
- *Installation: Configuring Additional Components*, GC27-2851, describes how to configure NetView functions beyond the base functions.
- *Installation: Configuring the NetView Enterprise Management Agent*, GC27-2853, describes how to install and configure the IBM Z NetView Enterprise Management Agent.
- *Installation: Getting Started*, GI11-9443, describes how to install and configure the base NetView program.
- *Installation: Migration Guide*, GC27-2854, describes the new functions that are provided by the current release of the NetView product and the migration of the base functions from a previous release.
- *IP Management*, SC27-2855, describes how to use the NetView product to manage IP networks.
- *Messages and Codes Volume 1 (AAU-DSI)*, GC27-2856, and *Messages and Codes Volume 2 (DUI-IHS)*, GC27-2857, describe the messages for the NetView product, the NetView abend codes, the sense codes that are included in NetView messages, and generic alert code points.

- *Programming: Pipes*, SC27-2859, describes how to use the NetView pipelines to customize a NetView installation.
- *Programming: REXX and the NetView Command List Language*, SC27-2861, describes how to write command lists for the NetView product using the Restructured Extended Executor language (REXX) or the NetView command list language.
- *Security Reference*, SC27-2863, describes how to implement authorization checking for the NetView environment.
- *Troubleshooting Guide*, GC27-2865, provides information about documenting, diagnosing, and solving problems that occur in the NetView product.
- *Tuning Guide*, SC27-2874, provides tuning information to help achieve certain performance goals for the NetView product and the network environment.
- *User's Guide: Automated Operations Network*, SC27-2866, describes how to use the NetView Automated Operations Network (AON) component, which provides event-driven network automation, to improve system and network efficiency. It also describes how to tailor and extend the automated operations capabilities of the AON component.
- *User's Guide: NetView*, SC27-2867, describes how to use the NetView product to manage complex, multivendor networks and systems from a single point.
- *User's Guide: NetView Enterprise Management Agent*, SC27-2876, describes how to use the NetView Enterprise Management Agent.
- *Using Tivoli System Automation for GDPS/PPRC HyperSwap Manager with NetView*, GI11-4704, provides information about the Tivoli® System Automation for GDPS®/PPRC HyperSwap® Manager with NetView feature, which supports the GDPS and Peer-to-Peer Remote Copy (PPRC) HyperSwap Manager services offering.
- *Licensed Program Specifications*, GC31-8848, provides the license information for the NetView product.
- *Program Directory for IBM Z NetView US English*, GI11-9444, contains information about the material and procedures that are associated with installing the NetView product.
- *Program Directory for IBM Z NetView Japanese*, GI11-9445, contains information about the material and procedures that are associated with installing the NetView product.
- *Program Directory for IBM Z NetView Enterprise Management Agent*, GI11-9446, contains information about the material and procedures that are associated with installing the IBM Z NetView Enterprise Management Agent.

The following books are archived:

- *Customization Guide*, SC27-2849, describes how to customize the NetView product and points to sources of related information.
- *Data Model Reference*, SC27-2850, provides information about the Graphic Monitor Facility host subsystem (GMFHS), SNA topology manager, and MultiSystem Manager data models.
- *Installation: Configuring Graphical Components*, GC27-2852, describes how to install and configure the NetView graphics components.
- *Programming: Assembler*, SC27-2858, describes how to write exit routines, command processors, and subtasks for the NetView product using assembler language.
- *Programming: PL/I and C*, SC27-2860, describes how to write command processors and installation exit routines for the NetView product using PL/I or C.
- *Resource Object Data Manager and GMFHS Programmer's Guide*, SC27-2862, describes the NetView Resource Object Data Manager (RODM), including how to define your non-SNA network to RODM and use RODM for network automation and for application programming.
- *SNA Topology Manager Implementation Guide*, SC27-2864, describes planning for and implementing the NetView SNA topology manager, which can be used to manage subarea, Advanced Peer-to-Peer Networking, and TN3270 resources.
- *User's Guide: NetView Management Console*, SC27-2868, provides information about the NetView management console interface of the NetView product.

Related publications

To use the information in this publication effectively, you need some additional knowledge, which you can obtain from the following publications:

- *Systems Network Architecture Formats*, GA27-3136-19

  Provides information about the IBM® Systems Network Architecture (SNA) formats, including those used between subarea nodes, between subarea nodes and peripheral nodes, and between nodes that implement Advanced Peer-to-Peer Networking or low-entry networking (LEN) protocols.

- *SNA/Management Services Alert Implementation Guide*, GC31-6809-00

  Provides information about the alert mechanism defined by SNA/Management Services as well as information about how to construct an alert.

You can find additional product information on the IBM Z NetView web site at https://www.ibm.com/us-en/marketplace/ibm-tivoli-netview-for-zos.

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

Terminology in this Library

The following terms are used in this library:

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

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

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

IBM Tivoli Netcool®/OMNIbus
For either of these products:

- IBM Tivoli Netcool/OMNIbus
- IBM Tivoli OMNIbus and Network Manager

MVS™
For z/OS® operating systems

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

NetView
For the following products:

- IBM Z NetView version 6 release 3
- IBM Tivoli NetView for z/OS version 6 release 2 modification 1
- NetView releases that are no longer supported

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

VTAM®
For Communications Server - SNA Services

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

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

Using IBM Z NetView online help

The following types of IBM Z NetView mainframe online help are available, depending on your installation and configuration:

- General help and component information
- Command help
- Message help
- Sense code information
- Recommended actions

Accessing publications online

IBM posts publications for this and all other products, as they become available and whenever they are updated, to the IBM Documentation at https://www.ibm.com/support/knowledgecenter. You can find IBM Z NetView documentation on IBM Z NetView Documentation.

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

Ordering publications

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

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

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

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

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

Accessibility

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

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

Tivoli user groups

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

Support information

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

Online
Please follow the instructions located in the support guide entry: https://www.ibm.com/support/home/pages/support-guide/?product=4429363.

Troubleshooting information
For more information about resolving problems with the IBM Z NetView product, see the *IBM Z NetView Troubleshooting Guide*. You can also discuss technical issues about the IBM Z NetView product through the NetView user group located at https://groups.io/g/NetView. This user group is for IBM Z NetView customers only, and registration is required. This forum is also monitored by interested parties within IBM who answer questions and provide guidance about the NetView product. When a problem with the code is found, you are asked to open an official case to obtain resolution.

Conventions used in this publication

This section describes the conventions that are used in this publication.

Typeface conventions

This publication uses the following typeface conventions:

Bold

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

Italic

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

Monospace

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

Operating system-dependent variables and paths

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

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

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

Syntax diagrams

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

For examples of syntax, see "Syntax examples" on page xv.

Symbols

The following symbols are used in syntax diagrams:

▶▶
Marks the beginning of the command syntax.

▶◀
Marks the end of the command syntax.

▶
Indicates that the command syntax is continued on the next line.

▶
Indicates that a statement is continued from the previous line.

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

Parameters

The following types of parameters are used in syntax diagrams:

Required
Required parameters are shown on the main path.

Optional
Optional parameters are shown below the main path.

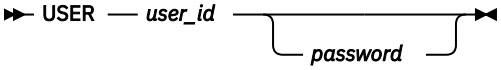
Default
Default parameters are shown above the main path. In parameter descriptions, default parameters are underlined.

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

When you issue a command, spaces are required between the parameters unless a different separator, such as a comma, is specified in the syntax.

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

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



Punctuation and parentheses

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

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

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

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

```
COMMAND_NAME opt_variable_1,,opt_variable_3
```

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

Abbreviations

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

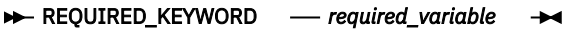
Syntax examples

The following examples show the different uses of syntax elements:

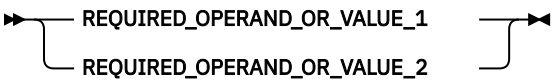
- “Required syntax elements” on page xv
- “Optional syntax elements” on page xv
- “Default keywords and values” on page xvi
- “Multiple operands or values” on page xvi
- “Syntax that is longer than one line” on page xvi
- “Syntax fragments” on page xvi

Required syntax elements

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

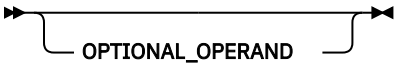


A required choice (two or more items) is shown in a vertical stack on the main path. The items are shown in alphanumeric order.

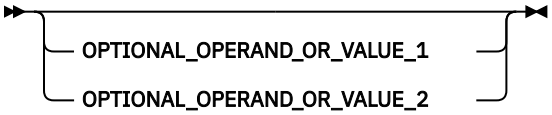


Optional syntax elements

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



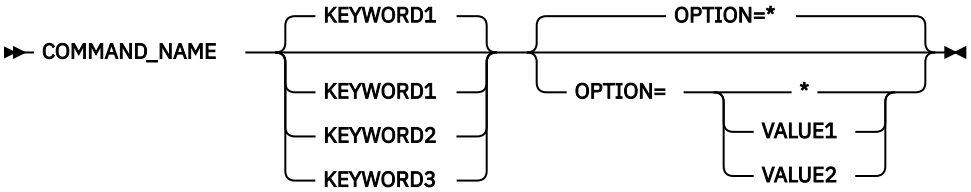
A required choice (two or more items) is shown in a vertical stack below the main path. The items are shown in alphanumeric order.



Default keywords and values

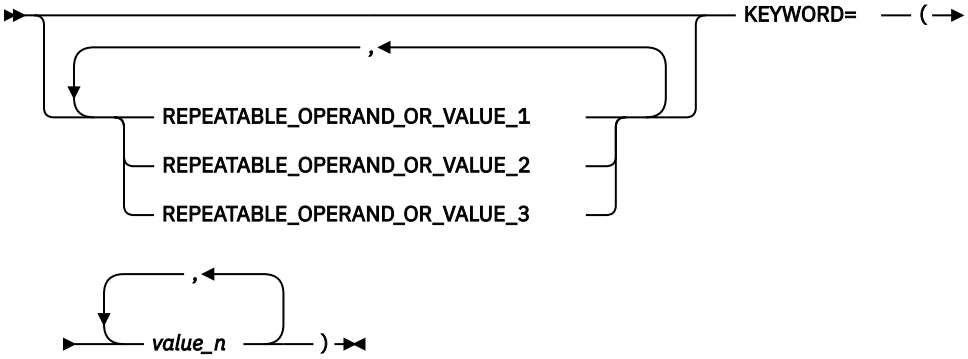
Default keywords and values are shown above the main syntax line in one of the following ways:

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



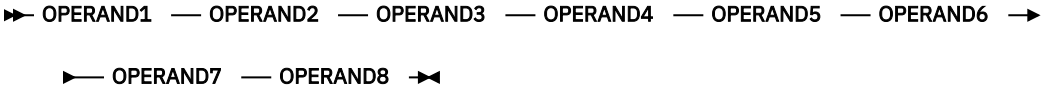
Multiple operands or values

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



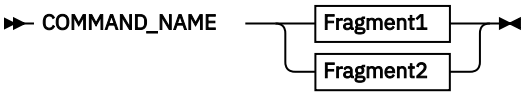
Syntax that is longer than one line

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



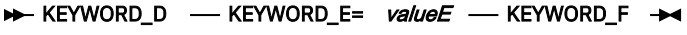
Syntax fragments

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



Fragment1

Fragment2



Chapter 1. Understanding the NetView Program-to-Program Interface

The program-to-program interface (PPI) runs as part of the IBM Z NetView (NetView) subsystem address space. When an application calls the program-to-program interface in MVS, the request is performed synchronously.

This chapter describes the following information:

- The functions of the program-to-program interface
- How applications pass requests to the program-to-program interface

This chapter does not describe how to build network management vector transport (NMVT) or control point management service unit (CP-MSU) vectors.

Reference: Application programmers need to be familiar with the SNA requirements in the following publications:

- *Systems Network Architecture Formats*
- *SNA/Management Services Alert Implementation Guide*

Note: Before you use the program-to-program interface, see Chapter 10, “Programming Techniques,” on page 71.

How the Interface Works

The program-to-program (PPI) interface enables application programs to send NMVT or CP-MSU formatted alerts to the NetView program and enables application programs to send data buffers to, or receive data buffers from, other application programs that are running in the same host as the NetView program. An application program can send, receive, or do both.

The NetView program receives NMVT or CP-MSU formatted alerts and processes them in the order of first in, first out. The NMVT or CP-MSU formatted alerts are processed by the NetView hardware monitor, which provides functions such as:

- Filtering alerts
- Displaying alerts on the Alerts-Dynamic panel
- Logging the alerts in the hardware monitor database
- Forwarding the alerts to a focal point NetView program

Alerts and resolution vectors can also be processed by the NetView automation table.

Figure 1 on page 2 shows examples of the following operations of the PPI:

- Program A sending an NMVT or CP-MSU formatted alert to the NetView program
- The NetView program receiving an NMVT or CP-MSU formatted alert from its buffer queue
- Programs B and W sending data buffers to Program Z
- Program Z receiving a data buffer from its buffer queue

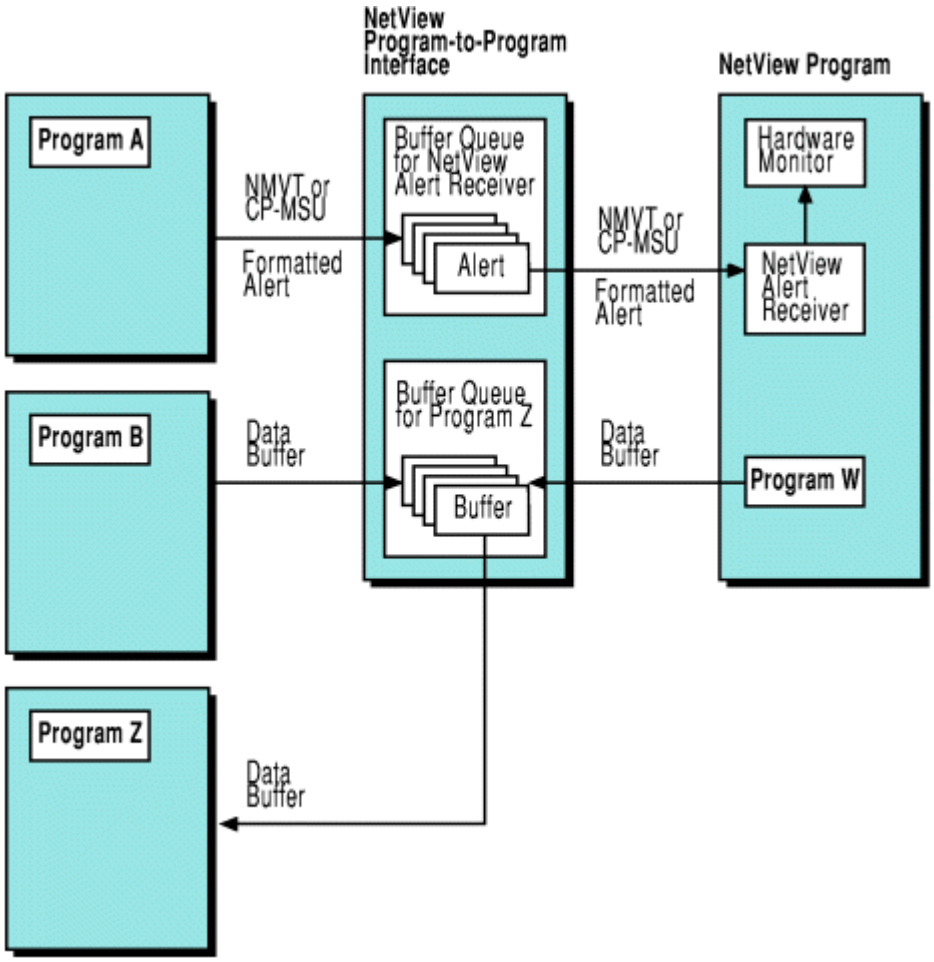


Figure 1. Example of the Program-to-Program Interface

Processing Requests

The PPI performs basic tasks called ***requests***. Each program you write can contain a series of requests. The PPI processes each request and generates a return code to indicate the status of the request.

Your program uses a request parameter buffer (RPB) to send a request to the PPI which uses the same RPB to send data back to your program. See "Building the Request Buffer" on page 13 for more information about buffer creation.

The request types available for use with the PPI for MVS are listed in Table 1 on page 2.

Table 1. Program-to-Program Interface Request Types for MVS

Request Type	Description
1	Query the status of the PPI.
2	Query the status of a receiver program.
3	Obtain the address space control block (ASCB) and task control block (TCB) addresses of the receiver.
4	Define and initialize a receiver.
9	Deactivate a receiver.
10	Delete a receiver.
12	Send an NMVT or CP-MSU formatted alert to the NetView program.

Table 1. Program-to-Program Interface Request Types for MVS (continued)	
Request Type	**Description**
14	Send a data buffer to a receiver program synchronously.
22	Receive a data buffer from the buffer queue.
23	Purge a data buffer from the buffer queue.
24	Wait for the receive or connect event control block (ECB) to be posted by the PPI.

Creating Buffer Queues

Each receiver program, including the NetView alert receiver (NETVALRT), has a *buffer queue* for temporarily storing incoming data buffers. These buffer queues reside in the PPI. A sender program sends a data buffer to a receiver buffer queue, and the receiver program retrieves the data buffer from the buffer queue.

When you define a program as a receiver (see "Request Type 4: Define and Initialize a Receiver " on page 21), you also define the ***buffer queue limit***. The buffer queue limit is the maximum number of outstanding buffers that can be stored in the receiver buffer queue. When the receiver buffer queue is full, sender programs receive a return code of 35 when they attempt to send a buffer to the receiver buffer queue.

Buffers are also queued for a receiver that becomes inactive. For example, if the NetView alert receiver task, CNMCALRT, becomes inactive, its incoming buffers are stored until CNMCALRT becomes active again.

The PPI allocates storage for a data buffer as the data buffer arrives in the buffer queue.

Sending an NMVT or CP-MSU Formatted Alert

Each program you write contains one or more requests. For example, Figure 2 on page 3 shows Program A sending an NMVT or CP-MSU formatted alert to the NetView program. Program A is using:

- Request type 1 to query the PPI status. This is an optional request.
- Request type 12 to send the NMVT or CP-MSU formatted alert.

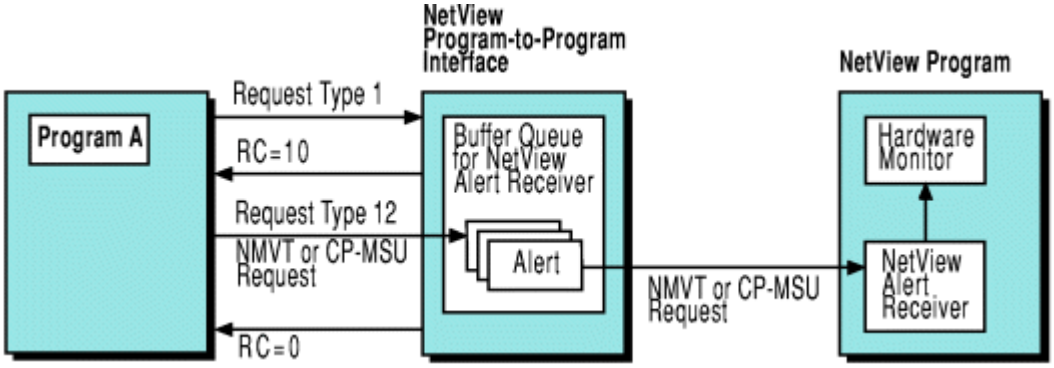


Figure 2. Overview of Sending an NMVT or CP-MSU Formatted Alert

The NetView distribution tape provides examples of programs that send an NMVT or CP-MSU formatted alert. See sample CNMS4287 for assembler, sample CNMS4257 for C, and sample CNMS4227 for PL/I.

Sending a Data Buffer Synchronously

Figure 3 on page 4 shows Program B synchronously sending a data buffer to the buffer queue for Program Z. Program B is using:

- Request type 1 to query the PPI status. This is an optional request.

- Request type 2 to query the status of Program Z. This is an optional request.
- Request type 14 to send the data buffer.

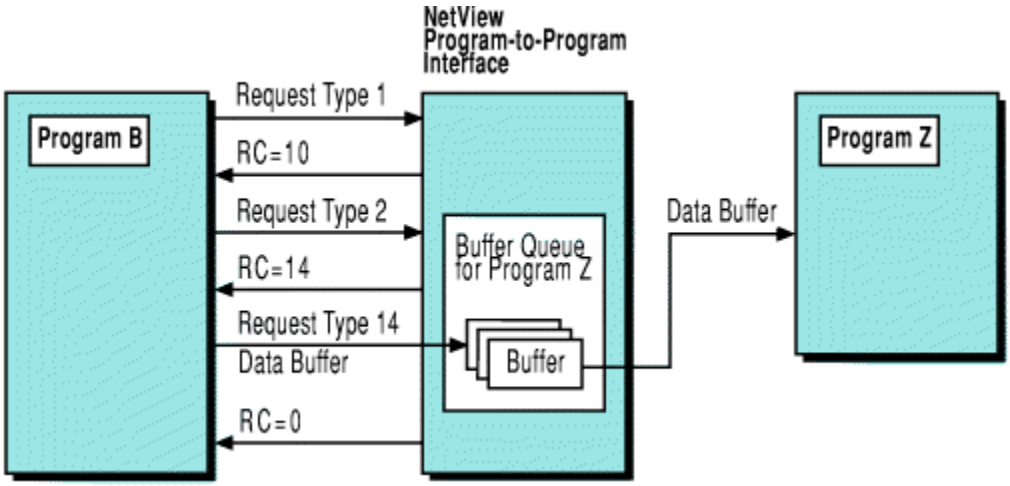


Figure 3. Overview of Sending a Data Buffer Synchronously

The NetView distribution tape provides examples of programs that send a data buffer. See sample CNMS4288 for assembler and sample CNMS4228 for PL/I.

Receiving a Data Buffer Synchronously

Figure 4 on page 5 shows Program Z receiving a data buffer from the Program Z buffer queue. Program Z is using:

- Request type 1 to query the PPI status. This is an optional request.
- Request type 3 to obtain the ASCB and TCB addresses. This is an optional request.
- Request type 4 to define itself as a receiver. The PPI returns the receiver ECB address.
- Request type 22 to retrieve a data buffer from the buffer queue.
- Request type 9 to deactivate the receiver.

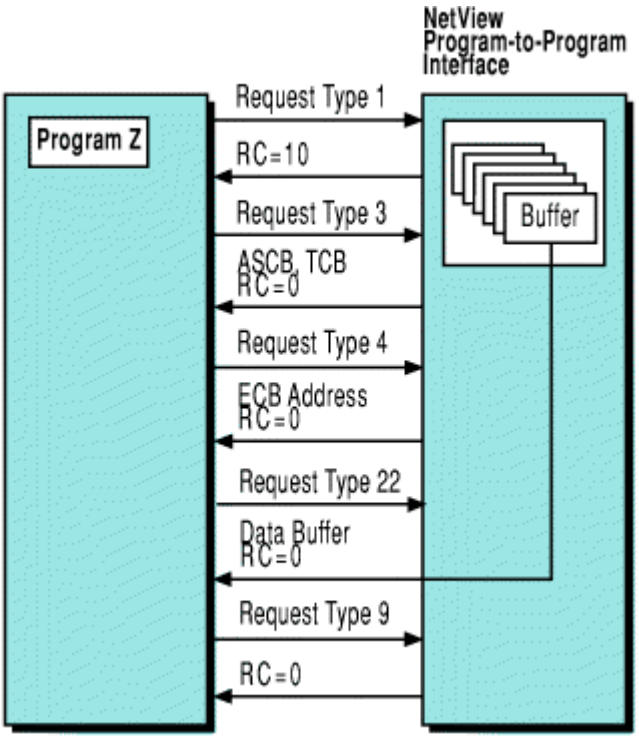


Figure 4. Overview of Receiving a Data Buffer Synchronously

The NetView distribution tape provides examples of programs, other than REXX, that receive a data buffer. See sample CNMS4289 for assembler and sample CNMS4229 for PL/I. For a REXX example, see "REXX Programming Examples" on page 80.

How the Interface Works with Applications

This section applies to applications written in languages other than REXX. For REXX programs, you can use DSIPHONE to access the program-to-program interface. For information on coding DSIPHONE, see Chapter 3, "Using REXX to Send Requests," on page 33.

Each program you write must use the CALL statement to pass your requests to the CNMCNETV module in the NetView subsystem. Control is returned to your program immediately following the CALL statement. The CALL statement runs the CNMCNETV module and passes the request parameter buffer.

The CNMCNETV module runs in 31-bit addressing mode only for MVS. Set the high-order byte correctly for all addresses so that they are valid for 31-bit addressing mode. Pass the addresses in registers 1, 13, 14, and 15.

You must either link edit the CNMCNETV module from the CNMLINK data set into your application or load it from the SCNMLPA1 or LPALIB data set at runtime. The advantage of loading the CNMCNETV module at runtime is that you are assured of using the latest copy rather than having to relink your application each time CNMCNETV changes due to the application of maintenance.

High-Level Language Programs

The following example shows the CALL statement for high-level languages, such as PL/I and C, where *rpb* is the name of the request parameter buffer.

```
CALL CNMCNETV (rpb)
```

For high-level language programs, you can link-edit the CNMCNETV module during the link-edit step.

Assembler Programs

You can code the CALL statement in assembler in either of two ways:

- You can use the module name.

  The following example shows using CNMCNETV in the CALL statement in assembler, where *rpb* is the name of the request parameter buffer.

  ```
      CALL CNMCNETV, (rpb)
  ```

- You can use the address of the module.

  In the following example, *rpb* is the name of the request parameter buffer, and *nn* is the register that contains the address of the CNMCNETV module.

  ```
      CALL (nn), (rpb)
  ```

For assembler programs, you can use the LOAD macro to load the CNMCNETV module into memory and then branch and link to the module. You can also link-edit the CNMCNETV module. For more information about coding the CALL statement in assembler, refer to the z/OS library.

Register Conventions

Programs written in high-level programming languages, such as C and PL/I, and programs written in assembler can use the CALL interface if they support the following register conventions expected by the CNMCNETV module:

Register 1
Points to a memory location that contains the address of the request parameter buffer.

Register 13
Contains the address for the calling program's 72-byte save area.

Register 14
Contains the return address for the calling program.

Register 15
Contains the entry address for the CNMCNETV module.

Program Placement

NetView application programs and other user programs running in any address space can pass NMVT or CP-MSU formatted alerts to the NetView hardware monitor for processing.

Additionally, a user program can send data buffers to (or receive data buffers from) another user program running in any address space.

Note: Programs sending NMVT or CP-MSU formatted alerts to the NetView program and sending and receiving data buffers need to reside in the host system that is processing the NetView program.

Sending Commands and Messages to the NetView Program

Programs (clients) can use the following methods to send commands and messages to the NetView program:

- CMDSERV

  The CMDSERV interface is a command server designed to run on a NetView autotask. The autotask accepts commands from an environment that has established an identity to the security product (SAF) and is defined there as a NetView user.

- APSERV

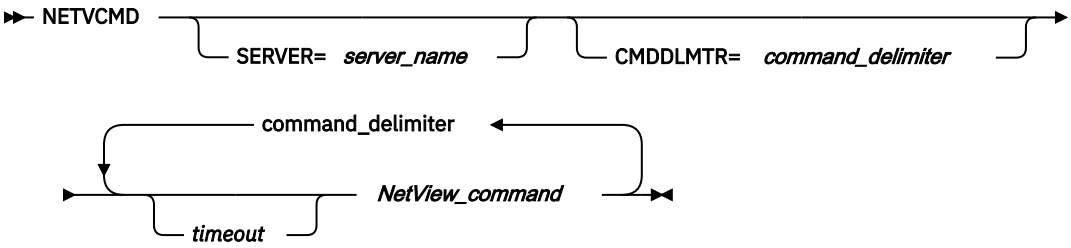
The APSERV interface is a command server that runs on a NetView autotask or on a virtual OST (VOST). APSERV accepts commands or messages from APF authorized programs only. Unlike CMDSERV, APSERV does not require the client to register with a security product; instead, the client specifies the user ID under which the command is to be authority checked.

Using the CMDSERV Interface

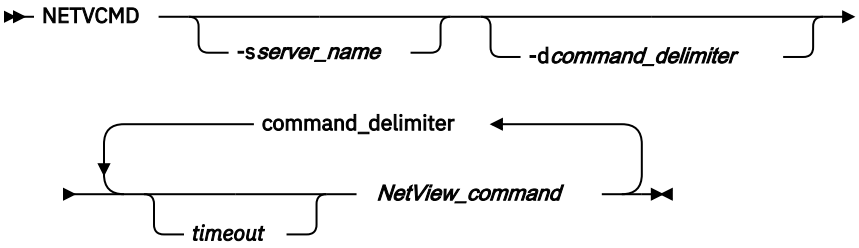
The CMDSERV interface receives NetView commands that are passed through the NetView PPI. CMDSERV processes the command and returns the results through the PPI to the originating program. See the *IBM Z NetView Command Reference Volume 1 (A-N)* or the NetView online help for more information about using the CMDSERV command.

One method for using the interface that the CMDSERV command sets up is to use the NETVCMD (CNMS8029) utility. The following formats are used:

For TSO or REXX batch usage:



For UNIX Usage:



command_delimiter
Specifies the delimiter used to separate NetView commands when more than one NetView command is entered, for example:

```
NETVCMD CMDDLMTR=/ LIST ''/ LIST OPER1
```

The default delimiter is the semicolon (;).

When running under a UNIX shell, you might need to use an escape character before the delimiter or other characters. For example:

```
NETVCMD LIST \'\'\; LIST OPER1
```

Or the entire NETVCMD parameter list can be enclosed in quotation marks as shown in the following example:

```
NETVCMD "LIST ''; LIST OPER1"
```

NetView_command
Indicates the NetView command to be issued.

server_name
Indicates the name of the server. If specified, this value must be the same name used, or the default value, on the name parameter of the CMDSERV command issued under the NetView program. If the server name is not specified, the default value is DSICMDSV.

timeout
Specifies an optional timeout value in seconds to be applied to the command that it precedes. The default is established by the NetView CCDEF command because the command ultimately runs in a pipe with the CORRCMD stage.

Example: Issuing NetView Commands from TSO and UNIX

To issue a NetView command from a TSO or UNIX command line, the PPI must be active and your TSO or UNIX user ID must be defined in your security product and as a NetView user. The NetView user ID must be authorized to issue the NetView commands sent to the NetView program. Define the user as a NetView operator; for more information about defining NetView operators, see the *IBM Z NetView Security Reference*.

In this example, a REXX CLIST named NETVCMD is used to issue the NetView LIST command. NETVCMD is a sample that is supplied with the NetView product and can be found in CNMS8029. When the NetView command is issued, *server_name* must either be the same as specified on CMDSERV NAME=*server_PPI_name* or default to DSICMDSV.

From TSO, enter the following command:

```
NETVCMD SERVER=server_name LIST DEFAULTS
```

From UNIX, enter the following command:

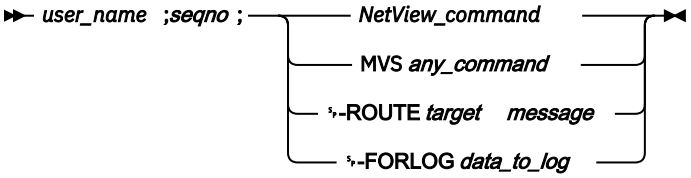
```
NETVCMD -sserver_name LIST DEFAULTS
```

Using the APSERV Interface

The APSERV (CNMEAPCS) command sets up an interface that authorized programs (clients) can use in the following ways:

- Run NetView or MVS commands under the authority of a specified NetView operator
- Place a message in the netlog
- Route a message to a specified operator or to the authorized receiver

To use the APSERV interface, an authorized program sends a buffer over the PPI to the specified receiver. The data sent over the PPI has the following format:



Note:

1. Non-printable characters (00x - 3Fx) cannot be used with the APSERV interface.
2. If REXX is used, the authorized program can use the DSIPHONE subroutine to transmit the data across the PPI. See Chapter 3, “Using REXX to Send Requests,” on page 33.

user_name
The *user_name* variable can be one of the following names.

- A NetView function name beginning with the question mark character (?).
- A NetView Tivoli Enterprise Portal user ID as defined with the NACMD.OPID.*TEPLogonid* statement in the CNMSTYLE member. A NetView Tivoli Enterprise Portal user ID can be 1 - 10 characters in length and can contain alphanumeric and the @, #, and $ characters.

  Under certain conditions when *user_name* is a NetView Tivoli Enterprise Portal user ID, the APSERV interface inserts records into NetView Tivoli Enterprise Portal workspaces; see “Usage Notes” on page 9.

- A NetView operator ID. If the specified name is both a NetView Tivoli Enterprise Portal user ID and a NetView operator ID, it is processed as a NetView Tivoli Enterprise Portal user ID. Under certain conditions when *user_name* is a NetView operator ID, the APSERV interface inserts records into NetView Tivoli Enterprise Portal workspaces; see "Usage Notes" on page 9.

When *user_name* is a NetView operator ID, the APSERV interface checks whether the operator is logged on and, if not, causes an autotask with that ID to be started. Currently, the NetView program does not recognize such an autotask as *attended*. Users might want to adjust their IDLEOFF settings. Commands are always queued at a low priority.

seqno
The *seqno* variable helps identify a particular transaction. The *seqno* can be 1 - 8 characters in length. It can contain alphanumeric and the @, #, and $ characters.

NetView_command
The *NetView_command* variable can be any regular NetView line mode command. The APSERV interface processes commands subject to separate authority checking. Do not send full-screen commands.

The first character of the command cannot be a dash or minus sign ('60'X). Instead, use the PIPE CORRCMD stage.

MVS *any_command*
The *any_command* variable is the command to be submitted to the z/OS operating system under the authority of the user name specified by the *user_name* variable.

-ROUTE *target message*
The *target* variable is a valid argument for the PIPE ROUTE stage command. At least one blank must follow the *target* variable. The *message* variable can be any data except non-printable characters (00x - 3Fx). All blanks after the *target* variable are discarded, and the remaining data is sent to the specified target.

-FORLOG *data_to_log*
When -FORLOG is specified, the *user_name* variable is ignored. The *data_to_log* variable can be any data except non-printable characters (00x - 3Fx), up to 255 bytes, to be logged by the task. The task is indicated by the *user_name* variable according to the DEFAULTS or OVERRIDE command settings in effect for that task.

Note: The data is logged and is submitted to user exit 4, but it is not submitted to other exits, message automation, ASSIGN processing, or trap matching.

Usage Notes

- For each NetView or MVS command received, the APSERV interface logs a BNH806I message. The message origin (domain field) for this message contains the SAF user name of the client sending the command. For the command echo, the message origin lists the PPI name for the client.
- Messages generated by the APSERV interface are logged as usual for the NetView task where they occur. Use the OVERRIDE NETLOG, SYSLOG, or HCYLOG options as needed for each task. Command echoes and command responses are logged at the target task specified by the client. Command response messages are submitted to the automation table before being logged. The BNH806I and error messages are logged at the autotask or VOST where APSERV is running. See the OVERRIDE command for more information about the options.
- The APSERV command is a long-running command. To end the command and close the PPI receive, issue STOP FORCE=*taskname* specifying the task name of the autotask. Stop a VOST with the DETACH command.
- If the APSERV command runs on an autotask (not a VOST), then ensure no other function is assigned to that autotask. The APSERV command continuously waits for input records.
- If the following conditions are met, the APSERV interface inserts records into the NetView Tivoli Enterprise Portal workspaces:
  - The TEMA tower is enabled.

  - The Z NetView enterprise management agent is active.
  - One of the following conditions is true:
    - *user_name* is a Tivoli Enterprise Portal user ID.
    - *user_name* is a NetView operator ID, and the -TEMA option was specified on the APSERV command; see the online help for more information about the APSERV command.

  The following records are inserted:

  - The command echo and all correlated command responses received within 30 minutes of issuing the command are inserted into the NetView Command Response workspace. Before insertion, these records are submitted to user exits and message automation, either of which can alter or suppress messages.
  - The BNH806I audit record is inserted into the NetView Audit Log workspace before being submitted to user exits and message automation.
- For more information about handling clients that begin each input command with a fixed string, see *APSERV.PREFIX* in the *IBM Z NetView Administration Reference*.

Chapter 2. Using High-Level Languages and Assembler to Send Requests

This chapter contains instructions for enabling the PPI. This chapter also describes the request parameter buffer fields and return codes associated with each request type.

Enabling the Interface for MVS

For the NetView program to receive NMVT or CP-MSU formatted alerts, enable the PPI in the following way:

1. Initialize the NetView subsystem address space. Refer to *IBM Z NetView Installation: Getting Started* for more information.
2. Ensure that the NetView subsystem address space has a specified region size large enough to hold the user data buffers that might be queued or stored in the subsystem address space. The required storage depends on how many receivers exist in the NetView subsystem address space. To estimate the required storage:
   a. Determine the storage required for each receiver, using the following formula:

      ```
      Average buffer size (bytes)  X  buffer queue limit (number of buffers)
      ```

   b. Add the storage requirements for all the receivers to get an estimate of the total storage required, in bytes.

   Some of the NetView functions using the PPI can require larger buffer queue lengths for the NetView and VTAM PPI receivers. Refer to the storage estimating information in the *IBM Z NetView Tuning Guide* to determine storage requirements.
3. If the NetView program is to start the NetView subsystem address space, make sure that the SSI.PPI statement is set to YES in your CNMSTYLE definitions. If you start the subsystem address space directly, make sure that the PPIOPT statement is set to PPI (this is the default value) in your CNMSTYLE definitions. Note that if you start multiple subsystem address spaces in the same LPAR, only one needs to support the PPI.
4. Review the NetView startup procedure before you start the PPI to determine how the NetView subsystem address space is used. For more information about the startup procedure, refer to the *IBM Z NetView Installation: Getting Started*. See the CNMSJ010 sample on the NetView distribution tape for an example of a NetView subsystem address space procedure.
5. Ensure that the PPI module resides in an MVS authorized program facility (APF) authorized library if the PPI module is to be processed as an APF-authorized program.
6. Enter the TRACEPPI command, if you want to enable the PPI trace facility. For more information about the PPI trace facility, see Chapter 12, “Using the Trace Facility,” on page 91. Refer to the NetView online help for information about the TRACEPPI command.
7. Activate and enable the generalized trace facility (GTF) for PPI. For more information, see Chapter 12, “Using the Trace Facility,” on page 91.

Note: Unpredictable results can occur, including system abends and lost data if you stop the NetView subsystem address space before you stop all applications that are using the address space.

Receiving Alerts

If the alert receiver task is started and the NetView subsystem or the DSICRTR task is inactive, the alert receiver task issues messages CNM563I and DSI295I, and waits until both the NetView subsystem and the DSICRTR task are active. To enable the NetView program to receive NMVT or CP-MSU formatted alerts using the PPI, complete the following steps:

1. Ensure that the PPI is fully initialized. See “Enabling the Interface for MVS” on page 11.
2. Ensure that the NetView communication network management task (DSICRTR) is active.
   a. Issue the following command from the NetView command line to find the status of the DSICRTR task:

      ```
      LIST DSICRTR
      ```

   b. Skip to step “3” on page 12 if the status is ACTIVE.
   c. Issue the following command if the status is INACTIVE:

      ```
      START TASK=DSICRTR
      ```

      Note: If multiple NetView programs are active, start the DSICRTR task only in the NetView program that performs problem determination (the network management NetView program).
3. Ensure that the NetView alert receiver task (CNMCALRT) is defined to the NetView program and is started when the NetView program is started. If you are not using the PPI to send NMVT or CP-MSU formatted alerts, defining or starting the CNMCALRT task is not necessary.
   a. Issue the following command from the NetView command line to find the status of the CNMCALRT task:

      ```
      LIST CNMCALRT
      ```

   b. If the status is ACTIVE, the NetView program can receive NMVT or CP-MSU formatted alerts and no further action is needed.
   c. Issue the following command if the status is INACTIVE and you are using the PPI to send NMVT or CP-MSU formatted alerts to the NetView program:

      ```
      START TASK=CNMCALRT
      ```

Routing Alerts to Multiple Receivers

To route an alert to more than one NetView system through the PPI, use the AlertRcvName statement in the CNMSTUSR or C*xx*STGEN member to set a different PPI alert receiver name on each NetView program. The default is NETVALRT. For information about changing CNMSTYLE statements, see *IBM Z NetView Installation: Getting Started*.

Use a request type 14 to send a data buffer to multiple alert receivers, specifying the proper receiver ID in the RPB. Figure 5 on page 13 shows how alerts are routed to multiple receivers. If you want to send an alert to multiple receivers, issue the send to each of the receivers.

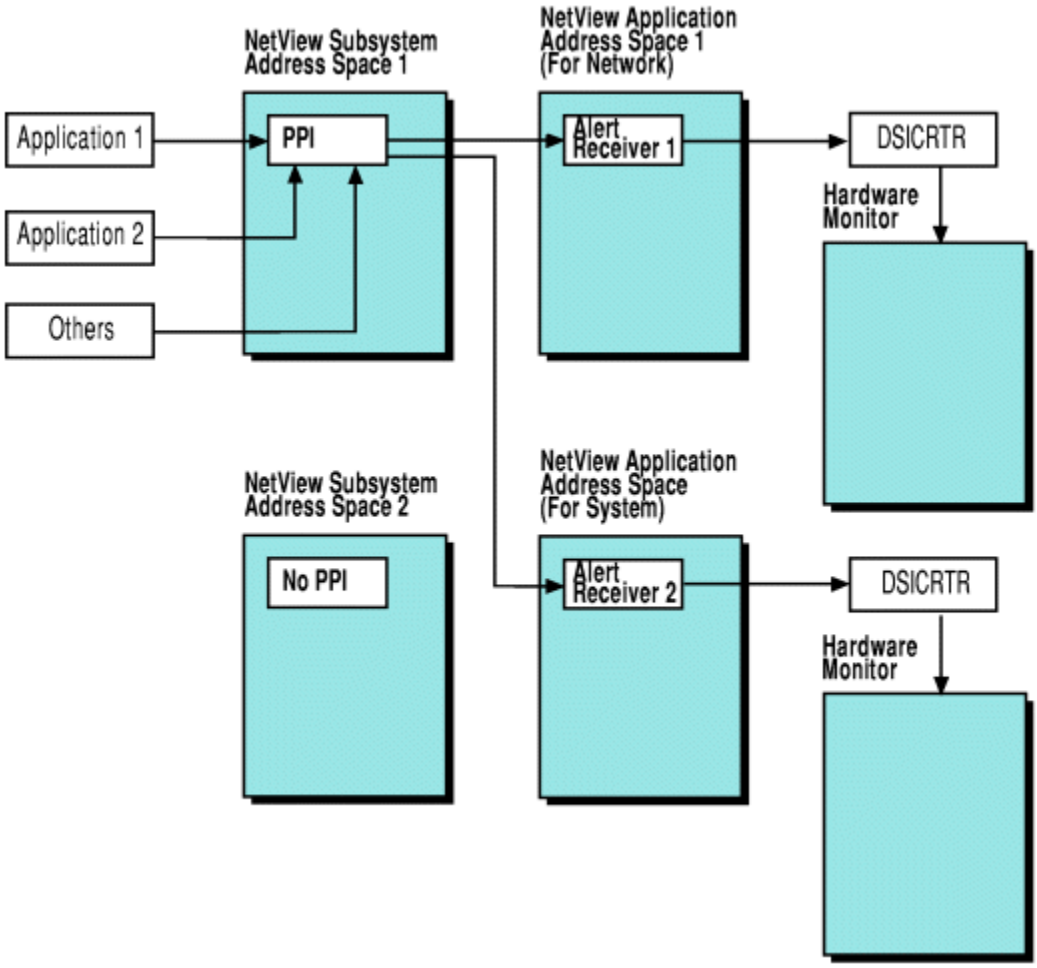


Figure 5. Sending Alerts to Multiple Receivers through the PPI

Building the Request Buffer

The request parameter buffer (RPB) is a 56-byte parameter list that you build for each request. Your program uses an RPB to send a request to the PPI, and the PPI uses the same RPB to return data to your program.

Using the RPB

The RPB fully describes your request by specifying items such as the request type and the receiver or sender identification. For example, if you want to send a data buffer to another user program, the RPB for your request type 14 contains the receiving program's identification, the sending program's identification, the length of the buffer you are sending, and the address to which the data buffer is sent.

In addition, the PPI uses fields in the RPB to return data to your program. For example, when you use a request type 4 to define your program as a receiver, the PPI returns the address of the receiver event control block (ECB) in the RPB. All return codes are also returned in the RPB.

Each request needs a RPB. For each RPB you build, use only those fields that apply to your request type. “Choosing the Request Type” on page 18 describes which fields are required for each request type.

Fields in the RPB

Table 2 on page 14 lists the fields in the RPB. The Data Field names used in this table are for descriptive purposes only. You are not required to use these names in your RPB. The DTR Field name corresponds to the DSIDTR macro shipped as part of the NetView macro set.

Macro DSIDTR includes two labels that can be used to determine the lengths of either the 56-byte structure or the 96-byte structure (total length of the DTR including those fields that occur following offset 56; see Table 2 on page 14 to identify these fields). User application code not utilizing those additional fields (that is, those that occur following offset 56) can use DTREND to calculate the length of the 56 byte RPB; if you use those additional fields, use DTREND1 to calculate total RPB length of 96. Fields following offset 56 must be zeroed if not being initialized to valid pointer values to ensure NetView code does not try to access an address that is not valid. User code that employs DSIGET with the default option CLEAR=YES need not specifically zero these fields, however code that requires a GETMAIN for storage must be cautious of residual data in these fields.

Table 2. Request Parameter Buffer Fields

Bytes	Data Field	DTR Field	Description
0–3	RPB-LEN	DTRLEN	Length of the request parameter buffer; this field must be set to 56 the length of the fields being used.
4, 5	TYPE	DTRREQT	Request type; a 2-byte integer value, for example, 22. The request types are described under "Processing Requests" on page 2.
6, 7	RECOPT	DTRRECOP	Recovery option indicator, which can have one of the following values: **0** No recovery is requested. **1** ESTAE recovery is requested. The ESTAE program isolates the user from abends and protects the user program from abnormally ending. If an error occurs while a user program is using the PPI, the ESTAE routine traps the error and sets a return code that it passes to the user program, preventing an abend. These return codes are described in Appendix B, "Program-to-Program Interface Return Codes," on page 103. ESTAE recovery is not available if the user program is running in cross-memory mode. For more information about the ESTAE routine, refer to the z/OS library.
8–11	RETCODE	DTRRETC	4-byte processing return code returned by the PPI on every request type.
12–15	WORK-ADR	DTRWKPTR	Work storage address required by the NetView service module, CNMCNETV. The work storage must be 128 bytes for MVS.
16–23	SENDER-ID	DTRSDID	Sender identification; 8-character identifier of the sender program. The ID can contain alphabetic characters A - Z, numeric characters 0 - 9, and the following special characters: dollar sign ($), percent sign (%), ampersand (&), at sign (@), and number sign (#). If the ID is not 8 characters long, it must be left-justified and padded with blanks on the right side.

Table 2. Request Parameter Buffer Fields (continued)

Bytes	Data Field	DTR Field	Description
16–19	ASCB-ADR	DTRASCB	Address space control block address; this is the ASCB address of the receiver program. The PPI returns this value on a request type 3. Some programming languages do not provide the mapping facilities for a receiver program to determine what to specify in the ASCB-ADR field. Therefore, the receiver program uses request type 3 to determine this field. If the address space with the specified ASCB-ADR ends, the receiver status is set to inactive.
20–23	ECB-ADR	DTRECB	Event control block address. The PPI returns this value on request types 4 and 22 when the buffer queue of the receiver has no buffers. The PPI posts the receiver ECB when a data buffer is received in the receiver buffer queue. Your program can use a WAIT macro or a request type 24 to wait for the PPI to post this ECB. The post code is the lower 3 bytes of the ECB. When the NetView subsystem ends, the ECB is posted with a post code of 99.
20–23	BUFFQ-L	DTRBQL	Buffer queue limit; the maximum number of outstanding data buffers that can be accepted for a receiver. This limit is defined in a request type 4 when the receiver is activated. The limit can be changed by a request type 9 when the receiver is deactivated, or by another request type 4 for that receiver. A sender can send data buffers to an active or inactive receiver as long as the receiver buffer queue is not full. When the receiver buffer queue is full, a sender program receives a return code of 35 and the data buffer is not accepted.
24–31	RECEIVER-ID	DTRRVID	An 8-character identifier of the receiver program. The ID can contain alphabetic characters A - Z, numeric characters 0 - 9, and the following special characters: dollar sign ($), percent sign (%), ampersand (&), at sign (@), and number sign (#). If the ID is not 8 characters long, it must be left-justified and padded with blanks on the right side. The NETV*xxxx* IDs are reserved for products related to the NetView program. NETVALRT is the ID of the NetView alert receiver task.

Table 2. Request Parameter Buffer Fields (continued)

Bytes	Data Field	DTR Field	Description
32–35	BUFF-LEN	DTRUBL	Buffer length; length of a data buffer or NMVT or CP-MSU formatted alert. For a request type 12 (sending an NMVT or CP-MSU formatted alert), sender programs use this field to specify the length of the NMVT or CP-MSU formatted alert. For a request type 14 (sending a data buffer), sender programs use this field to specify the length of the buffer. For a request type 22 (receiving an incoming data buffer), receiver programs use this field to specify the length of the buffer into which the incoming data buffer is to be copied. If the value specified is not large enough, the return code from the request type 22 is 31. If the request type 22 is successful, the PPI uses this field to return the actual length of the incoming data buffer.
32, 33	AUTH-IND	DTRAUTH	Authorization indicator; specifies that a receiver accepts data buffers only from an APF-authorized receiver. Receiver programs use this field in a request type 4 (initializing the receiver). If this indicator is set to 1, the receiver program is defined as authorized, and a sender program must be APF-authorized to send data buffers to this receiver.
34 (bit 0)	BUFFER-Q-FLAG	DTRBQFL	Receiver's buffer queue flag. Set on request type 2 to indicate whether space is available on the receiver's buffer queue. **0** No space on the queue **1** Space available on the queue
36–39	BUFF-ADR	DTRUBPTR	Buffer address; data buffers are copied to (or from) this address. Both sender and receiver programs use this field.
36–39	TCB-ADR	DTRTCB	Task control block address; the PPI returns this value on a request type 3. Some programming languages do not provide the mapping facilities for a receiver program to determine what to specify in the TCB-ADR field. Therefore, the receiver program uses request type 3 to determine this field. If the TCB with specified TCB-ADR ends, the receiver status is set to inactive.
40–45			Not used for MVS.
46, 47	CKBTS	DTRCKBTS	Request indicators.

Table 2. Request Parameter Buffer Fields (continued)

Bytes	Data Field	DTR Field	Description
46 (bit 0)	EX-ACT	DTREACT	Exclusive check for active receiver. The receiver program uses this field in request type 4 (see "Request Type 4: Define and Initialize a Receiver " on page 21 for more information). If this bit is on and the receiver program is already active, a value of 16 is returned in DTRRETC.
46 (bit 1)	VER-CHECK	DTRVRCHK	PPI version check. Your program uses this field with request type 1 to perform a PPI version check. The PPI returns a value in the DTRVERSN field indicating the functional level of the PPI.
46 (bit 4)	MATCH-SENDER-ID	DTRRCVNM	If DTRSDNAM is not zero, receive or purge buffers only from the sender ID indicated by the DTRSDNAM token.
46 (bit 5)	MATCH-ASID	DTRRCVAT	If DTRSDAST is not zero, receive or purge buffers only from the address space indicated by the DTRSDAST token.
46 (bit 6)	MATCH-TCB	DTRRCVTT	If DTRSDTT is not zero, receive or purge buffers only from the task indicated by the DTRSDTT token.
48–51	PPI-VERSION	DTRVERSN	PPI version. When the DTRVRCHK bit is set on for request type 1, the PPI returns the functional level in this field. The levels are: **0** NetView V2R3 or earlier releases **1** NetView V2R4 to TME 10 NetView for OS/390® V1R1 **2** Tivoli NetView for OS/390 V1R3 or later releases
52–55	SAF-ADR	DTRSAFWK	Address of a 1024-byte work area required by a send request which needs to communicate its SAF ID to the receiver.
		DTREND	Label used to determine length of 56-byte structure.
56–63	SENDER-SAF-ID	DTRSAFID	Sender's SAF ID, for senders which supply DTRSAFWK - filled in by PPI on a receive request. Binary zeroes are returned if no SAF ID.
64–71	SENDER-NAME/ID	DTRSDNAM	Sender's PPI name, same as DTRSDID - filled in by PPI on a receive request. Can be set by user prior to a PPI receive.
72–79	ASID–TOKEN	DTRSDAST	Address space token associated with the sender - filled in by PPI on a receive request. Can be set by user prior to a PPI receive.

Table 2. Request Parameter Buffer Fields (continued)			
Bytes	**Data Field**	**DTR Field**	**Description**
80–95	TCB–TOKEN	DTRSDTT	Task token associated with the sender - filled in by PPI on a receive request. Can be set by user prior to a PPI receive.
		DTREND1	End label of total 96-byte structure.

Choosing the Request Type

The request types are the building blocks of your programs. For each program you write, perform the following steps:

1. Build the request parameter buffer (RPB) for each request you want to issue.
2. Build the NMVT or CP-MSU formatted alert or data buffers as appropriate.
3. Issue the CALL statement to pass each RPB and its associated NMVT or CP-MSU formatted alert or data buffer to the CNMCNETV module.
4. Check the return code field in the RPB.

For a complete list of return codes generated by the PPI and their hexadecimal equivalents, see Appendix B, “Program-to-Program Interface Return Codes,” on page 103. Return code 20 is not a valid request type.

The remainder of this section is a description of each request type for MVS, including the:

- RPB fields you specify in your program
- RPB fields that are returned by the PPI
- Return codes from the PPI

Request Type 1: Query the PPI Status

Request type 1 is used in both sender and receiver programs. Make the first request in any program you write a request type 1 to query the PPI status. If the PPI is not active, no requests are processed.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
12–15	WORK-ADR	DTRWKPTR
46 (bit 1)	VER-CHECK	DTRVRCHK

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
48–51	PPI-VERSION	DTRVERSN

Return codes

Return Code	Description
10	The PPI is available to process user requests.
24	The PPI is not active.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
90	A processing error has occurred.

Request Type 2: Query a Receiver's Status

Request type 2 is used in both sender and receiver programs. A request type 2 determines the status of the receiver you specify in the RPB. A receiver's status can be active, inactive, or undefined. You can send data buffers to an active or inactive receiver, as long as the receiver buffer queue is not full, but you cannot send data buffers to an undefined receiver.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
12–15	WORK-ADR	DTRWKPTR
24–31	RECEIVER-ID	DTRRVID

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
34 (bit 0)	BUFFER-Q-FLAG	DTRBQFL

Return codes

Return Code	Description
14	The receiver program is active.
15	The receiver program is inactive.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
26	The receiver program is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
40	Receiver ID is not valid.
90	A processing error has occurred.

Usage Notes

- The RECEIVER-ID for the NetView alert receiver task is NETVALRT.
- You can use a request type 2 to query the status of this task.

Request Type 3: Obtain the ASCB and TCB Addresses

Request type 3 is used in receiver programs. When the request type 3 completes successfully, the NetView program returns the addresses of the ASCB and the TCB into the fields ASCB-ADR and TCB-ADR respectively.

The NetView program uses the ASCB-ADR like a password. When you define a receiver (request type 4), you must specify the ASCB-ADR. Any subsequent request to receive a data buffer (request type 22), to deactivate the receiver (request type 9), or to reset the buffer queue limit (request type 4) must include this ASCB-ADR. If the ASCB-ADR is not correct in a subsequent request, the NetView program does not process that request.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
12–15	WORK-ADR	DTRWKPTR

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
16–19	ASCB-ADR	DTRASCB
36–39	TCB-ADR	DTRTCB

Return codes

Return Code	Description
0	The request completed successfully.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
28	An active subsystem interface address space was found, but an active PPI address space was not found.

Usage Notes

Use request type 3 if your programming language does not provide the mapping facilities to determine the ASCB and TCB addresses.

Request Type 4: Define and Initialize a Receiver

Request type 4 defines your program as a receiver and sets its status to active. Use this request type to reset the buffer queue limit.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–19	ASCB-ADR	DTRASCB
20–23	BUFFQ-L	DTRBQL
24–31	RECEIVER-ID	DTRRVID
32, 33	AUTH-IND	DTRAUTH
36–39	TCB-ADR	DTRTCB
46 (bit 0)	EX-ACT	DTREACT

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
20–23	ECB-ADR	DTRECB

Return codes

Return Code	Description
0	The request completed successfully.
16	The receiver program is already active.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
32	No NetView storage is available.
36	ESTAE recovery cannot be established as requested.
40	RECEIVER-ID is not valid.
90	A processing error has occurred.

Usage Notes

- The BUFFQ-L field specifies the maximum number of outstanding buffers that a receiver buffer queue can have in storage. You can also change the BUFFQ-L when the receiver is deactivated (request type 9).
- A change in buffer queue limit does not affect buffers already in the queue. That is, if the limit is decreased, buffers already in the queues are not lost. However, any new buffers that arrive for the receiver are rejected if any existing buffers have reached or exceeded the buffer queue limit.
- The ECB-ADR field contains the address of the receiver's 4-byte event control block (ECB). The address is returned by the PPI. The PPI posts the ECB to notify a receiver that a data buffer has arrived in the receiver buffer queue.
- Your program can use a WAIT macro to wait for the PPI to post the receiver ECB. If a WAIT macro is not available, your program can use a request type 24 to wait for the PPI to post the receiver ECB.
- The AUTH-IND field specifies that this receiver program accepts data buffers from APF-authorized programs only.
- You can adjust the buffer queue limit while the system is running by reissuing a request type 4 (define and initialize a receiver) with a different BUFFQ-L value. The receiver program, or any other program, can perform this adjustment if you specify all the RPB fields. The EX-ACT value must be zero. You receive return code 0 even though the receiver program is already active.
- When the EX-ACT field is set on, it specifies the performance of an exclusive check for an active receiver program. You receive return code 16 if the receiver program is already active.
- When the ECB post code is zero, data buffers are waiting to be processed by the receiver. When the post code is 99, the PPI is ending. The post code is located in the lower 3 bytes of the receiver ECB.
- ASCB-ADR is a required field for request type 4. ASCB-ADR must contain the address of a valid ASCB. All subsequent requests by the receiver program must include this same ASCB address. The PPI cannot verify the validity of the ASCB address. The receiver program must ensure that it submits a valid ASCB-ADR.
- If the ASCB-ADR field is 0, the receive ECB is not posted when the PPI receives a data buffer for that receiver, or when the PPI ends.

Request Type 9: Deactivate a Receiver

Request type 9 sets the receiver status to inactive. You can also reset the buffer queue limit when you deactivate the receiver.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–19	ASCB-ADR	DTRASCB
20–23	BUFFQ-L	DTRBQL
24–31	RECEIVER-ID	DTRRVID

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC

Return codes

Return Code	Description
0	The request completed successfully.
15	The receiver program is inactive.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
25	The ASCB address is not correct.
26	The receiver program is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
36	ESTAE recovery cannot be established as requested.
40	RECEIVER-ID is not valid.
90	A processing error has occurred.

Usage Notes

- Ensure that your program issues a request type 9 before it ends.
- If you do not set the buffer queue limit, it is automatically set to an unpredictable limit.
- The ASCB-ADR for this request must be the same as that specified for request type 4 for this receiver.

Request Type 10: Delete a Receiver

Request type 10 deletes an active receiver from the program-to-program interface and deletes the receiver's buffers from the buffer queue.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–19	ASCB-ADR	DTRASCB
24–31	RECEIVER-ID	DTRRVID

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC

Return codes

Return Code	Description
0	The request completed successfully.
15	The receiver program is inactive.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
25	The ASCB address is not correct.
26	The receiver program is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
36	ESTAE recovery cannot be established as requested.
40	RECEIVER-ID is not valid.
90	A processing error has occurred.

Usage Notes

The ASCB-ADR for this request must be the same as that specified in request type 4 for this receiver.

Request Type 12: Send an NMVT or CP-MSU Formatted Alert to the NetView Program

Request type 12 is used in sender programs. Request type 12 tells the program-to-program interface that the data buffer you are sending is an NMVT or CP-MSU formatted alert and that the receiver is the NetView alert receiver (NETVALRT). You do not need to specify a RECEIVER-ID in this RPB.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–23	SENDER-ID	DTRSDID
32–35	BUFF-LEN	DTRUBL
36–39	BUFF-ADR	DTRUBPTR
52–55	SAF-ADR	DTRSAFWK

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
56–63	SENDER-SAF-ID	DTRSAFID

Return codes

Return Code	Description
0	The request completed successfully.
4	The specified receiver is not active. The PPI has received a copy of the NMVT or CP-MSU.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
26	The receiver program task is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
32	No NetView storage is available.
33	The buffer length is not valid.
35	The receiver buffer queue is full.
36	ESTAE recovery cannot be established as requested.
40	SENDER-ID is not valid.
90	A processing error has occurred.

Usage Notes

- An NMVT or CP-MSU formatted alert has no length restriction. An alert must include an NMVT or CP-MSU header.
- Control is returned to your program immediately after the NMVT or CP-MSU buffer is copied into the PPI.
- The PPI does not release the storage for the buffer. Your program must release this storage.
- The buffer queue limit for the NetView alert receiver is 1000 NMVT or CP-MSU formatted alerts. If this limit is exceeded, your buffer is not accepted. If you receive a return code of 22 or greater for the request type 12, the buffer has not been sent to the PPI.
- The SENDER-ID is used as the resource name on the hardware monitor Alerts-Dynamic panel. If the hardware monitor hierarchy/resource list subvector (X'05') exists in the NMVT or CP-MSU formatted alert buffer, the resource name specified in this subvector is used instead of the SENDER-ID as the resource name on the Alerts-Dynamic panel.

Request Type 14: Send a Data Buffer to a Receiver Synchronously

Request type 14 is used in sender programs. With a request type 14, you can send a data buffer to the program you specify in the RECEIVER-ID field.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–23	SENDER-ID	DTRSDID
24–31	RECEIVER-ID	DTRRVID
32–35	BUFF-LEN	DTRUBL
36–39	BUFF-ADR	DTRUBPTR
52–55	SAF-ADR	DTRSAFWK

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
56–63	SENDER–SAF–ID	DTRSAFID

Return codes

Return Code	Description
0	The request completed successfully.
4	The specified receiver is not active. The PPI has received a copy of the data buffer.
22	The program issuing this request is not running in primary addressing mode.
23	The sender program is not authorized.
24	The PPI is not active.
26	The receiver program is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
32	NetView storage is not available.
33	The buffer length is not valid.
35	The receiver buffer queue is full.
36	ESTAE recovery cannot be established as requested.
40	SENDER-ID or RECEIVER-ID is not valid.
90	A processing error has occurred.

Usage Notes

- Your program must be APF-authorized to send a data buffer to an authorized receiver. A receiver is defined as authorized by the AUTH-IND field for the request type 4 that initialized the receiver.
- Following the CALL, control is returned to your program immediately after the data buffer has been copied into the receiver buffer queue in the PPI.
- The NetView program does not release the storage for the user data buffer. Your program must release this storage.

Request Type 22: Receive a Data Buffer

Request type 22 is used in receiver programs. A request type 22 receives one data buffer from the receiver buffer queue.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–19	ASCB-ADR	DTRASCB
24–31	RECEIVER-ID	DTRRVID
32–35	BUFF-LEN	DTRUBL
36–39	BUFF-ADR	DTRUBPTR
46–47	CKBTS	DTRCKBTS
64–71	SENDER'S-NAME/ID	DTRSDNAM
72–79	ASID-TOKEN	DTRSDAST
80–95	TCB-TOKEN	DTRSDTT

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC
16–23 [1]	SENDER-ID	DTRSDID
20–23 [1]	ECB-ADR	DTRECB
32–35	BUFF-LEN	DTRUBL
56–63	SENDER-SAF-ID	DTRSAFID
64–71	SENDER'S-NAME/ID	DTRSDNAM
72–79	ASID-TOKEN	DTRSDAST
80–95	TCB-TOKEN	DTRSDTT
Note: One or the other of these fields can be returned, but not both.		

Return codes

Return Code	Description
0	The request completed successfully.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
25	The ASCB address is not correct.
26	The receiver program is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
30	Data buffer is not in the receiver buffer queue.
31	The receiver buffer is not large enough to receive the incoming data buffer.
33	The buffer length is not valid.
36	ESTAE recovery cannot be established as requested.
40	SENDER-ID or RECEIVER-ID is not valid.
90	A processing error has occurred.

Usage Notes

- A receiver can receive one data buffer at a time from the receiver buffer queue in the order of first in, first out (FIFO).
- If the request type 22 is successful, the PPI returns the identifier of the sending program in the SENDER-ID field.
- If the return code is 30, the receiver program can use a request type 24 or a WAIT function to wait until more data buffers are received in the receiver buffer queue. The PPI posts the receiver ECB when the next buffer is received into the receiver buffer queue.
- Ensure that the ASCB-ADR is the same as that specified in the request type 4 that defined this receiver.
- The PPI returns the length of the incoming buffer in the BUFF-LEN field. If the return code is 31, the receiver program needs to allocate a larger buffer and issue request type 22 again.
- Because the receive ECB is cleared whenever a request type 22 returns return code 30, you do not usually need to clear it.
- The settings of CKBTS (byte 46 bit 4, bit 5, and bit 6) determine how buffers are to be received. PPI receives can be done in FIFO order or they can be done by the sender name, sender address space token, sender task token, or any combination of the three. The DTRSDNAM, DTRSDASR, and DTRSDTT fields are filled in on each receive independent of the DTRCKBTS settings. If you set byte 46 bit 4, bit 5, or bit 6, store a value in the corresponding token field prior to a receive unless you want to use the values from a previous operation. If one or more of Byte 46 bits 4, 5, or 6 is set on a subsequent receive, the receive returns data only from senders whose tokens match those indicated by the flags and corresponding fields in the DTR.

Note: If the sender is running in SRB mode (as VTAM sometimes does), the DTRSDTT field is set to binary zeros.

Request Type 23: Purge a Data Buffer

Request type 23 is used in receiver programs. With a request type 23, you can purge a data buffer from the buffer queue.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
6, 7	RECOPT	DTRRECOP
12–15	WORK-ADR	DTRWKPTR
16–19	ASCB-ADR	DTRASCB
24–31	RECEIVER-ID	DTRRVID
46–47	CKBTS	DTRCKBTS
64–71	SENDER'S-NAME/ID	DTRSDNAM
72–79	ASID-TOKEN	DTRSDAST
80–95	TCB-TOKEN	DTRSDTT

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC

Return codes

Return Code	Description
0	The request completed successfully.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
25	The ASCB address is not correct.
26	The receiver program is not defined.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
30	Data buffer is not in the receiver buffer queue.
36	ESTAE recovery cannot be established as requested.
90	A processing error has occurred.

Usage Notes

- The ASCB-ADR for this request must be the same as that specified in the request type 4 that defined this receiver.

- The settings of CKBTS (byte 46 bit 4, bit 5, and bit 6) determine how buffers are to be purged. PPI purges can be done in FIFO order or they can be done by the sender's name, sender's address space token, sender's task token, or any combination of the three. The DTRSDNAM, DTRSDASR, and DTRSDTT fields are filled in on each purge independent of the DTRCKBTS settings. If you set byte 46 bit 4, bit 5, or bit 6, then you must store a value in the corresponding token field prior to a purge unless you want to use the values from a previous operation. If one or more of byte 46 bits 4, 5, or 6 is set on a subsequent purge, the purge purges data only from senders whose tokens match those indicated by the flags and corresponding fields in the DTR.

Request Type 24: Wait for the Receive or Connect ECB

Request type 24 is used in receiver programs returning from the program-to-program interface. A request type 24 functions as a wait macro. Use this request if your programming language does not provide a WAIT function and you want your receiver program to wait for the NetView program to post the receiver ECB.

RPB Fields Specified in the Program

Bytes	Data Field	DTR Field
0–3	RPB-LEN	DTRLEN
4, 5	TYPE	DTRREQT
12–15	WORK-ADR	DTRWKPTR
20–23	ECB-ADR	DTRECB

RPB Fields Returned by the Program-to-Program Interface

Bytes	Data Field	DTR Field
8–11	RETCODE	DTRRETC

Return codes

Return Code	Description
0	The request completed successfully.
18	The receiver ECB is not zero.
22	The program issuing this request is not running in primary addressing mode.
24	The PPI is not active.
28	An active subsystem interface address space was found, but an active PPI address space was not found.
90	A processing error has occurred.

Usage Notes

- Use this request type only if your programming language does not have the ability to wait on an ECB.
- The NetView program returns the ECB-ADR on the request type 4 that initializes the receiver.
- You can use request type 24 after you receive a return code 30 in response to a request type 22, indicating that your receiver program has received all available data buffers. At that point, your receiver can end or wait for the PPI to post the receiver ECB. The PPI posts the receiver ECB when the next data

buffer is received into the receiver buffer queue. You can receive the data buffer using a request type 22.

- Results are unpredictable if the NetView subsystem address space ends while your program is using a request type 24. The post might not occur.
- For applications using this request type running as a NetView application, run them under a NetView optional task. If the application is run under a NetView OST, AOST, or PPT task, DOM buffers might accumulate causing an out-of-storage condition before the PPI wait is satisfied.

Chapter 3. Using REXX to Send Requests

DSIPHONE is a REXX external subroutine that you can use to send and receive data across the NetView PPI.

DSIPHONE is used to return data back to the NetView program when commands are issued to TSO using the PIPE TSO stage.

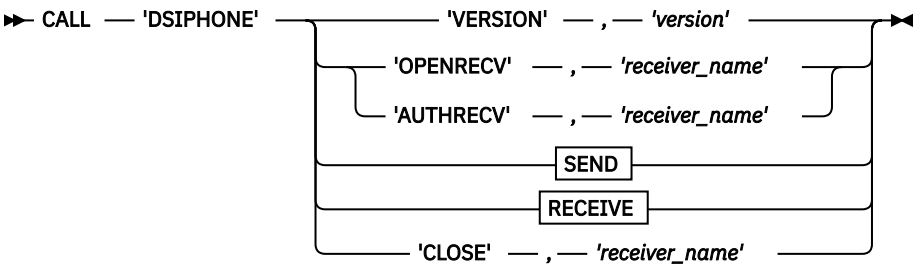
This function enables any z/OS application (capable of running REXX) to open, close, send data to, or receive data from a PPI receiver. For a coding example that defines a server and client application, see “REXX Programming Examples” on page 80.

DSIPHONE is called as a subroutine or function, and requires parameters. It cannot be used in the NetView program (use the PPI pipe stage instead).

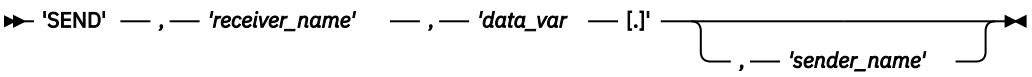
The format for DSIPHONE follows.

DSIPHONE

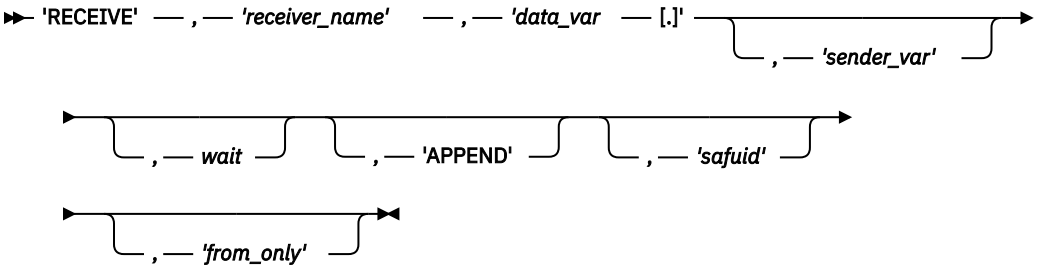
DSIPHONE



SEND



RECEIVE



Note:

1. Quotation marks are optional around the routine name, DSIPHONE, although they are recommended. The absence of quotation marks means that REXX attempts to resolve the routine call internally first.
2. If you do not specify a positional parameter, you must indicate its absence by specifying a comma in its place.

REXX resolves values of variables passed as parameters and passes those values to DSIPHONE. If the variable has not been defined, the name of the variable is passed as the value. If the parameter is enclosed in quotation marks, the literal value of that parameter is passed.

Parameters

VERSION
Return DSIPHONE version information in a REXX variable.

version
The name of a REXX variable into which the DSIPHONE version text string is to be stored.

OPENRECV
Send a request to the PPI to define a new receiver.

AUTHRECV
Send a request to the PPI to define a new receiver that accepts data only from an APF-authorized sender.

SEND
Send data to a PPI receiver.

RECEIVE
Receive a data buffer from a PPI sender.

CLOSE
Send a request to the PPI to close a receiver.

receiver_name
Name of a PPI receiver to which a SEND or RECEIVE request is directed.

sender_name
The originator of a PPI SEND request. This can be the name of the PPI receiver that the sender defined to receive responses. This can be useful when used in conjunction with the *from_only* parameter for client/server applications with multiple clients.

data_var[.]
The name of a REXX variable [or stem] containing data either to be sent to a PPI receiver with a SEND request or to be received from a RECEIVE request. The presence of a period at the end of the name indicates to DSIPHONE that the name is a stem. A period elsewhere in the name indicates a stem element or a compound variable and is treated as a regular variable.

sender_var
The name of a REXX variable into which the sender's PPI receiver name is to be stored.

wait
A numeric value indicating the time (in seconds) to wait for a RECEIVE call to be completed. By default, no time limit exists and DSIPHONE waits indefinitely on a receive call. Any positive value specified for WAIT on a receive call results in a timeout (rc=25) if data does not arrive on the PPI receiver within the specified time. If a WAIT value of 0 is specified and no data is queued to the specified receiver, then DSIPHONE ends with PPI RECEIVE failed (rc=13).

APPEND
Specifies that DSIPHONE is to append to an existing stem. The default is that DSIPHONE creates a new stem or replace an existing one. If APPEND is specified for a stem that does not exist or has no elements, APPEND is effectively ignored.

safuid
The name of a REXX variable into which the SAF userid associated with the data being received is to be stored. The SAF userid is that of the original sender of the data.

from_only
A value specified here restricts the RECEIVE to data sent by programs with this sender name.

DSIPHONE Usage Notes

When a REXX program running in TSO calls DSIPHONE to register a PPI receiver, that receiver remains active only until the program completes. This occurs because TSO drives an end-of-memory routine when the program completes. When the PPI detects this, it marks the receiver inactive.

MLWTO Attributes Support

When sending a stem, default MLWTO attributes are applied to each element of the stem. By building a *dollar* stem, users of DSIPHONE can control MLWTO attribute specification.

When receiving data from the PPI, a *dollar* variable or stem automatically is defined by DSIPHONE to contain the MLWTO attributes currently applying to that data. MLWTO attributes control the line type (control, label, data, or end), color, intensity, and highlighting applied to each line of a message.

A *dollar* variable or stem corresponds to the name of the variable or stem containing the actual data, preceded with the dollar sign ($). A *dollar* variable or stem is a string of data containing blank-delimited, two-byte specifications, such as CR (Color Red), HR (Highlight Reverse), and TD (line Type Data). Specifications that are not valid are ignored, and the last multiply-defined attribute is used. For example, if an attribute string contained 'TD CB CR', the CR color attribute is assumed.

In the following example, the text in `myvar` displays as a red, reversed line when sent across the PPI and received by the NetView program:

```
$myvar = 'TD CR HR'
myvar = 'A line of text'
```

For more information about MLWTO attributes, refer to *IBM Z NetView Programming: Pipes* and *IBM Z NetView Programming: Assembler*.

DSIPHONE Results

Because DSIPHONE is a REXX external routine, the result of a REXX call to DSIPHONE is contained in the REXX-defined variable, `result`. If the DSIPHONE external function has a non-zero completion code, the REXX language processor indicates an `'incorrect call to routine'` return string.

DSIPHONE generates a return string in the REXX variable `result`, which can be parsed in the following way:

```
parse var result phoneCode diagnostic ', rc = ' reasonCode
```

In this case, the REXX variable phoneCode is a 4-byte positive integer that is left-padded with blanks. If applicable, the REXX variable `reasonCode` is the return code from an unsuccessful internal program call by DSIPHONE. The REXX variable `diagnostic` describes the error or the unsuccessful function call or both.

A REXX coding example for handling the DSIPHONE result string follows.

```
call 'DSIPHONE' ..........
parse var result phoneCode diagMsg ', rc = ' reasonCode
if phoneCode <> 0 then
 do
  msg = 'DSIPHONE returned' phoneCode'.'
  if reasonCode ¬= " then
   msg = msg'; PPI return code =' reasonCode
  say msg
  say diagMsg
 end
```

The following table lists the return codes generated by DSIPHONE requests.

Table 3. DSIPHONE Return Codes

nnnn	Text Associated with nnnn	Description
0	(blank)	The call to DSIPHONE completed successfully.
1	DSIPHONE called without arguments	DSIPHONE was called without specifying any parameters.

Table 3. DSIPHONE Return Codes (continued)		
nnnn	**Text Associated with nnnn**	**Description**
3	Too many parameters for this request type	More parameters were passed to DSIPHONE than expected for the given request type. For example, only one parameter is expected on the VERSION call.
4	Too few parameters for this request type	Fewer parameters were passed to DSIPHONE than expected for the given request type. For example, at least two parameters are expected on the SEND call.
5	The request type is not valid.	The first parameter passed to the NetView program was not one of the valid request types (VERSION, SEND, RECEIVE, OPENRECV, AUTHRECV, CLOSE).
6	REXX stem or variable name too long	The name specified for the variable or stem name exceeds 250 bytes.
7	REXX variable *operation* failed, rc =	The REXX command processor reported an error attempting to handle a value for a variable specified in the call to DSIPHONE. For information about the return code value, see the TSO/E REXX library (section on IRXEXCOM). The value *operation* indicates the attempted REXX function.
8	PPI receiver name is too long	The name specified for the PPI receiver name is longer than eight characters.
9	PPI *request* failed, rc =	An unexpected error occurred attempting a PPI *request*. The value after rc= is the return code from the PPI pertaining to the given request. Check the description of this return code in the description of the related PPI *request* in Chapter 2, “Using High-Level Languages and Assembler to Send Requests,” on page 11.
10	Invalid MLWTO attributes	The user has coded MLWTO attributes that are not valid (the dollar variable).
11	REXX variable fetch failed, rc =	The REXX command processor reported an error attempting to retrieve the value of a variable specified in the call to DSIPHONE. The value after rc= is documented in the IRXEXCOM section in the *TSO/E REXX/MVS Reference*.
12	PPI SEND failed, rc =	An unexpected error occurred attempting to send data to a PPI receiver. The value after rc= is the return code from the PPI. Check the description of this return code in the description of request type 14 in Chapter 2, “Using High-Level Languages and Assembler to Send Requests,” on page 11.

Table 3. DSIPHONE Return Codes (continued)		
nnnn	**Text Associated with nnnn**	**Description**
13	PPI RECEIVE failed, rc =	An unexpected error occurred attempting to retrieve data from a PPI receiver. The value after rc= is the return code from the PPI. Check the description of this return code in the description of request type 22 in Chapter 2, “Using High-Level Languages and Assembler to Send Requests,” on page 11.
14	PPI CLOSE failed, rc =	An unexpected error occurred attempting to delete a PPI receiver name. The value after rc= is the return code from the PPI itself. Check the description of this return code in the description of request type 10 in Chapter 2, “Using High-Level Languages and Assembler to Send Requests,” on page 11.
15	Stem .0 element missing	The name of a stem was passed as a variable on a SEND request but the stem .0 value (stem size) was not found.
16	Stem .0 element is not valid.	The name of a REXX stem was passed as a parameter, but the stem's .0 value (stem size) is not a positive integer.
18	Too many elements in stem	The number indicated in the stem's .0 value was greater than 2^{32}–1 (2147483647).
19	First line of MLWTO not control line	MLWTO attributes for a stem were specified, but the first element of the stem was not identified as control line, 'TC'.
21	The WAIT interval specified is not valid.	An value that is not valid was specified for *wait* on the RECEIVE call. The value must be a positive integer value less than (2^{32}–1)/100 (21474836).
23	Argument 6 is not "APPEND" or blank	The sixth parameter on a RECEIVE call, if specified, is not "APPEND".
25	PPI RECEIVE timed out	The interval specified in the *wait* parameter has passed, but no data has been received on the PPI receiver specified.
27	The MLWTO line type attribute is not valid.	The MLWTO line type attribute is not control (TC), label (TL), data (TD), or end (TE).
29	The MLWTO attributes length, *attribute_length* is not valid.	The length of the MLWTO attribute string exceeds 255 bytes.
30	Unable to obtain storage, rc=	DSIPHONE was unable to obtain enough additional memory to store or send a very large REXX value. The value after *rc=* is the return code from the z/OS STORAGE macro.
31	The attempt to call DSIPHONE from NetView is not valid.	Calling DSIPHONE from the NetView program is not allowed.
9999	Internal error, *condition_code*	Contact IBM Software Support.

Chapter 4. Using the NetView LU 6.2 Transport APIs

This chapter describes the NetView management services (MS) transport application program interface (API). It also describes the use of the NetView high performance transport API and the differences between it and the NetView MS transport API.

NetView MS Transport API

The NetView MS transport API handles the protocols required by LU 6.2 conversations. The API provides data to the NetView program to send, defines the destination for the data, and provides a command processor that the NetView program calls when data is received for the API.

The NetView program contains an LU 6.2 API provided by the NetView MS transport that is used to implement a multiple-domain support (MDS) defined by Systems Network Architecture (SNA).

MDS Function

MDS sends and receives management services data, such as alerts or data pertaining to remote operator control, from other devices, including System/390® (S/390®) and non-S/390 hosts. MDS supports high data integrity across the transport and provides guaranteed error notifications for each message.

MS Transport Restrictions

To avoid performance problems, do not use the NetView MS transport for the following functions:

- Forwarding NetView system management facilities (SMF) records between communication management configurations
- Sending the entire contents of databases between two different S/390 systems
- Running performance-critical applications (unless the architecture requires the use of MDS for the application)

NetView High Performance Transport API

The NetView high performance transport API is a generic LU 6.2 API. It functions much like the NetView MS transport API, but uses different LU 6.2 protocols that enhance performance. Most of the external functions for the high performance transport API are the same as for the NetView MS transport API.

For example, the application receives data in the same manner for both APIs. The multiple-domain support message unit (MDS-MU) is put on the command processor initial data queue. The application can use CNMGETD (CNMGETDATA) or DSIGETDS to get the MDS-MU. Also, MDS-MU general data stream (GDS) variables are the enveloping format used for both APIs.

Differences between Transports

The NetView high performance transport API and the NetView MS transport API differ in the following ways:

- The high performance transport API does not perform confirmations on every multiple-domain support message unit (MDS-MU) that is sent. The NetView MS transport API does.
- The high performance transport API enables the LU 6.2 conversations that it uses to remain persistent, or active, even when no data is available to send. This eliminates stopping a conversation during idle periods and then starting it again when data needs to be sent. Starting and stopping the conversation can affect performance if the idle time is short.

The high performance transport API defaults to persistent conversations. However, you can specify that the conversations are to be nonpersistent, or nonpersistent on an LU name basis, by using definition statements.

Conversations on the MS transport API are always nonpersistent.

- The high performance transport API enables applications to specify the logmode to use when sessions are established for the applications to use to send data. Applications can use different logmodes with different session characteristics, as appropriate. When an application registers, it specifies which logmode to use.
- The high performance transport API is not used by the focal point support of the NetView program. This means that high performance applications cannot be focal point applications or receive notification about changes in focal points.
- With the high performance transport API, the LU 6.2 verb flow for the send process is different.

  The high performance confirmation request (CONFIRM) is sent only for the first piece of data sent after an ALLOCATE. All subsequent sends on the conversation are sent unconfirmed. SEND DATA is issued until the sending transaction processor has no more data to send, at which time it issues a FLUSH request. The conversation is not usually deallocated.
- The high performance transaction program name carried in the function management header-5 (FMH-5) is different. For a high performance transport API transaction, the program name is X'23F0F0F2'.

High Performance Transport Restrictions

The following restrictions are for the high performance transport API:

- The high performance transport API applications cannot use the logmode SNASVCMG that is supplied by IBM.
- Applications that require a management services capabilities exchange (for example, focal point applications) cannot use the high performance transport API.

Deciding Which Transport API to Use

When building an application on top of the NetView program, decide which NetView LU 6.2 API your application is to use. This topic provides guidelines on how to choose between the NetView MS transport API and the NetView high performance transport API.

When to Use the NetView MS Transport API

Use the NetView MS transport API in the following situations:

- Your application performs remote operations. Determine whether your application is using the functions and GDS variables (for example, ACTIVATE, DEACTIVATE, or CANCEL) described in the operations architecture. For additional information, refer to *Systems Network Architecture Formats*.
- Your application performs an architected function that collects data from other devices that support only the base function multiple-domain support (MDS), as described in *Systems Network Architecture Formats*.
- Your application is written for a node that performs remote operations functions or forwards alerts to the NetView program, and the node is not a NetView node. Use the MS transport because you are implementing network management architecture categories on a node that is not a NetView node and that communicates using MDS.

When to Use the NetView High Performance Transport API

Use the high performance transport API to write programs that communicate outside of the local host frequently without having those applications affect the LU 6.2 sessions used by the MDS support function. It is not meant to be used for bulk data transfer; the NetView file transfer program (FTP) is an example of a bulk data transfer product.

Use the NetView high performance transport API in the following situations:

- Your application is only transferring data between two NetView systems (for example, forwarding NetView SMF records) and does not need to be expanded to cover other devices.
- Your application performs a function not covered by SNA.
- You require a high transfer rate between your application and the partner application on the other device. Use the high performance transport even if you are transferring architected data as long as the architecture does not specify to use MDS to perform the data transfer.
- Your application works in an environment in which confirmations are not performed on every data flow. For more information, see "NetView System Programmer" on page 68.

Considerations for Applications

If you are a system programmer writing applications that use the LU 6.2 protocols, consider the following items in your design:

- Send-Receive interface
- Tasking structure
- MDS transactions

Send-Receive Interface

NetView programs that send and receive MS data can be written in assembler, C, or PL/I. You can also split the send and receive functions and use different interfaces for each.

NetView send interfaces choices:

- You can provide the NetView program a prebuilt MDS-MU or have the NetView program build the MDS-MU. This is determined by the input parameters that you specify.

  Using a prebuilt MDS-MU simplifies the application, but can require more setup work to use the interface.
- You can choose to wait for responses to requests and decide to buffer the responses or forward them immediately.

  For programs using PL/I and C, you can choose whether the NetView program suspends the program while waiting for a response to a data request. If the program is to remain active, you can specify that replies to a data request are buffered or are immediately forwarded to the program.

  For programs using assembler, you can specify whether replies to a data request are buffered or are immediately forwarded to the program. See "Receiving Synchronous and Asynchronous Replies" on page 42 for more information.

Tasking Structure

As part of designing the application, consider the tasking structure of the application. The task under which the registration macro or command is issued becomes the task to which unsolicited data intended for the application is sent.

A program running under other NetView tasks, however, can pass the application name to the NetView program as the origin application when issuing send requests. In such cases, replies and error messages pertaining to the send request are sent to the task issuing the send request. If send requests are being issued from tasks other than the registering task, ensure that the task is authorized to run the command specified on the registration.

MDS Transactions

MDS architecture supports data requests that require a response. For example, an MDS transaction can consist of an operator's request to a remote host and the expected response from the remote host.

MDS architecture also supports stand-alone requests. These requests are commands or data transmissions that do not require the receiver to respond to the sender.

Logical requests and responses to MDS transactions are implemented entirely at the application level, not at a networking level. Networking software, such as VTAM, generates SNA requests and responses as part of delivering a transaction, but these are not apparent to the application receiving the request, and are not related to the MDS transaction. The response to the transaction comes from the application that receives it.

MDS architecture provides an agent unit-of-work correlator, which provides the ability to correlate requests and responses. MDS architecture also provides error notification to applications if the connection between the applications is lost while a transaction is open.

Requests and Replies

MDS architecture defines the multiple domain support-message unit (MDS-MU) as the data envelopes for requests and replies. MDS-MUs include the agent-unit-of-work correlator, and flag bits that define the purpose of the data in relation to a transaction.

The following requests and replies are possible.

- Request expecting reply

  This request initiates a transaction. The application sends data and expects to receive an MDS-MU containing data related to the request. The transaction is uniquely identified by the agent unit of work correlator in the MDS-MU.

- Request not expecting reply

  This request defines a stand-alone request. The application is not expecting a response, although the NetView program can send an error message to the application using the agent unit of work correlator from the request, if the request itself cannot be successfully sent to its destination.

- Reply not last

  This is part of a multipart reply. The application expects additional replies pertaining to the same request. Each reply contains the same agent unit of work correlator in the MDS-MU.

- Reply last

  This reply completes a transaction. It is the only reply to a request, or is the last part of a multipart reply to a request. When the reply is successfully sent to the application, the NetView program cleans up its internal resources that were being used to track the transaction.

- MDS error message

  This is an error indicator, containing an SNA condition report (SNACR), sent to an application. The error indicator describes an error that occurred while the NetView program was sending an MDS-MU. If the agent unit of work correlator contained in the error message matches the correlator of an outstanding transaction, the NetView program cancels the transaction. The application involved must retry the request or response. If a transaction is canceled this way, both applications involved in the transaction receive an error message.

  An error message is also generated if the timer interval specified (or defaulted) on the send request is exceeded before the last reply is received.

Receiving Synchronous and Asynchronous Replies

Applications using the send interface can receive replies to requests either synchronously or asynchronously. Applications written in assembler can receive only asynchronous replies. Applications written in PL/I and C can receive synchronous or asynchronous replies. NetView users can use the CNMGETD (CNMGETDATA) or DSIGETDS services to obtain replies to requests. Replies can be synchronous or asynchronous:

Synchronous
When replies are received synchronously, the application is suspended after the send request is issued. Control is returned to the application at the next sequential instruction after the last reply to the request is received, or after an error message canceling the request is received.

An error message is generated if the timer interval specified (or set by default) on the send request is exceeded before the last reply is received.

Asynchronous
Asynchronous replies are received after the application issuing the send request returns control to the NetView program. The application is not suspended, and the NetView program uses a specified command to process the received data.

The application can specify that the NetView program is to forward each reply to the application as the reply is received or that the NetView program is to buffer the replies. If the replies are buffered, the NetView program waits until the last reply is received or the request is canceled before issuing the specified application command.

Chaining Replies

Applications can send multipart, chained replies to a request. When sending chained replies, the application must specify the same MDS-MU agent unit of work correlator for each chained reply. The last reply must be identified as last, either by using the appropriate parameter on the send service or by setting the flag bits in a user-built MDS-MU.

An alternative to sending separate replies is to block multiple replies together as one data transmission and use only one NetView send request. The NetView program supports data transmissions of up to 31K. In the case of blocked replies, the receiving application must unblock the replies before processing them.

Saving and Using MDS-MU Correlators

Replies to transactions must use the same agent unit of work correlator contained in the original request. This requires applications to save the correlator until the transaction is complete. Correlators can be built and returned to the application when a request is made, or the application can supply its own correlator.

For replies or error messages pertaining to a transaction, the send interface provides a parameter for passing the correlator for the NetView program to use, or the application can provide it when passing a completely built MDS-MU. If the application provides its own correlator, the correlator must be unique.

Specifying Timer Intervals

MDS transactions involving the NetView program have a limited amount of time to complete before the NetView program cancels the transaction. This time interval can be specified on the send interface and is affected by time intervals (MAXREPLY and RCVREPLY) established with the NetView DEFAULTS command. Refer to NetView online help for more information about the DEFAULTS command. To specify a time interval, choose one of the following time intervals:

MAXREPLY
The first time interval to consider is the maximum amount of time a transaction can remain open. The NetView program default is one day, but it can be set up to one year.

An application can specify up to the DEFAULTS MAXREPLY value on the send interface. The DEFAULTS MAXREPLY value is also used by a NetView application receiving a request.

The amount of time a transaction stays open between two NetView applications is the shorter of the MAXREPLY value on the receive side or the value specified (or set by default) on the sending side.

RCVREPLY
The second time interval to consider is the RCVREPLY value set by the DEFAULTS command.

If no time interval is specified on the send interface, the NetView default of two minutes is used. However, the value can be as high as the value of MAXREPLY.

In determining the timer values for your installation, keep in mind that they apply to all applications using the LU 6.2 protocols. Set the time interval to an appropriate value for all of your applications.

Handling MDS Error Messages

MDS error messages are used to cancel transactions and to tell applications that replies or requests not expecting replies cannot be routed successfully. The NetView program builds error messages and sends them to applications. In such cases, the origin application name is a hexadecimal string of X'23F0F1F0'.

When an MDS error message is sent, an SNA condition report GDS variable is included. This variable has a sense field that describes the error. If an application is sending an error message to cancel the transaction, it can supply its own sense data in the GDS variable.

Error messages can be sent from either the origin or destination of a transaction to cancel the transaction. In addition to an error message, the origin of a transaction can also send a reply last message to cancel a transaction. Your application needs to be able to receive a reply last message even if it is the target of a request.

Chapter 5. Using the SEQUENT Command to Serialize Access to Resources

When one or more programs using a serially reusable resource modify the resource, they should not use the resource simultaneously with other programs or other instances of the same program running in different tasks. Consider, for example, a global variable, global KEEP, or data set (that will hereafter be referred to as the resource) that contains data that needs to be updated by multiple programs running at the same time. In order to maintain the integrity of the data, the updating of the resource needs to be serialized. Some of these programs might only need to read the data; because they are not updating the data, they can access it simultaneously with other programs. Other programs using the data, however, might read, update, and replace the data in the resource. Each of these programs must be able to update and replace the data without other programs accessing it. In addition, none of the programs that are only reading the data should retrieve the data that another program is updating until after the data has been replaced.

If your program uses a serially reusable resource, you must prevent incorrect use of the resource. You must ensure that the logic of your program does not allow a second use of the resource before completion of the first use. An Assembler program running in the NetView address space could use the ENQ and DEQ macros to serialize uses of resources. However, command procedure environments do not have access to ENQ/DEQ, unless they invoke an assembler program to do so. Using this method, however. would not be a viable solution, because while waiting for the resource to become available, the entire task does not run. Because of this, NetView processing such as giving control to high priority commands and automating messages does not take place during the wait. The longer the wait for a resource using ENQ, the more backed up a NetView task could become. The solution to these issues is to use the SEQUENT command when running in a command procedure environment (REXX, HLL, or CLIST language). This command allows for suspension of only the roll group that invokes it while waiting for a resource to become available.

Use the SEQUENT command with the OBTAINEX (for exclusive use) or OBTAINSH (for shared use) keyword to assign control of a resource to the current roll group. The NetView program determines the status of the resource and takes one of the following actions:

- If the resource is available, the NetView program grants the request by returning control to the requesting program.
- If the resource has been assigned to another task exclusively, the NetView program suspends the requesting roll group until the resource becomes available.
- If the resource has been assigned to another roll group running in the same task, the NetView program fails the request if either program requested the resource as exclusive. Otherwise (another roll group running under the same task has shared use of the resource and the current request is for shared), the NetView program grants the request.

A program that has obtained a resource makes it available to other requests by using the RELEASE keyword on the SEQUENT command. When all shared uses or the single exclusive use of the resource has been released, other waiting programs are given a chance to obtain control of the resource.

When an obtain request fails or the resource becomes available, the program resumes processing, with the return code set to indicate the completion condition, as long as the roll group has not been interrupted by a high priority command. In such a case, if the requesting roll group is not able to immediately resume, other requesting programs are given a chance to obtain the resource. See Processing the Requests in this chapter for more information.

Be especially careful when using a serially reusable resource in an exit routine or a program running as high priority that gets control in the same task as one which performs processing related to that resource. Because these programs can interrupt other programs running under the same task, they can cause a lockout condition. See Avoiding Interlock in this chapter for more information.

Naming the Resource

The SEQUENT command identifies the resource by a character string, sometimes referred to as the "SEQUENT name". This name needs not have any relation to any actual name of the resource whose usage is being serialized. The NetView program does not associate a name with an actual resource; it merely processes requests having the same name (matching exactly, including case) on a first-in, first-out basis (as long as the requesting roll group is available to obtain the resource when it becomes available). It is up to you to associate the name with the resource by ensuring that all users of the resource use the same name to represent the same resource. The NetView program treats request having different SEQUENT names as requests for different resources.

Choose SEQUENT names carefully. Because the NetView program itself might use the SEQUENT command internally, any names that are used would begin with one of its 3-character prefixes in uppercase, which includes the following prefixes:

- AAU, AQN, BNJ, CNM, DSI, DUI, DWO, EJN, EKG, EZL, FKV, FKX, FLB, FLC, and IHS.

Therefore, you should avoid using SEQUENT names that begin with one of these prefixes or any other that is owned by an IBM program that can run in the NetView address space.

Requesting Exclusive or Shared Control

To obtain a SEQUENT resource, you specify either OBTAINEX (for exclusive control) or OBTAINSH (for shared control).

When your program has exclusive control of a resource, it is the only one that can use that resource; all other programs that issue the SEQUENT command with the OBTAINEX or OBTAINSH keyword for the same SEQUENT name must wait until your program issues the SEQUENT command with the RELEASE keyword, specifying that name. If your program is written to change the contents of the resource, it should request exclusive control.

When your program has shared control of a resource, other programs can have concurrent access to the same resource, if they also use the OBTAINSH keyword and there is not a request for exclusive use before their request. If another program requests exclusive control over the resource during the time your program has shared use of the resource, that program will have to wait until all the current users (with shared access) have issued the SEQUENT command with the RELEASE keyword to release the resource. If your program is written to read but not change the contents of the resource, it should request shared control.

For an example of how the NetView program processes requests for exclusive and shared control of a resource, see Processing the Requests in this chapter.

Controlling overuse of SEQUENT Names

The NetView program performs periodic cleanup of unused SEQUENT names. The more names that are currently known, the more often this cleanup occurs, and the less time since a name was last referenced needs to have passed before a name is eligible to be deleted. Normally, SEQUENT names are kept around for some time so that displays can be done (using the LIST keyword) to show the latest usage of the names, and also to reduce the overhead of recreating control blocks when a new name is defined. However, when a name has not been referenced for some time, the NetView program might determine that it can be deleted, to recover some storage. If just a few names are known, names will stay around much longer (up to a week after having been used last). When many names are known, unused names will be deleted after a much shorter time (as little as 2 minutes). This allows for the possibility of a program mistakenly using many different names and then being corrected to use the proper set of names. Having those names being defined for the life of the NetView program would be a waste of storage and

processing time, so the cleanup function allows the number of defined SEQUENT names to again become manageable.

Processing the Requests

The NetView program constructs a unique list for each SEQUENT name that is requested to be obtained. When a program (REXX, HLL, or CLIST language directly or indirectly called on behalf of a command processor) makes an obtain request using the SEQUENT command, the NetView program searches the existing lists for a matching name. If it finds a match, the NetView program adds the request to the end of the existing list, unless another roll group under the same task is already waiting to obtain or owns the same resource (and both requests are for shared access), in which case the request is added just before that request (as it is most likely interrupting the other roll group and would need to be satisfied first). If the NetView program does not find a matching name, it creates a new list, and adds the roll group's request as the first (and only) element. The roll group gets control of the resource based on the following conditions:

- The position of the roll group's request on the list.
- Whether the request was for exclusive or shared control.
- Whether the roll group is able to be resumed when the resource becomes available.

The best way to describe how the NetView program processes the list of requests for a resource is through an example. The following figure shows the status of a list built for a particular SEQUENT name. The S or E next to the entry indicates that the request was for either shared or exclusive control. The roll group added as the first entry on the list always gets control of the resource, so the roll group represented by GROUP1 in Step 1 is assigned the resource. The request that established GROUP2 in step 1 was for exclusive control, so the corresponding roll group is suspended, as well as the roll groups represented by all the rest of the entries in the list.

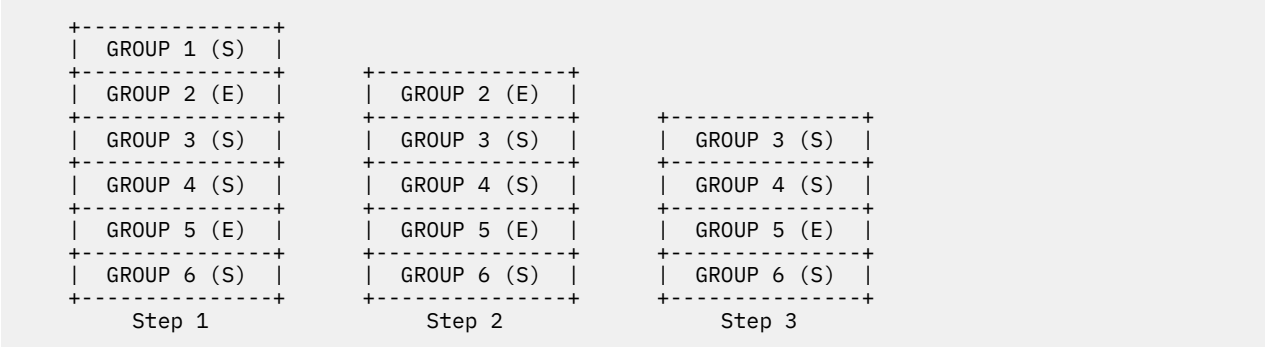


Figure 6. SEQUENT obtain processing

Eventually, the roll group represented by GROUP1 issues the SEQUENT command with the RELEASE keyword for the SEQUENT name to release control of the resource, and the entry for GROUP1 is removed from the list. As shown in the Figure 6 on page 47, Step 2, GROUP2 is now first on the list, and, as long as the roll group is able to be resumed (i.e. it has not been interrupted by a high priority command or other processing is not taking place under the task), the corresponding roll group is assigned control of the resource. The roll group is given a very short amount of time to take control of the resource; if that time has elapsed without taking control, the next request is offered control. (Note that the issuing program is not running, so there is nothing that code can do to affect this situation.) Because the request that established GROUP2 was for exclusive control, the roll groups represented by all the other entries in the list remain suspended.

In the Figure 6 on page 47, Step 3 shows the status of the list after the task represented by GROUP 2 releases the resource. Because GROUP3 is now at the top of the list, the roll group represented by GROUP3 is offered control of the resource. The request issued by GROUP3 indicates that the resource can be shared, and, because the request issued by GROUP4 also indicates that the resource can be shared, GROUP4 is also offered control of the resource. If both GROUP3 and GROUP4 get control of the resource,

the roll group represented by GROUP5 does not get control of the resource until both the roll groups represented by GROUP3 and GROUP4 release control.

The NetView program uses the following general rules in manipulating lists when the roll groups are resumable at the time a resource becomes available:

- The roll group represented by the first entry in the list always gets control of the resource.
- If the request is for exclusive control, the roll group is not offered control of the resource until its request is the first entry in the list.
- If the request is for shared control, the roll group is offered control either when its request is first in the list or when all the entries before it in the list also indicate a shared request.

The NetView program uses the following general rules in manipulating lists when one or more roll groups are not available to be resumed at the time a resource becomes available:

- When all roll groups in a set (i.e. contiguous shared requests or an exclusive request) have been offered control and no roll groups in that set have responded, roll groups in the next set are offered control.
- After an exclusive request has been passed, only one shared request is allowed control of the resource. This is to give earlier exclusive requests further chances to take ownership before offering it to other shared requests again.
- If no roll groups are available to take ownership, the resource is available to any roll group that next becomes active.
- Each time a resource becomes available (i.e. upon a release request that leaves the resource as unowned), the NetView program starts with the first request on the chain.

Releasing the Resource

Use the SEQUENT command with the RELEASE keyword to release a serially reusable resource that you obtained by using the SEQUENT command with the OBTAINEX or OBTAINSH keyword. If a program running in a roll group tries to release a resource which it does not control, the NetView program returns a non-zero return code. The NetView program might suspend many roll groups in many tasks while one roll group maintains ownership of a resource. Having many tasks in this state might reduce the amount of work being done by the system; therefore, you should issue a SEQUENT command with RELEASE as soon as possible to release the resource, so that other programs can use it.

The NetView program automatically releases a resource, and issues message BNH417I when a SEQUENT resource is held and one of the following conditions occurs:

- The roll group terminates before issuing a RELEASE of the resource. This might be as a result of the program neglecting to issue a RELEASE request before terminating or because a RESET command was issued on the task.
- The task is normally terminated (i.e. LOGOFF).
- The task is abnormally terminated (i.e. ABEND).

Avoiding Interlock

An interlock condition happens when two tasks are waiting for each other's completion, but neither task can get the resource that it needs to complete. The Figure 7 on page 49 shows an example of an interlock. Task A has exclusive access to resource Seq1, and task B has exclusive access to resource Seq2. When task B requests exclusive access to resource Seq1, task B is suspended because task A has exclusive control of resource Seq1.

The interlock becomes complete when task A requests exclusive control of resource Seq1. Note that the same thing can happen if the OBTAINEX request for Seq2 under task A had been issued by a command running as high priority, which interrupted the command running low priority that issued the OBTAINEX

for Seq1. Other tasks requiring either of the resources are also suspended because of the interlock, although in this case they did not contribute to the conditions that caused the interlock.

```
        Task A                          Task B
     SEQUENT OBTAINEX=Seq1
                                    SEQUENT OBTAINEX=Seq2
                                    SEQUENT OBTAINEX=Seq1
     SEQUENT OBTAINEX=Seq2
```

Figure 7. Interlock Condition

The example in the Figure 7 on page 49 involving two tasks and two resources is a simple example of an interlock. The example could be expanded to cover many tasks and many resources. It is important that you avoid interlock. The following procedures indicate some ways of avoiding interlocks:

- Do not request resources that you do not need immediately. If you can use the serially reusable resources one at a time, request them one at a time and release one before requesting the next.
- Share resources as much as possible. If the requests in the lists shown in the above figure had been for shared control, there would have been no interlock. This does not mean that you should share a resource that you will modify. It does mean that you should analyze your requirements for the resources carefully, and not request exclusive control when shared control is enough.
- If there are many users of a group of resources and some of the users require control of a second resource while retaining control of the first resource, it is still possible to avoid interlocks. In this case, each user should request control of the resources in the same order. For instance, if resources A, B, and C are required by many tasks, the requests should always be made in the order of A, B, and then C. An interlock situation will not develop, since requests for resource A will always preceded requests for resource B.
- Keep in mind that commands running at high priority (possibly invoked with CMD HIGH) that interrupt other commands can be of as an extension of the interrupted command, with respect to SEQUENT resources. When writing a program, consider the effects of obtaining SEQUENT resources with respect to other commands it might interrupt if running as high priority.
- Because NetView operators have the ability to issue the ROLL command to change the order of roll groups, try to avoid holding a SEQUENT resource while ROLL can be issued. Changing the order of when roll groups get control can result in an interlock condition.
- Be aware that scenarios involving multiple SEQUENT resources are not the only things that might be involved in interlocks. If, for example, a program running under task B holds SEQUENT resource Seq1, but is also written to continue only when a program running under task A has completed a certain function. However, if the program running under task A is requesting exclusive use of Seq1 before beginning the function (which it can't get because task B owns the resource), and interlock would occur. Therefore, consider dependent functions in addition to SEQUENT resources when choosing the order in which to perform functions.

REXX Example

Programs (most often commands) running in the NetView address space are oftentimes executed on different tasks. This means that the same code can run in several places, possibly at the same time. Since different operator IDs are also MVS tasks, there might be a need to serialize usage of a common resource.

Consider, for example, the need for a program to keep track of which operator IDs are executing that program. A PIPE using global KEEP could be written to keep a list of those operator IDs that are executing the code. Here is some REXX code that would perform such a function:

```
myOP = LEFT(OPID(),8)
    'PIPE (NAME TRACKOPS END ;)' ,
         '  KEEP GLOBAL /SEENOPS/' ,
         '| NLOC /'myOP'/' ,
         '| GATHER: FANIN' ,
         '| KEEP GLOBAL /SEENOPS/ *' ,
```

```
         '; VAR myOP' ,
         '| GATHER:'
```

This PIPE first reads the contents of global KEEP named SEENOPS, removes the ID of the current operator if it's in the KEEP, and re-writes the global KEEP with the ID of the current operator at the end of the list. Although this is a single command, it contains two global KEEP stages: one to read the contents and one to write the contents. If this piece of code were to run on multiple tasks at the same time, the contents of the KEEP might not be reliable. Two or more tasks could be reading the contents of the KEEP at the same time, and then update the KEEP with the new contents, eliminating the operator ID from the list for one or more of the tasks. Or, one task could be reading the KEEP while another is writing to it, so it might end up with only part of the contents, or even a duplicate of some of the operator IDs.

So, you see that this piece of code needs to be serialized. Here is the same piece of code protected by the SEQUENT command:

```
myOP = LEFT(OPID(),8)
    'SEQUENT OBTAINEX=KEEPOPERLIST'
    'PIPE (NAME TRACKOPS END ;)' ,
         '  KEEP GLOBAL /SEENOPS/' ,
         '| NLOC /'myOP'/' ,
         '| GATHER: FANIN' ,
         '| KEEP GLOBAL /SEENOPS/ *' ,
         '; VAR myOP' ,
         '| GATHER:'
    'SEQUENT RELEASE=KEEPOPERLIST'
```

Note that the OBTAINEX and RELEASE are issued as close to the function needing serializing as possible, to minimize the amount of time that the resource is held. Having the SEQUENT command invocations present allows for the integrity of the global KEEP to be maintained. Note also that the name of the SEQUENT (KEEPOPERLIST) doesn't match any name referenced in the PIPE.

To see the list of operators in the KEEP while other operators might be updating it, the following piece of REXX code could be written:

```
'SEQUENT OBTAINSH=KEEPOPERLIST'
    ADDRESS NETVASIS ,
      'PIPE (NAME LISTOPS)' ,
           '  KEEP GLOBAL /SEENOPS/' ,
           '| LIT /This is the list of operators:/' ,
           '| CONS'
    'SEQUENT RELEASE=KEEPOPERLIST'
```

Note that as the global KEEP is only being read, shared access is sufficient to provide data integrity. The SEQUENT OBTAINSH invocation will prevent the updating of the list by the first piece of code, as OBTAINEX requires exclusive access. However, multiple operators can see the list at the same time, since the access is shared.

Chapter 6. Management Services Applications

The NetView MS transport makes it possible for management services applications that are supplied with the NetView product or written by users to communicate with management services applications in other logical units (LUs) over LU 6.2 sessions. The transport acts as an interface between the application and VTAM, establishing LU 6.2 conversations and keeping track of outstanding requests and time outs.

The NetView MS transport has the following three interfaces:

- Registration services
- Send macro
- Receive macro

For additional information about the PL/I, C, and assembler interfaces, see these books:

- *IBM Z NetView Programming: PL/I and C*
- *IBM Z NetView Programming: Assembler*

Registration Services

An application uses the CNMRGS (CNMREGIST) service routine in PL/I and C, or the DSI6REGS macro in assembler, to inform the MS transport that it is ready to send and receive data.

The application can specify:

- Its application name (required).
- A command to run when unsolicited data is received for that application (required).
- A focal point category about which it wants to receive focal point information.
- Whether the application is a focal point application.
- Whether the application is to receive special notification of session outages at other management services nodes.
- Whether the application is suspended after it issues a send request (CNMREGIST service routine only).
- Whether replies to a send request are buffered or immediately forwarded to the application (for the CNMREGIST service routine, only if the application is not suspended).

Session Outage Notification

When registering, an application can choose the type of session outage notification:

ALL
Session outage notifications are received even if the NetView program cannot determine that the outage is caused by a problem.

ERROR
Notification is received only when the NetView program can determine that the session outage is abnormal and it cannot establish a conversation with the affected node.

NONE
No session outage notifications are received (the default).

Session outage information is provided only when the last LU 6.2 session to a node with which the MS transport has been in contact is lost. Non-LU 6.2 session outages do not drive the notification.

REGISTER Command

Use the REGISTER command to access the functions provided by the CNMRGS (CNMREGIST) service routine in PL/I and C, or the DSI6REGS macro in assembler. Refer to NetView online help for more information about the REGISTER command.

Send Macro

An application uses the CNMSMU (CNMSENDMU) service routine in PL/I and C, or the DSI6SNDS macro in assembler, to send data to another registered application in its own node or another node.

When using the CNMSMU (CNMSENDMU) service routine, an application can specify whether it is suspended after it issues a send request, or, if it remains active, whether replies to the send request are buffered or immediately forwarded to the application. When using the DSI6SNDS macro, an application can specify whether replies to a send request are buffered or immediately forwarded to the application.

Destination Name

Your application can specify either a NetView LU name or a VTAM CP name as the destination name of an MDS send request.

Use the VTAM CP name instead of the NetView LU name when issuing an MDS send request to a NetView program running under VTAM. This procedure affects the following NetView services:

- DSI6SNDS macro
- CNMSMU (CNMSENDMU) HLL interface
- FOCALPT command
- DEFFOCPT statement

Note: Current® applications that use the NetView LU names do not need to be changed.

When several NetView programs run under VTAM, only one of them can receive multiple domain support message units (MDS-MUs) addressed to the VTAM CP name. Specify which NetView program can receive MDS-MUs addressed to the VTAM CP name specified with the VTAMCP.USE statement in the CNMSTYLE member. See the *IBM Z NetView Administration Reference* for information about the VTAMCP statement.

The receiving NetView program must have VTAMCP.USE=YES specified in the CNMSTYLE member (if Version 5 or later), or it must have VTAMCP USE=YES specified in the DSIDMNK member (for releases prior to Version 5).

When replying to an MDS send request using NetView services, you must ensure that the reply destination matches the origin of the request. A request originating from another NetView program with the VTAM CP name as its origin must be replied to using the origin CP name (in place of the NetView LU name) as the destination.

Note: For send requests within the same NetView program, the send service enters the NetView LU name as the origin LU.

Restrictions

The send macro has several restrictions:

- VTAM must be active for data to be sent, even to an application within the same node.
- When data is sent within the same node, the origin and destination application names must be different.
- If one of the applications is an operations management served application, the other application communicating with it must use the routing and targeting instruction general data stream (R&TI GDS) variable in the data that it sends to the operations management application. For an application-reported problem, use an MS request or reply containing a routing report, as defined in MS architecture, to report the problem.

- If you have interconnected networks, destination LU names must be unique to ensure reliable routing. If a blank NETID is supplied (or set by default), the NetView program fills in the correct NETID prior to sending data through the network.

Receive Macro

A receive service, CNMGETD (CNMGETDATA) in PL/I and C and DSIGETDS in assembler, allows an application to receive data from another application, including a focal point notification from operations management.

Implementing the Application

To implement a management services application in the NetView program, system programs and NetView operators need to follow the instructions described in this section. You can implement the application in LUs with the NetView program or in LUs without the NetView program.

NetView Operator

The NetView MS transport support is transparent to the NetView operator. However, the system programmer can instruct the operator to issue the registration command (REGISTER) to define the NetView program status as a focal point or entry point for a certain category of management services. An operator can also use the REGISTER command with the QUERY option to display a list of registered applications. Refer to NetView online help for more information about the REGISTER command.

The NetView operator sees messages related to the transport functions. Several other messages are logged only to the network log to assist in problem determination. Most of the messages have prefixes of DWO45 and DWO46. LU 6.2 problems might generate VTAM error messages or DSI769I messages. Refer to NetView online help for a complete description of these messages.

Syntax errors in data received from other nodes cause an alert to be passed to the hardware monitor.

NetView System Programmer

To implement a management services application in the NetView program, the system programmer performs the following tasks:

1. Identify a management services category to process.

   Management services categories (IBM-defined or user-defined) are collections of processes used to monitor and manage networks. IBM defines the following major management services categories:

   - Problem management
   - Performance and accounting management
   - Configuration management
   - Change management

   Within each major category, IBM has defined functional subsets of the category. ALERT_NETOP, for example, is one of the subsets of the problem management category that deals with alert data. The functional subsets are defined in *Systems Network Architecture Formats*.

   The functional subsets are implemented by the system programmer as management services application programs that send and receive data. These applications can participate in focal point-entry point relationships. For details on focal points and entry points, refer to *IBM Z NetView Automation Guide*.
2. Create a management services application.

   Write command processors to send and receive data for that management services category. You can use one command processor to handle both sending and receiving. You can access the interfaces to send data from high-level language (HLL) or assembler command processors, or access the interfaces to receive data from HLL, assembler, or REXX command lists.

3. Register the management services application.

   You can use the macro interface in a command processor or the command interface.
4. Determine whether your application uses the NetView destination LU name or the VTAM CP name as the destination name in MDS send requests.
5. Use the CNMSMU (CNMSENDMU) service routine in PL/I and C or the DSI6SNDS macro in assembler to:

   a. Send requests to other management services applications in the same node or a different node.

   b. Send replies to requests received from other management services applications.
6. Deregister the management services application. You can use the macro interface in a command processor or the command interface. Do this when you no longer want to send or receive data for that management services category.

   When an application is deregistered, any outstanding send requests expecting a reply are canceled, and MDS error messages are sent to the other applications involved in the transaction. This is true even for send requests originating under a task other than the registered task. Deregistration prevents applications from sending data using the deregistered application name as the origin application.

Other System Programmer

Applications running in LUs without the NetView program can communicate with the MS transport. For this communication, this type of LU must implement MDS-SEND and MDS-RECEIVE transaction programs that are similar to those of the NetView MS transport. For more information about communicating from these types of applications, see *Systems Network Architecture Formats*.

The criteria for program communication using the NetView MS transport includes the following items:

- Applicable portions of the LU 6.2 architecture
- The send process
- The timeout message

Applicable LU 6.2 Architecture

This section describes the portions of the LU 6.2 architecture that provide application choices; it does not describe the entire LU 6.2 architecture. It also describes function management header-5 (FMH-5) restrictions on programs communicating with the NetView program.

Refer to *Systems Network Architecture Formats* for details of the MDS-MU encoding and the MDS transport architecture for the MDS-SEND and MDS-RECEIVE transaction programs.

BIND Setting

The NetView program uses VTAM LU 6.2 support. The following information describes how the VTAM program handles the bytes in the BIND. The information uses zero-origin indexing. The first byte of the BIND is zero (0), the second is 1, and so on.

On initial contact with a previously unknown LU, the VTAM program assumes that the LU supports parallel sessions (byte 24 of the VTAM BIND settings set to X'23') and attempts to establish a SNASVCMG session to negotiate session limits.

You can change the byte-24 setting to X'2C' if your LU supports only single sessions and you want to negotiate the BIND.

You can reject the BIND with a sense code of X'0835xxxx', where xxxx is one of the following values:

- The value of X'0018', the offset of the byte for parallel session support (byte-24 in Table 4 on page 55)
- The offset of the first character of the SNASVCMG name in the mode name structure subfield

Table 4. VTAM BIND Settings	
Bytes	**Description**
0–6	Always set to X'31001307B0B050'. The second byte implies that the BIND can be negotiated. The LU is not obligated to do anything other than send a positive or negative response, but can perform other functions.
7	Specifies contention results. **X'B3'** The BIND sender is the contention winner for the session. **X'A3'** The BIND sender is the contention loser for the session. This byte also indicates that VTAM includes control vectors in the BIND, making it an extended BIND.
8, 9	Controls secondary logical unit (SLU) pacing. VTAM indicates one-stage pacing, and the exact pacing window values can vary. VTAM supports adaptive session-level pacing.
10	Controls the SLU's maximum send RU size. VTAM accepts values from X'80' to X'FF'. The default is X'85' (256 bytes). See *Systems Network Architecture Formats* for the hexadecimal format of this byte.
11	Controls the primary logical unit's (PLU's) maximum send RU size. VTAM supports X'80' to X'FF' with a default of X'85'.
12, 13	Controls PLU pacing. VTAM supports adaptive session pacing. VTAM's default pacing window sizes are X'3F'.
14–22	Always set to X'060200000000000000'.
23	Indicates security support. The NetView program does not use security features that are available in LU 6.2 architecture. Therefore, the default value of this byte is X'00'.
24	Contains LU 6.2 flags. The VTAM program sets the byte to X'23' for parallel session capable partners or X'2C' for single-session capable partners.
25, 26	Always set to zero. This indicates that limited-resource sessions and cryptography are not supported.

VTAM includes the following structured subfields in the BIND:

- Mode name
- Session-instance identification (ID)
- Network-qualified PLU name

No user-request correlator is present. The fully-qualified program control interrupt (PCI) control vectors, class-of-service control vectors, and transmission priority control vectors are included.

FMH-5 Restrictions

The NetView program does not use the full range of LU 6.2 options on its conversations. The following list shows restrictions on the function management header-5 (FMH-5) that can be sent to start a conversation:

- The transaction program (TP) name you specify in the FMH-5 must be X'23F0F0F1', which is the architected name for MDS-RECEIVE.
- The NetView program supports only basic conversations.
- The synchronization level must be CONFIRM.
- Security subfields are not used and are not accepted.

- The NetView program does not use the logical agent unit of work correlator (UOWC) that can be included in the FMH-5.
- Program initialization parameter (PIP) data is not supported.

The NetView LU 6.2 support uses the SNASVCMG mode to implement simple send and receive transaction programs that use only a small subset of permissible LU 6.2 verbs. Your LU should not attempt to maintain a conversation to send data indefinitely. SNASVCMG might be needed for internal LU processing. The partner LU must periodically deallocate the conversation. To prevent problems, follow the management services architecture closely.

Data flow on a conversation is always one-way. If an error is detected on a conversation, the correct way to report it is to deallocate the conversation abnormally (DEALLOC TYPE=ABNDPROG *verb*). Partner LUs should not attempt to reverse the data flow direction. Specifically, your LU should not issue the following requests:

- SEND-ERROR or REQUEST-TO-SEND from a receiving application
- RECEIVE or PREPARE-TO-RECEIVE from a sending application

If SEND-ERROR, RECEIVE, or PREPARE-TO-RECEIVE are detected, the NetView program abends the conversation and ignores REQUEST-TO-SEND.

In addition, partner LUs should not attempt to include error log data on an abnormal conversation deallocation, because the NetView program does not support the function. The NetView program receives error log data, then discards it.

The NetView program includes confirmation requests on the data it sends and expects confirmation requests sent with the data it receives.

Send Process

The NetView program uses the following process to send the data that has been passed to it:

1. The NetView program determines whether session limits have been established with your LU. If not, the NetView program issues a change number of sessions (CNOS) verb request.

   Note: If your LU has established a session with the NetView program using an INIT-SELF or has set session limits as part of its initialization, the NetView program does not issue the CNOS request.
2. When session limits are set, the NetView program issues an ALLOCATE request.
3. The NetView program issues SEND-DATA and CONFIRM messages until nothing is left to send. In some error recovery cases, the NetView program can issue a CONFIRM message with no data preceding it.
4. When no data remains to be sent, the NetView program issues a DEALLOCATE request.

Time Out Message

The NetView program enables command processors using the transport service to specify a time interval within which a response must be received for requests. If this timer expires, the NetView program cancels the request and sends a message to your LU indicating that a reply is no longer required. This message is in the form of a sense code (X'08A90003').

Chapter 7. Operations Management Served Applications

Operations management is an MS application that enables operations management served applications that are supplied by the NetView product or written by users to send architected operations management commands to remote systems for processing. The extended routing capabilities available using routing and targeting instruction (R&TI) GDS variables within a CP-MSU make it possible to:

- Route the command to the appropriate command processor in the target system for processing.
- Route the acceptance report, completion reports, and other delayed replies back to the same instance of the issuing served application.
- Correlate the reports and delayed replies with the original command.
- Inform a served application in an entry point node of the identity of the focal point for unsolicited operations management data.

With these capabilities, two operators or autotasks can use the same served application program to control two different remote systems. Each can receive the replies and reports for the system it is controlling.

The following interfaces enable a served application to achieve operations management communication:

- Registration service
- Send macro
- Receive macro

Refer to the following documents for more information about the PL/I, C, and assembler interfaces:

- *IBM Z NetView Programming: PL/I and C*
- *IBM Z NetView Programming: Assembler*

Registration Service

A registration service macro CNMRGS (CNMREGIST) service routine in PL/I and C or the DSI6REGS macro in assembler specifies the following information to operations management:

- Service application name
- Command processor name
- Task name for receiving unsolicited data
- Whether to be informed of focal point information

Buffering Replies

When using the CNMRGS (CNMREGIST) service routine, an application can specify whether it is suspended after it issues a send request, or, if it remains active, whether replies to the send request are buffered or immediately forwarded to the application. When using the DSI6REGS macro, an application can specify whether replies to a send request are buffered or immediately forwarded to the application.

Session Outage Notification

When registering an application, the type of session outage notification it receives can be specified:

ALL
Session outage notifications are received even if the NetView program cannot determine that the outage is caused by a problem.

ERROR
Notifications are received only when the NetView program can determine that the session outage is abnormal.

NONE
No notifications are received (the default).

Session outage information is provided only when the last LU 6.2 session to a node with which the MS transport layer has been in contact is lost. Non-LU 6.2 session outages do not drive the notification.

REGISTER Command

Use the REGISTER command to access the functions provided by the CNMRGS (CNMREGIST) service routine in PL/I and C, or the DSI6REGS macro in assembler. Refer to NetView online help for more information about the REGISTER command.

Send Macro

A send service macro (CNMSENDMU service routine in PL/I and C or the DSI6SNDS macro in assembler) allows the served application to send data to another registered application in the same node or a remote node.

When using the CNMSMU (CNMSENDMU) service routine, an application can specify whether it is suspended after it issues a send request or, if it remains active, whether replies to the send request are buffered or immediately forwarded to the application. When using the DSI6SNDS macro, an application can specify whether replies to a send request are buffered or immediately forwarded to the application.

Destination Name

Your application can specify either a NetView LU name or a VTAM CP name as the destination name of an MDS send request.

Use the VTAM CP name instead of the NetView LU name when issuing an MDS send request to a NetView program running under VTAM. This procedure affects the following NetView services:

- DSI6SNDS macro
- CNMSMU (CNMSENDMU) HLL interface
- FOCALPT command
- DEFFOCPT statement

Note: Current applications that use the NetView LU names do not need to be changed.

When multiple NetView programs run under VTAM, only one of the NetView programs can receive multiple domain support message units (MDS-MUs) addressed to the VTAM CP name. Determine which one of several NetView programs can receive MDS-MUs addressed to the VTAM CP name with the VTAMCP.USE statement by using the CNMSTUSR or C*xx*STGEN member. For information about the VTAMCP statement, see *IBM Z NetView Administration Reference*; for information about changing CNMSTYLE statements, see *IBM Z NetView Installation: Getting Started*.

Both the sending and receiving NetView programs must have VTAMCP.USE=YES specified in the CNMSTYLE member (if Version 5 or later), or it must have VTAMCP USE=YES specified in DSIDMNK (for releases prior to Version 5).

When replying to an MDS send request using NetView services, ensure that the reply destination matches the origin of the request. A request originating from another NetView program with the VTAM CP name as its origin must be replied to using the origin CP name (in place of the NetView LU name) as the destination.

Note: For sends within the same NetView program, the send service enters the NetView LU name as the origin LU.

Restrictions

The send macro has several restrictions:

- VTAM must be active for data to be sent, even to an application within the same node.
- Unless an application is operations management, when data is sent within the same node, the origin and destination application names must be different. If the origin application and destination application are operations management, and the served application is sending from within the same node, the origin application name in the R&TI must be different from the destination application name.
- Served applications use the R&TI GDS variable in the data they send to other served applications. A routing report contained in an MDS request or reply should be used to report problems to partner applications. The origin application in the MDS-MU must be operations management, and the origin application name in the R&TI is the served application name. If an origin application specifies a destination instance identifier when it sends a request with reply, the identifier must be the same as the issuing task.

Receive Macro

A receive service routine, CNMGETD (CNMGETDATA) in PL/I and C and DSIGETDS in assembler, enables the served application to receive data from another application, including a focal point notification from operations management.

Implementing the Application

To implement an operations management served application in the NetView program, system programmers and the NetView operator must follow the instructions described in this section. You can implement the application in the NetView program or in LUs that do not use the NetView program.

NetView Operator

The NetView MS transport support is transparent to the NetView operator. However, the system programmer can instruct the operator to issue the registration command (REGISTER). This might be necessary for the operator to send and receive data from other management services applications using operations management. The operator can also use the REGISTER QUERY command to display a list of registered applications.

NetView System Programmer

To implement an operations management served application in the NetView program, a system programmer performs the following tasks:

1. Creates an operations management served application. Writes command processors to send and receive data. You can use one command processor to handle both sending and receiving. You can access the interfaces to send data from high-level language (HLL) or assembler command processors, or access the interfaces to receive data from HLL, assembler, or REXX command lists.
2. Registers the operations management served application. Either the macro interface in a command processor or the command interface can be used.
3. Determines whether the application uses the NetView destination LU name or the VTAM CP name as the destination name in MDS send requests.
4. Uses the CNMSMU (CNMSENDMU) service routine in PL/I and C or the DSI6SNDS macro in assembler interfaces to perform both of the following actions:
   a. Send requests to other management services applications or operations management served applications in the same node or a different node.
   b. Send replies to requests received from other management services applications or operations management served applications.
5. Deregisters the operations management served application.

Either the macro interface in a command processor or the command interface can be used. Do this when the task ends or when you no longer want to send or receive data for that management services category.

Other System Programmer

Operations management does not have system interfaces that are not NetView system interfaces other than those that are specified for the MS transport (see “Other System Programmer” on page 54). However, the data must be in the form of an MDS-MU containing a CP-MSU with an R&TI.

Operations Management Routing Considerations

Application designers should be aware of the following operations management routing considerations:

- Operations management can have two defined application names:
  - At initialization time, operations management registers itself as an entry point, X'23F0F1F6'.
  - If operations management is also a focal point, X'23F0F1F7' is also registered.

  Unless you are sure that operations management is a focal point, use X'23F0F1F6' when sending or building an MDS-MU. Your code can properly process MDS-MUs with either name in them.

  The operations management name is contained in the origin or destination application field of the X'1311' GDS variable contained in the MDS-MU header. See “MDS Routing Information (X'1311') GDS Variable” on page 94 for more information about the MDS-MU header.
- Unsolicited data is routed by operations management to the task that is specified in the destination instance identifier. If the destination instance identifier is not present or is inactive, the request is sent to the task from which the registration macro or command defining the served application was issued. If the registration task is not active, a routing report is returned.
- For replies, the destination instance identifier is not used even if it is present. Replies always go back to the task issuing the send request.
- For routing reports that are not replies, the report is sent to the task that issued the send request that generated the routing report, unless operations management has purged its internal representation of the original request because of elapsed time. If this has happened, replies are sent to the destination instance identifier if it is present. If this task is not present or is inactive, the routing report goes to the registration task. If this procedure fails, an error message is logged.
- For MDS error messages, operations management sends the error message to the task that issued the request generating the error message, unless operations management has purged its representation of the request because of elapsed time. In this case, a message is logged and the MDS error message is discarded.

Chapter 8. Using NetView Web Services Gateway

NetView Web Services Gateway provides you with an industry-standard open interface into the NetView program. The following functions are provided:

- Synchronous NetView commands and responses are text based. The command input and the output is in XML format. A timeout value is used to wait for the command response. If a command fails to return data in time, an error is returned in the form of a SOAP fault.
- The transport mechanism is SOAP over HTTP/HTTPS. This provides a data encryption facility using the SSL V3 protocol. Encryption can be provided by the Application Transparent Transport Layer Security (AT-TLS) function.
- User ID and password or password phrase authentication and command authorization is done using existing NetView security functions.
- Certificate authentication is supported.
- A Web Services Definition Language (WSDL) file is provided for automatic generation of a proxy-client or to use with a Dynamic Invocation Interface (DII) client.
- NetView Web services are IPv6 enabled.

Introduction to the SOAP Client

SOAP is a communications XML-based protocol that lets applications exchange information through the internet. SOAP is platform independent and language independent. SOAP uses XML to specify a request and reply structure. It uses the HTTP protocol as the transport mechanism to drive the request and to receive a reply.

Using Web Services Gateway

The NetView program provides a SOAP transport that you can use to issue commands and receive responses.

The SOAP request contains a NetView operator ID and its password or password phrase. The operator ID and password or password phrase, and the response, can be protected by enabling the HTTPS protocol by using the Application Transparent Transport Layer Security (AT-TLS) function.

The operator ID and password or password phrase are authenticated by the NetView program or a SAF product such as RACF®. The command is run under the authority of the specified operator ID.

Making a SOAP Request

You can make a SOAP request in the following ways:

- "Using a SOAP Envelope" on page 61
- "Using a WSDL-Generated Proxy Client" on page 62
- "Using a Java SAAJ Client" on page 62
- "Using a Dynamic Invocation Interface Client" on page 62

Using a SOAP Envelope

To use a SOAP envelope:

1. Format a SOAP envelope (see "Formatting a SOAP Envelope" on page 63).
2. Open an HTTP or HTTPS socket connection with the server and deliver the payload.

The response is sent back in a SOAP envelope. If there are errors, a SOAP fault element is sent back.

Using a WSDL-Generated Proxy Client

The NetView Web Services Description Language (WSDL) files automatically generate a proxy-client connection. You can create the client for any language that supports WSDL (for example, Java™, C/C++, COBOL, C#, JavaScript, and Perl). Java users can create client-side bindings using the Axis WSDL-to-Java tool. The following example uses the basic invocation form:

```
% java org.apache.axis.wsdl.WSDL2Java (WSDL-file-URL)
```

You can use one of the following URLs depending on the output format (see Table 5 on page 62) that you use:

- http or https://*yournvhost:port*/znvwsdl.wsdl
- http or https://*yournvhost:port*/znvwsdl1.wsdl
- http or https://*yournvhost:port*/znvwsdl2.wsdl

The output generates bindings that are necessary for the client. You can also use Rational Rose® Technical Developer V7.0 to generate the proxy-client connection.

Table 5 on page 62 lists the supported output formats for the NetView WSDL files.

Table 5. XML Output Formats

URL Value	Output Format	WSDL File	Description

/znvsoa	`<resp>`
	`<dl>`
	`...`
	`line-mode-output`
	`...`
	`</dl>`
	`</resp>`
		znvwsdl.wsdl	Use to capture the output of a command that is entered from the command line. The <dl> tag encloses a single line of output.

/znvsoa1	`<xmlout>`
	`...`
	`xml document fragment`
	`...`
	`</xmlout>`
		znvwsdl1.wsdl	Use to exchange XML or HTML document fragments. Output escape mode is disabled. When an XSLT processor translates an XML or HTML string, it escapes any significant characters, for example, angle brackets (< >), quotation mark ("), and ampersand (&). Do not use this file if you expect special characters in your output or if you are using unescaped special characters. Verify that the output document fragment is well-formed.

/znvsoa2	`<xmloutp>`
	`<out>...`
	`xml document fragment`
	`...`
	`</out>`
	`</xmloutp>`
		znvwsdl2.wsd	Use to exchange XML or HTML document fragments. Output escape mode is enabled. The special charactersangle brackets (< >), quotation mark ("), and ampersand (&). are escaped in the output using <, >, &, and ". Standard Java utilities are available to convert them back to the XML specification.

Using a Java SAAJ Client

You can use Java SAAJ support to manually create SOAP envelopes to access the server.

Using a Dynamic Invocation Interface Client

You can use a Dynamic Invocation Interface (DII) client to call the service. If you use DII, you do not need access to tools such as the WSDL2Java tool or the WS compiler.

Formatting a SOAP Envelope

The SOAP requests are embedded in a standard SOAP 1.2 envelope. The envelope contains the following elements:

<SOAP-ENV:Header>
Contains NetView operator ID and password

<SOAP-ENV:Body>
Contains the SOAP method call

The SOAP envelope header contains the NetView operator ID and password as shown in Table 6 on page 63.

Table 6. SOAP Envelope Header

Supported tags	SOAP tag usage examples

<Name>
The NetView operator ID under which the request is to be run. The operator ID is defined in the DSIOPF member. The command is sent to this task to be run under the authority of this operator.
<Password>
The NetView operator password or password phrase that is defined in the DSIOPF member or using a SAF product such as RACF. Either the NetView program or RACF is used to authenticate this password or password phrase.
The IBM-037 and UTF-8 code pages are used for conversion between the host and distributed environment. Verify that passwords or password phrases are printable in both code pages. Enclose the password or password phrase in a CDATA tag if it has special characters. If you are using the HTTPS protocol, the password or password phrase is encrypted.
	`<Name>`
	`NetView User ID`
	`</Name>`
	`<Password>`
	`NetView Password`
	`</Password>`

The SOAP envelope body contains the SOAP request and its tags as shown in Table 7 on page 63.

Table 7. SOAP Envelope Body

SOAP request	Supported tags	NetView tag usage examples

NVCMD
Send a command request to the NetView program
	<cmd>
	Any valid NetView command. This includes regular command processors, but does not include full screen commands or DST commands.
	The command must be enclosed with a CDATA tag if it contains special characters, for example the ampersand (&). The command is converted to uppercase unless you precede the command with the `netvasis` command.
		`<NVCMD>`
		`<cmd>`
		`<![CDATA[`*`NV_Cmd`*`]]>`
		`</cmd>`
		`</NVCMD>`

Figure 8 on page 64 shows a sample NVCMD SOAP request. This XML stub is embedded in a standard SOAP 1.2 envelope. SOAP envelope elements are shown in bold. Input and output data are shown in italics.

```
<?xml version="1.0" encoding="EBCDIC-CP-US" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-
 envelope">
<SOAP-ENV:Header>
  <h:BasicAuth xmlns:h="http://soap-authentication.org/basic/2001/10/"
   SOAP-ENV:mustUnderstand="1">
   <Name>
    NetView_operator_ID
    </Name>
      <Password>
     NetView_Password
        </Password>
    </h:BasicAuth>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
   <NVCMD>
    <cmd>
     <![CDATA[NetView_Command]]>
     </cmd>
   </NVCMD>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 8. SOAP Request Sent to SOAP Endpoint

Figure 9 on page 64 shows the response received. The response is sent in a SOAP element enclosed by the <xmlout> tag.

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body>
  <xmlout>
       CNM353I LISTVAR : OPSYSTEM = MVS/ESACNM353I LISTVAR : MVSLEVEL =
       SP7.0.8CNM353I LISTVAR : ECVTPSEQ = 01010800CNM353I LISTVAR :
      CURSYS   = NMP119CNM353I LISTVAR : VTAMLVL  = V618CNM353I LISTVAR     :
        VTCOMPID = 5695-11701-180CNM353I LISTVAR : NetView  = IBM Z
        NetView V6R3CNM353I LISTVAR : NETID    = USIBMNTCNM353I
        LISTVAR : DOMAIN   = NTV77CNM353I LISTVAR : APPLID   =
        NTV77010CNM353I LISTVAR : OPID     = SYSADMINCNM353I LISTVAR : LU
        = SYSADMINCNM353I LISTVAR : TASK     = OSTCNM353I LISTVAR :
        NCCFCNT  = 0CNM353I LISTVAR : HCOPY    = CNM353I LISTVAR : IPV6ENV
        = MIXEDCNM353I LISTVAR : TOWERS   = MVSCMDMGT NPDA NLDM
        TCPIPCOLLECT TEMACNM353I LISTVAR : CURCONID = CNM353I LISTVAR :
        AUTCONID =
  </xmlout>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 9. SOAP Response from SOAP Endpoint

Figure 10 on page 64 shows the fault element that is received.

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <SOAP-ENV:Fault>
   <faultcode>
      Server.Fault
   </faultcode>
   <faultstring>
      String
   </faultstring>
      <detail>
    <znvFault>
      Error cnmcmd. Hlbrc=20
    </znvFault>
      </detail>
       <faultactor>""<faultactor>
  </SOAP-ENV:Fault>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 10. SOAP Fault Element

The encoding used for the SOAP envelopes is ISO-8859-1.

Output Format

The NetView Web Services server supports different XML output formats depending on the URL value supplied when connecting to the SOAP server. Including HTML in XML documents might result in parsing problems because HTML coding does not follow XML and XSL rules for well-formed documents. The XML output method uses escape characters for the ampersand (&), quotation mark ("), greater-than symbol (>), and less-than symbol (<) when outputting text nodes. For example, if output escaping is disabled, <abc>&</abc> remains intact. With output escaping enabled, the output include single quotation marks: '<abc>&</abc>'. This ensures that the output is well-formed XML. This is needed when output is not well-formed XML, for example, when the output includes ill-formed sections that are to be transformed into well-formed XML by a subsequent non-XML process.

Chapter 9. NetView High Performance Transport API

The NetView high performance transport API provides a better-performing alternative to the NetView MS transport for user-written applications that need an LU 6.2 API. The NetView high performance transport API makes it possible for applications that are supplied with the NetView product or written by users to communicate with applications in other LUs (that do or do not use the NetView program) over LU 6.2 sessions.

Communication between applications using this transport is in the form of MDS-MUs. See Appendix A, "Data Formats for LU 6.2 Conversations," on page 93 for information about the format of MDS-MUs.

NetView functions that are not performing architected management services functions also use the NetView high performance transport API.

The NetView high performance transport API acts as an interface between the application and VTAM. It establishes LU 6.2 conversations and monitors outstanding requests and timeouts. The NetView high performance transport API has the following three interfaces:

- Registration service
- Send service
- Get data facility

For additional information about the PL/I, C, and assembler interfaces refer to *IBM Z NetView Programming: PL/I and C* and *IBM Z NetView Programming: Assembler*.

Registration Service

An application uses the CNMHRGS (CNMHREGIST) service routine in PL/I and C or the DSIHREGS macro in assembler to inform the NetView high performance transport API that it is ready to send and receive data. The application specifies the following information:

- Its application name
- A command to run when data is received for that application
- The logmode the application uses

  No validity checking of the specified logmode is done, except to ensure that if it is specified on a REPLACE=YES registration, it matches the existing logmode. To change the logmode used to send data, an application deregisters and then reregisters.

  If the logmode does not exist in the VTAM logmode table, the first entry in the table is used for the default. Because VTAM overrides many of the BIND parameters, the first logmode works in setting up an LU 6.2 session. The session parameters, however, might or might not be wanted in such a case.

When using the CNMHRGS (CNMHREGIST) service routine, an application can specify whether it is suspended after it issues a send request, or, if it remains active, whether replies to the send request are buffered or immediately forwarded to the application. When using the DSI6HREGS macro, an application can specify whether replies to a send request are buffered or immediately forwarded to the application.

The application can also specify whether it wants to receive special notification of session outages at other high performance nodes.

When registering an application, the type of session outage notification it receives can be specified:

ALL
Session outage notifications are received even if the NetView program cannot determine that the outage is caused by a problem.

ERROR
Notifications are received only when the NetView program can determine that the session outage is abnormal.

NONE
No notifications are received (the default).

Session outage information is provided only when the last LU 6.2 session to a node is lost and the high performance transport has been in contact with the node using the logmode of the lost session. Non-LU 6.2 session outages do not drive the notification.

This service can also be used to inform the NetView high performance transport API that the application no longer wants to send or receive data.

Send Service

An application uses the CNMHSMU (CNMHSENDMU) service routine (in PL/I and C) or the DSIHSNDS macro (in assembler) to send data to another registered application in its own node or another node. In PL/I and C, the service is provided by the CNMHSMU (CNMHSENDMU) service routine.

When using the CNMHSMU (CNMHSENDMU) service routine, an application can specify whether it is suspended after it issues a send request, or, if it remains active, whether replies to the send request are buffered or immediately forwarded to the application. When using the DSI6SNDS macro, an application can specify whether replies to a send request are buffered or immediately forwarded to the application.

The send macro has several restrictions:

- VTAM must be active for data to be sent, even to an application within the same node.
- When data is sent within the same node, the origin and destination application names must be different.
- If you have interconnected networks, destination LU names must be unique to ensure reliable routing. If a blank NETID is supplied (or the default value is used), the NetView program fills in the correct NETID prior to sending data through the network.

Get Data Facility

To get the MDS-MU that is on the initial data queue, an application can use one of the following ways:

- Service routine CNMGETD (CNMGETDATA) for PL/I and C
- Macro DSIGETDS for assembler
- A command list written in REXX, using NetView MSU information functions such as MSUSEG. Refer to *IBM Z NetView Programming: REXX and the NetView Command List Language* for additional information.

Implementing High Performance Transport API Applications

To implement a high performance transport API application in the NetView program, system programmers must follow the instructions described in this section. You can implement the application in NetView LUs or in LUs that do not use the NetView program.

NetView System Programmer

To implement a NetView application that uses the high performance transport API, the NetView system programmer performs the following tasks:

1. Defines an application name (1–8 characters) that is used to identify this application. The name can consist of characters A - Z (uppercase only) and 0 - 9.
2. Creates an application by writing command processors to send and receive data using that application identification.

   Consider the following guidelines:

   - One command processor can handle both sending and receiving.
   - The interfaces to send data can be accessed from an HLL or assembler command processor.
   - The interfaces to receive data can be accessed from an HLL, assembler, or REXX command list.

- As part of designing the application, also consider the tasking structure of the application. For more information about the application's tasking structure, see "Tasking Structure" on page 41.

3. Registers the application using the REGISTER command or the application name registration services API from a command processor.

   The application name registration service API is described in *IBM Z NetView Programming: PL/I and C*.

   When registering the application, decide which logmode the application is to use:

   - Use an existing logmode, if the application can share its data transmissions with other applications.

     All applications using the same logmode at any given time use the same conversations and the same path to send data to a given LU. While one application is sending data, another application using the same logmode is queued until the first send request is processed.

     A logmode to which more than one application is registered is driven by the X'08A80012' sense code in case of session failure.

     For example, the RMTCMD function uses the PARALLEL logmode. Other applications registering with the high performance transport should not register under the PARALLEL logmode, unless it is acceptable for both RMTCMD and the user-written application to be driven by the X'08A80012' sense code if either experiences a session failure.

     Note: To determine the logmode with which an application was registered, use the REGISTER QUERY command.

   - Use a logmode that other applications do not use, if the application has a lot of traffic and you want to prevent interferences with other applications.

     Using a unique logmode for each application guarantees that the application is driven with the X'08A80012' sense code only when that application has experienced a session failure.

     A different logmode enables the following actions:

     – The data can be sent on different paths, depending on the logmode to class of service (COS) to path mapping that is performed during the VTAM definition.
     – System programmers can define unique system parameters for sessions that the application uses, including maximum RU size.
     – The application can send data without waiting for other sends to complete, except for sends from the same application.

     Note: If you copy and rename an existing logmode, the logmode is not driven by insignificant X'08A80012' sense codes.

4. Use the CNMSMU (CNMSENDMU) service routine in PL/I and C or the DSI6SNDS macro in assembler to:

   a. Send requests to other management services applications in the same node or a different node.
   b. Send replies to requests received from other management services applications.

5. Receive data from the initial data queue using the CNMGETD (CNMGETDATA) or DSIGETDS interface.

   This data can be:

   - An MDS-MU that other applications sent to this application as a request.
   - A reply to a request that this application generated.
   - Error data in one of the following forms:

     – An MDS error message to specify that a particular piece of data identified by the agent unit of work correlator (UOWC) was not sent.
     – A generic error message with a correlator supplied by the MDS router.
     – A generic error message with a sense code of X'08A80012' indicating that a connectivity problem has occurred with the LU specified in the SNA condition report over the mode the application is

using. The message does not indicate whether the error has affected data the application might have previously sent.

Note: If multiple applications share the same logmode, both receive the X'08A80012' error message whenever a problem occurs. This happens regardless of which application caused the problem during a send. The application must determine if the outage is significant.

6. Deregister the application using the macro interface in a command list when you no longer want to send or receive data with this application or when the default receiving task ends.

   When an application is deregistered, any outstanding send requests expecting a reply are canceled, and MDS error messages are sent to the other applications involved in the transaction. This occurs even for send requests originating under a task other than the registered task. Deregistration prevents applications from sending data using the deregistered application name as the origin application.

Other System Programmer

In addition to applications running on the NetView program that use the registration service, send service, and get data facility, some applications running in LUs without the NetView program can communicate with applications running on the NetView program by using the high performance transport API. For this communication, the LU that runs without the NetView program must implement the MDS_HP_RECEIVE transaction program and send functions similar to those in the high performance transport API.

For more information about communications between the NetView program and other programs and the differences between the MS transport and the high performance transport, see “Other System Programmer” on page 54 and “Differences between Transports” on page 39.

Maintaining Data Integrity

If you want to maintain data integrity while using the high performance transport API, use the following techniques:

- Put a sequence number on every flow between two applications. If you do this, the receiving application recognizes whether a number is missing and whether an error occurred. After detecting the error, the two applications can resynchronize.
- Use a start and end message. For example, if the sending application recognizes that it has 50 pieces of data to send, it can:
  - Send a start message indicating that 50 pieces of data exist.
  - Send the 50 pieces of data.
  - Send an end message indicating that the transfer is complete.

  If the application receives start and end messages, but does not receive 50 pieces of data, it recognizes that an error occurred. If the application receives another start message before an end message, it means that an error occurred because the end message is missing from the previous send.

  If the application receives an end message followed by more data, an error has occurred.
- Specify a time interval within which a response must be received (assuming the request required one). If this time interval passes, the API cancels the request and sends an MDS error message to the other node to specify that the UOWC is no longer outstanding and no reply is required. The MDS error message, which is also sent to the application that generated the request, has a sense code of X'08A90003'.

  You can use the SENSE command to display the meaning of a sense code. Refer to NetView online help for more information about the SENSE command.

Chapter 10. Programming Techniques

This chapter describes programming techniques and provides pseudocode examples explaining how to use the program-to-program interface (PPI). You can transport these pseudocode examples to other operating systems.

Writing Effective Programs

Use the following information to write programs that use the PPI most effectively:

- Ensure that you have completed the steps under "Receiving Alerts" on page 11.
- When building the NMVT buffer, do not skip any bytes. All fields must be adjacent. In some cases, you might need to change the declarations for a field so that boundaries do not cause bytes to be skipped. For example, in many languages, a declaration for an integer causes the storage area assigned for the integer to begin on a word or halfword boundary. This can cause bytes to be skipped. In that case, declare the variable as a different type (such as character) that does not cause bytes to be skipped.
- If necessary, reinitialize variables that have been written over during a previous call to the interface.
- Some of the fields in the request parameter block (RPB) overlap other fields (for example, ASCB-ADR and ECB-ADR overlap SENDER-ID). If your program makes multiple calls to the NetView program, you might need to reinitialize some of these overlapping fields.
- If you use PL/I, refer to *IBM Z NetView Programming: PL/I and C*.
- The PPI runs synchronously with any application that makes a call to the PPI. The PPI resides in the NetView subsystem address space. After a call is made, control is passed to the PPI. The NetView subsystem address space must be active for control to be passed successfully to the PPI.
- When the NetView subsystem address space that you specified with a PPI option is inactive, the PPI is inactive. When the subsystem address space is stopped and restarted and the PPI is started, the receivers that were previously defined in the PPI are still defined, but the receiver's buffer queue is lost.
- Only one PPI is allowed in a host subsystem. It is contained in the subsystem interface address space you specify.
- When a task that has a registered receiver ends or abnormally ends, the PPI is notified and sets the receiver to inactive.
- When a return code of 30 (no buffer available) is generated from request type 22 (receive a buffer), the address of the receive ECB in bytes 20-23 of the RPB is also returned. This is the ECB returned in the ECB-ADR of the RPB from request type 4 (define and initialize a receiver).

Note: You can use the 40-byte or 52-byte RPB.

Using Regular Expressions for Advanced Data Processing

The NetView program provides basic support for the use of regular expressions to process string data. Regular expressions can be used in any of the following areas:

- The LOCATE and NLOCATE stages of the PIPE command
- The built-in NetView REXX function MATCH
- The built-in Automation Table Function DSIAMMCH

Introduction to a Regular Expression

A regular expression is a pattern that abstractly describes the desired format of a string. In this way, regular expressions are more powerful than a substring, and can describe many different elements that

resemble character data. You might use a regular expression to filter data from a file, or validate user inputs to a program.

The basic elements of a regular expression are its ***tokens*** and ***quantifiers***. A ***token*** describes what character, or type of character, should appear in the string. A ***quantifier*** describes how many of that token there should be. Quantifiers can be exact numbers or ranges of numbers.

By default, a string matches a regular expression if the match occurs anywhere in the string. However, you can confine the match to the beginning, end, or entirety of the string, if desired.

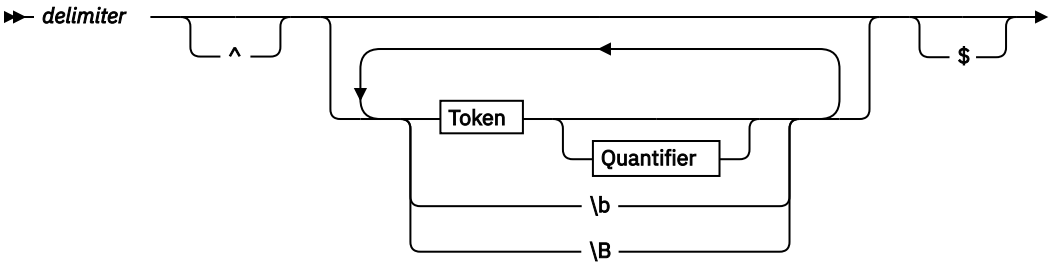
The maximum length of a regular expression, including its delimiters and options, is 255 characters.

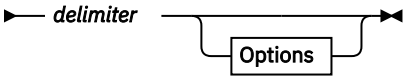
The following tables appear at the end of this section, and describe different types of characters:

- Table 1: Invalid delimiters
- Table 2: Word characters
- Table 3: Whitespace characters
- Table 4: Reserved characters

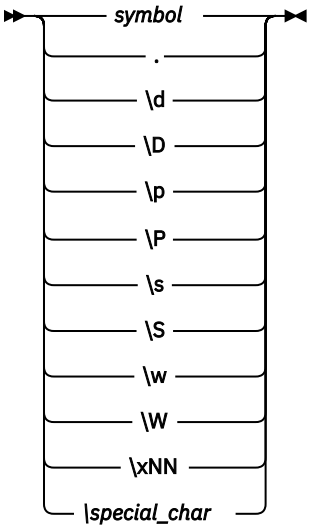
Syntax of a Regular Expression

A regular expression has the following syntax:

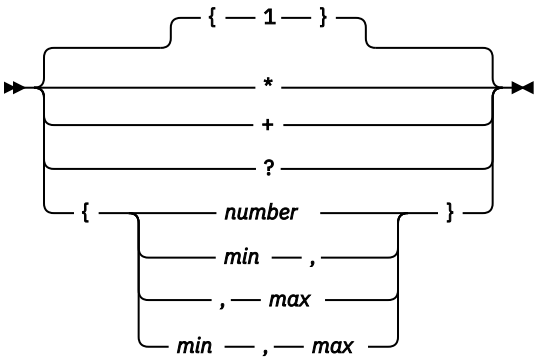




Token



Quantifier



Options

Operand Descriptions

delimiter
Specifies the regular expression delimiter. A regular expression must start and end with the same delimiter character. SeeTable 1 for a list of invalid delimiters.

^
The caret (x'B0') is a positional anchor that confines the regular expression to the beginning of the string. If specified, the string is only considered a match if the match occurs at the beginning of the string. Otherwise, the match fails, even if it occurs elsewhere.

$
The dollar sign (x'5B') is a positional anchor that confines the regular expression to the end of the string. If specified, the string is only considered a match if the match occurs at the end of the string. Otherwise, the match fails, even if it occurs elsewhere.

\b
An escape sequence that represents a word boundary. See the Word Boundaries section for more information about this escape sequence.

\B
An escape sequence that represents a non-word boundary. See the Word Boundaries section for more information about this escape sequence.

Token

.
The period (x'4B') is a wildcard that represents any character, including unprintable characters.

\d
An escape sequence that represents a digit from 0 to 9 (x'F0' to x'F9'), inclusive.

\D
An escape sequence that represents a non-digit: any character other than 0 to 9 (x'F0' to x'F9'), inclusive.

\p
An escape sequence that represents a lowercase letter from a to z (x'81' to x'A9'), inclusive.

\P
An escape sequence that represents an uppercase letter from A to Z (x'C1' to x'E9'), inclusive.

\s
An escape sequence that represents a whitespace character. See Table 3 for a list of whitespace characters.

\S
An escape sequence that represents a non-whitespace character. See Table 3 for a list of whitespace characters.

\w
An escape sequence that represents a word character. See Table 2 for a list of word characters.

Note that this escape sequence refers to word characters, not full words.

\W
An escape sequence that represents a non-word character. See Table 2 for a list of word characters.

Note that this escape sequence refers to word characters, not full words.

\x*NN*
An escape sequence that represents a single-byte EBCDIC hexadecimal character. If this escape sequence is coded, it must be followed by exactly 2 hexadecimal digits (for example, \x0D for a carriage return).

special_char
An escape sequence that represents an escaped character literal for a reserved character. See Table 4 for a list of reserved characters.

symbol
Specifies a single character. A symbol can be any character that you can type.

Note that if the symbol is a reserved character, it must be escaped using the backslash (x'E0'). See Table 4 for a list of reserved characters.

Quantifier

*****
Specifies zero or more of the preceding tokens.

+
Specifies one or more of the preceding tokens.

?
Specifies zero or one of the preceding token.

{*number*}
Specifies exactly number occurrences of the preceding tokens.

{*min*,}
Specifies at least *min* occurrences of the preceding tokens.

{,*max*}
Specifies no more than *max* occurrences of the preceding tokens.

{*min*,*max*}
Specifies at least *min*, but no more than *max*, occurrences of the preceding tokens. *min* must be less than or equal to *max*.

Options

i
Specifies that all character comparisons should be case-insensitive.

Usage Notes

- A regular expression is nothing more than a string that is processed in a special way by the NetView program. As such, regular expressions must always appear in the form of strings. Inside the string, the regular expression must be enclosed within its own set of delimiters, as matching options may appear after the ending delimiter.
- The simplest regular expression is 2 delimiters together, such as, //. This regular expression matches only the empty string.
- The backslash (x'E0') is always the escape character. In addition to coding the built-in escape sequences, you can use the backslash to specify escaped character literals for reserved characters, including the backslash itself. See Table 4 for a list of reserved characters.
  - If your regular expression contains the delimiter character as part of the expression, you can also escape it with the backslash, or choose a different character as the delimiter.

- Each quantifier, if specified, corresponds only to the token that appears immediately before it. In other words, quantifiers other than the default {1} must be coded for each token that is desired.
  - For example, the regular expression /abc{3}/ matches the string “abccc”, but not the string “aaabbbccc”.

Word Boundaries

A word boundary is a zero-width test between two characters. To pass the test, there must be a word character on one side, and a non-word character on the other side. It does not matter which side each character appears on, but there must be one of each.

Table 2 defines word characters. For the word boundary test only, which must always have two characters to consider, the beginning and end of the string are considered non-word characters.

To better understand the concept of a word boundary, consider the string:

```
HELLO WORLD!
```

Now, place an imaginary line between any two characters. These two characters, on either side of the line, are now the candidates for the test. In this example, the imaginary line is represented by a vertical bar between the letters W and O. Both characters are underlined for emphasis.

```
HELLO W|ORLD!
```

Because the characters W and O are both word characters, the imaginary line is not on a word boundary, and the test fails.

Now, place the imaginary line at the very beginning of the string:

```
|HELLO WORLD!
```

For this test, there is only one real character, H, so the beginning of the string counts as the second character. Since H is a word character, and the beginning of the string is considered a non-word character (for the word boundary test only), the imaginary line is on a word boundary, and the test passes.

Character Tables

All ranges in the following tables are inclusive. If N/A appears in the Symbol column, it corresponds to a non-displayable character.

Table 8. Invalid delimiters

EBCDIC Byte Value or Range	Symbol or Range	Description
x'00'	N/A	Null
x'05'	N/A	Horizontal tab
x'0B'	N/A	Vertical tab
x'0C'	N/A	Form feed
x'0D'	N/A	Carriage return
x'15'	N/A	Newline
x'25'	N/A	Line feed
x'40'	N/A	Blank
x'4B'	.	Period

Table 8. Invalid delimiters (continued)		
EBCDIC Byte Value or Range	**Symbol or Range**	**Description**
x'4D'	(	Left parenthesis
x'4E'	+	Plus sign
x'4F'	|	Vertical bar
x'5B'	$	Dollar sign
x'5C'	*	Asterisk
x'5D'	)	Right parenthesis
x'60'	-	Minus sign
x'6B'	,	Comma
x'6F'	?	Question mark
x'7B'	#	Pound sign
x'7C'	@	At sign
x'81' to x'89'	a – i	Lowercase letters from a to i
x'91' to x'99'	j - r	Lowercase letters from j to r
x'A2' to x'A9'	s - z	Lowercase letters from s to z
x'B0'	^	Caret
x'BA'	[	Left square bracket
x'BB'	]	Right square bracket
x'C0'	{	Left curly brace
x'C1' to x'C9'	A - I	Uppercase letters from A to I
x'D1' to x'D9'	J - R	Uppercase letters from J to R
x'E2' to x'E9'	S - Z	Uppercase letters from S to Z
x'D0'	}	Right curly brace
x'E0'	\	Backslash
x'F0' to x'F9'	0 – 9	Digits from 0 to 9

Table 9. Word characters		
EBCDIC Byte Value or Range	**Symbol or Range**	**Description**
x'81' to x'89'	a – i	Lowercase letters from a to i
x'91' to x'99'	j - r	Lowercase letters from j to r
x'A2' to x'A9'	s - z	Lowercase letters from s to z
x'C1' to x'C9'	A - I	Uppercase letters from A to I
x'D1' to x'D9'	J - R	Uppercase letters from J to R
x'E2' to x'E9'	S - Z	Uppercase letters from S to Z
x'F0' to x'F9'	0 – 9	Digits from 0 to 9
x'6D'	_	Underscore

Table 10. Whitespace characters		
EBCDIC Byte Value	**Symbol**	**Description**
x'05'	N/A	Horizontal tab
x'0B'	N/A	Vertical tab
x'0C'	N/A	Form feed
x'0D'	N/A	Carriage return
x'15'	N/A	Newline
x'25'	N/A	Line feed
x'40'	N/A	Blank

Table 11. Reserved characters		
EBCDIC Byte Value	**Symbol**	**Description**
x'4B'	.	Period
x'4D'	(	Left parenthesis
x'4E'	+	Plus sign
x'4F'	|	Vertical bar
x'5B'	$	Dollar sign
x'5C'	*	Asterisk
x'5D'	)	Right parenthesis
x'60'	-	Minus sign
x'6F'	?	Question mark
x'B0'	^	Caret
x'BA'	[	Left square bracket
x'BB'	]	Right square bracket
x'C0'	{	Left curly brace
x'D0'	}	Right curly brace
x'E0'	\	Backslash

Examples

Example: Filtering data from a NetView command

Suppose you want to list all of the default settings of your NetView instance whose values are only numbers, including those values with a percent sign. To accomplish this, issue the following PIPE command:

```
PIPE NETVIEW LIST DEFAULTS
| SEPARATE
| STRIP
| LOCATE REGEX '/\d+%?$/'
| CONSOLE
```

This PIPE performs the following actions:

SEPARATE
The output of the LIST DEFAULTS command is a multiline message, which must be separated into individual messages.

STRIP
The output also contains leading and trailing blanks, which are removed by the STRIP stage.

LOCATE
Selects only messages whose contents end with one or more digits, and an optional (zero or one) percent sign.

Note that the regular expression itself is just a string, which is coded as part of the LOCATE stage specification. The entire string is enclosed within single quotation marks, and the regular expression is enclosed between two forward slashes.

This regular expression is interpreted as follows:

- One or more digits, followed by
- An optional percent sign (zero or one)
- At the end of the string

Example: Validating user input

Suppose you have the following exec called EMAILVFY, which determines the validity of a user-specified email address:

```
/* EMAILVFY EXEC */
ARG theEmail
/*--------------------------------------------------------------------*/
/* This is a very simple regular expression for an email address. */
/* Email addresses can of course be more complex than this, but we */
/* use this one for the purposes of this example. */
/*--------------------------------------------------------------------*/
IF MATCH('/^\w+\.?\w+@\w+\.\w+$/', theEmail) THEN
SAY "The email address is valid."
ELSE
SAY "Invalid email address!"
```

This regular expression is interpreted as follows:

- The entire string must consist of
- One or more word characters, followed by
- An optional period (zero or one), followed by
- An "at" sign, followed by
- One or more word characters, followed by
- A period, followed by
- One or more word characters

Note that the period ordinarily matches any character. Because only a literal period character is desired, it is escaped with the backslash.

Examples of email addresses that match this regular expression:

- someone@example.com
- USER_123@example.com
- john.doe@example.com

Examples of email addresses that do not match:

- firstname.@ibm.com
- .last name@ibm.com

- Email With Spaces @ ibm.com

High-Level Language and Assembler Programming Examples

The following pseudocode examples are designed to help you write code that can be easily transported from one system to another.

Initializing a Receiver

The following pseudocode example shows initializing a receiver:

```
( Fill in all required fields in the Request Parameter Block for        )
( request type 3.                                                       )

CALL CNMCNETV(RPB control block)          ( Issue request type 3.        )

IF RETCODE = 0 THEN
  save the ASCB returned from PPI        (                              )
ELSE                                     (                              )
  EXIT with error
ENDIF

( Fill in all required fields in the Request Parameter Block for request)
( request type 4.                                                       )

CALL CNMCNETV(RPB control block)         ( Issue request type 4          )

IF RETCODE = 12 THEN                    ( If the connect was delayed.   )
                                        ( Issue the wait yourself or    )
  WAIT on ECB passed back in ECB-ADR    ( issue request 24 to do it for )
                                        ( you. Either way a request 24  )
     and/or                             ( must be issued to make sure   )
                                        ( the connect completed         )
  TYPE = 24                             ( successfully.                 )

  CALL CNMCNETV(RPB control block)       ( Check to see if the connect   )
                                        ( succeeded.                    )
ENDIF

SELECT on RETCODE
 ( Take appropriate action for the return code.                         )
 ( NOTE, it is no longer necessary to save the ECB returned from the    )
 ( connect. PPI always returns the ECB to wait on whenever a wait   )
 ( is necessary.                                                        )
ENDSELECT.
```

Receiving a Buffer

The following pseudocode example shows receiving a buffer:

```
( Fill in all required fields in the Request Parameter Block for a      )
( request type 22, if the length of the incoming buffer is unknown,     )
( set the BUFFQ-L to 0.  Use same work area used for request type 4.    )

RETCODE  = 30                             ( Initialize return code    )
                                          ( to buffer available.      )

LOOP WHILE RETCODE = 30                   ( Loop until a buffer is    )
                                          ( successfully received.    )

  CALL CNMCNETV(RPB control block)          ( Issue request type 22.    )

  SELECT on RETCODE

    CASE 0
      ( A buffer was successfully received,  take appropriate action    )

    CASE 31
      ( The buffer length was too small, DTRUBL will be filled in with  )
      ( the length of the incoming buffer.  If DTRUBL was set to zero,  )
      ( this will be the return code sent back.                         )

      GET STORAGE for the number of bytes returned in DTRUBL

    CASE 30
```

```
      ( No buffer to receive. Issue a wait yourself or issue a request  )
      ( type 24 to do the wait for you.                                 )

      WAIT on ECB returned in ECB-ADR
          or
      TYPE = 24

      CALL CNMCNETV(RPB control block)

    OTHERWISE
      ( An error occurred, take appropriate action.                     )

  ENDSELECT
ENDWHILE
```

Sending a Buffer Synchronously

The following pseudocode example shows sending a buffer synchronously:

```
( Fill in all required fields in the Request Parameter Block for        )
( request type 12 or 14.                                                )

CALL CNMCNETV(RPB control block)           ( Issue request type 12 or 14.)

SELECT on RETCODE

  CASE 0
     ( Send successfully completed, take appropriate action.            )

  OTHERWISE
     ( An error occurred, take appropriate action.                      )

ENDSELECT
```

Disconnecting a Receiver

The following pseudocode example shows disconnecting a receiver:

```
( Fill in all required fields in the Request Parameter Block for        )
( request type 9, restore ASCB-ADR field for MVS.  Use same work area   )
( that was used for request type 4.                                     )

CALL CNMCNETV(RPB control block)             ( Issue request type 9.     )

SELECT on RETCODE

  CASE 0
     ( Disconnect successfully completed, take appropriate action.      )

  OTHERWISE
     ( An error occurred, take appropriate action.                      )

ENDSELECT
```

REXX Programming Examples

The following pseudocode examples illustrate the usage of the DSIPHONE REXX external routine.

Usage Scenario

Two application programs running anywhere that TSO REXX runtime environments are available, within the same MVS image, can send data to and receive data from each other using a PPI receiver name known to both applications.

The following example shows a server/client application:

Client application

```
/* REXX */
/* Define a new PPI receiver and name it CLIENT */
Call DSIPHONE 'OPENRECV', 'CLIENT'
```

```
command = 'Some commmand to be processed.....'
/* Send command to PPI receiver, SERVER,  */
/* specifying the sender's name is CLIENT */
call DSIPHONE 'SEND', 'SERVER', 'COMMAND', 'CLIENT'
/* Wait (forever) for data to arrive on our PPI receiver.   */
/* When it arrives, receive it into a stem called output.   */
/* but only if the sender's receiver name is SERVER.  Store */
/* the SAF userid of the sender in REXX variable, safuid    */
call DSIPHONE 'RECEIVE', 'CLIENT', 'OUTPUT.',,,,'SAFUID','SERVER'
 if safuid = 'expected SAF userid for SERVER' then
  do
   /* process output. stem */
  end
 else /* this data is not from whom we expected it to be from */
  nop
/* Delete the PPI receiver named CLIENT */
call DSIPHONE 'CLOSE', 'CLIENT'
exit 0
```

Server application

```
/* REXX */
/* Define a new PPI receiver and name it SERVER */
call DSIPHONE 'OPENRECV', 'SERVER'
do forever
 /* Wait (forever) for data to arrive on receiver SERVER.  Receive  */
 /* each buffer into REXX variable, cmd.  Put the sender's receiver */
 /* name into REXX variable, clientReceiver                         */
 call DSIPHONE 'RECEIVE', 'SERVER', 'CMD', 'CLIENT_RECEIVER'
 /* process command and put the result into the REXX stem, output.  */

 /* send stem output. to the sender's receiver name */
 call DSIPHONE 'SEND', client_receiver, 'OUTPUT.', 'SERVER'
end
exit
```

Common Operations Services Commands

The common operations services (COS) commands support and enhance the NetView program control of service points. A service point application manages non-SNA devices, such as front-end line switches and multiplexers. You can send commands to the service point application to do problem determination for the non-SNA devices.

You can use the following NetView COS commands with service points for problem determination. Refer to NetView online help for the format of the COS commands.

RUNCMD

Sends service point application commands to the service point applications. Replies are returned to the operator or the command list issuing RUNCMD.

As part of its outgoing record, RUNCMD builds an unformatted subvector 31. This unformatted subvector contains no subfields, in deviation of the current architecture. The incoming reply can be formatted or unformatted.

The COS commands are long-running commands that suspend the command list when they are run. The command list resumes when the COS command is completed. When the command is completed, a return code is set.

For more information about the COS commands, refer to *IBM Z NetView Programming: REXX and the NetView Command List Language*. For detailed information about the vector and SNA formats used by COS, refer to the following publications:

- *SNA Management Services Reference*
- *Systems Network Architecture Formats*

COS Command Flow

COS commands can flow over the communication network management interface (CNMI) or the management services (MS) transport. The DSIGDS task registers with the MS transport as SPCS (COS_NETOP) to enable COS commands to flow over the MS API.

When commands flow over the MS API, the major vector is placed inside the X'1212' GDS variable of the control point management services unit (CP-MSU) of a multiple-domain support message unit (MDS-MU). The SP parameter of the RUNCMD command is the destination LU name, and EP_COS is the destination application name.

If the NET parameter is not specified in a COS command, the NetView program checks the CNMI configuration table for the service point (SP) name. If the NetView program finds the SP name, the CNMI transport is used; if not, the MS transport is used. If the MS transport fails with SNA sense category and modifier X'08A8' or X'08A9', a retry takes place on the CNMI. If the MS transport attempt fails for any other reason, no retry occurs and a message is issued.

If the CNMI request succeeds, the SP name is placed in the COS configuration table. You can use the PURGE command to purge the table. You can use the DEFAULTS command to set the timeout value of COS commands flowing over the CNMI and MS transport. Refer to NetView online help for more information about the PURGE and DEFAULTS commands.

If replies to RUNCMD have more than 32 KB of data, the issuer of the RUNCMD receives message `DSI296 INVALID DATA RECEIVED`. No notification is sent to the Service Point Application indicating the reply was rejected.

The maximum size for a CP_MSU is 31743 (X'7BFF') bytes. The DSIGDS task can handle 32 KB of data, but because the service point application might not be aware of the transport being used, use the smaller maximum reply size of 31743 bytes.

Message to Operator

The message to operator (MTO) sends unsolicited messages to a NetView operator console. MTO is serviced by the DSIGDS task. You can use automation to trap the messages.

The NetView program uses the X'50' subfield of the X'06' subvector of the X'006F' major vector to route the MTO to the NetView operator console. All MTOs are sent to automation in the native MSU format. Note that some MTOs arrive enveloped in a MDS-MU across LU 6.2 sessions, whereas other MTOs arrive enveloped in NMVTs across SSCP-PU sessions. Regardless of the envelope, all MTOs are first sent to automation as an MSU (MDS-MU or NMVT). If a match for the MSU MTO exists in the automation table, the MTO is not displayed to the NetView operator's console. If *no* a match for the MSU MTO exists in the automation table, the MTO is sent to automation as a series of single line messages. After the message automation process, the single line messages are sent to the NetView operator console.

Using COS Command Lists

The NetView program provides command lists that issue COS commands. You can modify these command lists for your particular application or use them for your own command lists. The following list gives the names and descriptions of each of these command lists:

Name
Description

INITCNFG
Contains the service point resource information.

The configuration defined in the INITCNFG command list must match the configuration of the lines controlled by your service points. Consider adding the INITCNFG command list to the CNMSTUSR or C*xx*STGEN member so that it runs automatically. For information about changing CNMSTYLE statements, see *IBM Z NetView Installation: Getting Started*.

Note: This command list does not work if you use lowercase values.

You can modify the following fields to contain the configuration of all the lines you control with the COS commands:

LINE
Line name

SP
Service point name

APPL
Application name

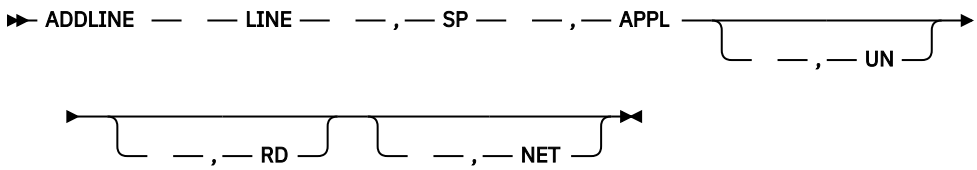
UN
Using- node name

RD
Remote device name

NET
Network name

ADDLINE
Is used to define common global variables to the SPLOOKUP command list for a new line. The syntax is:

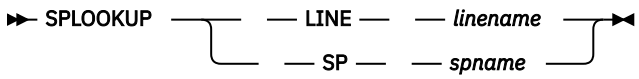
ADDLINE



SPLOOKUP
Eables you to issue COS commands for a given line. SPLOOKUP determines the parameters needed to issue a COS command for a line using the global variables set up by INITCNFG.

If SPLOOKUP is called with the LINE option, it returns the SP name, the APPL name, using-node, and remote device for the specified line in task global variables SP, APPL, UN, RD, and NET. If called with the SP option, SPLOOKUP returns a list of lines defined to the specified SP name in the task global variables LINECNT, LINE1, LINE2, and so on. The syntax is:

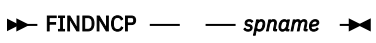
SPLOOKUP



FINDNCP
Displays events that the NetView program receives from the using-node for a given service point. This command list is useful if you are having problems getting data back from a service point.

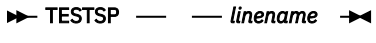
FINDNCP looks up the using-nodes for a given SP. If multiple using-nodes are defined, they are listed. If only one using-node is defined, FINDNCP calls the Most Recent Events panel of the hardware monitor to show any events for the using-node. Usually, the using-node is the NCP linked to the SP. The syntax is:

FINDNCP



TESTSP
Is used to issue COS commands for a given line. TESTSP calls SPLOOKUP, issues the COS commands, and displays the results. You can use TESTSP as a base for command lists that run COS commands automatically. The syntax is:

TESTSP



TESTRCMD

Tests the use of RUNCMD with the CLISTVAR keyword. This command list issues RUNCMD to run a command at the service point. Results are stored in command list variables, and the data is displayed when control is returned. The syntax is:

TESTRCMD

Where:

command

Is the command to run at the service point.

If you do not specify the *net* variable, the system defaults to the local network name. The command must be in single quotation marks.

Chapter 11. Using the NetView REST Server

The NetView REST Server makes it possible for users to send requests to, and receive data back from the NetView program by using a set of Application Programming Interfaces, or APIs. These APIs are each assigned a specific task or set of tasks with the exception of the General NetView Command API.

After following *Configuring and Running NetView REST Server* in the IBM Z NetView Installation: Configuring Additional Components to enable the server, the APIs are available for use. If you have Zowe™ installed, the sample Zowe application that NetView provides makes use of the APIs. For more information, see *Installing Zowe™ Sample Application* in *Configuring and Running NetView REST Server* in the IBM Z NetView Installation: Configuring Additional Components. The sample introduces use of the APIs and the kind of applications are possible. In addition to the sample, the NetView information center and NetView-supplied Swagger documentation for the APIs is available. The Swagger documentation can be reached at:

```
https://[server domain]:[NetView Server Port]/swagger-ui.html
```

From there, you can view the APIs that have been provided, test them, and review the documentation that shows how and when to use them. Some specific features on that page that can be helpful:

- Near the top of the Swagger page is a hyperlink to a JSON copy of the API documentation that contains more detailed information than the Swagger page. The hyper link will be in the following format:

  ```
  https://[server domain]:[NetView Server Port]/v2/api-docs?group=NetView-api-1.0
  ```

- Each group of APIs will have its own drop-down menu. You can open these drop-downs to see the individual APIs belonging to that group. The APIs are sorted by their URIs.
- For each individual API, you can click to open them, which will show you with the API description, the parameters it expects, links to drive the API for test purposes, descriptions of the response codes you can expect, and the structure of the payload being sent back.
- Near the bottom of the page there is a drop-down menu called **NetView** that contains all of the APIs in one menu.
- There is a **Models** drop-down menu at the bottom of the Swagger page. If you click the Models box to expand it, you can find all of the objects that have been created as models for the NetView REST APIs. This shows the kind of data sent for a request and the contents of the response.
- Before trying the APIs, you must run the login request API with valid credentials.

When logged in, you are able to begin building NetView APIs into your own standalone applications as well as Zowe™ applications.

Description of Available APIs

Following are the provided APIs. Each API shows the a brief description plus any required parameters needed. Further information can be found in the Swagger Documentation. In particular, information regarding the format of the JSON objects being sent and received by the server is documented there.

- Automation
- Canzlog
- Command
- Enterprise
- Network
- Task

Automation

- Create the Automation Table Statement

  Create Automation Table statement based on input JSON. This request will be accompanied by a JSON object that will contain information to build the automation table statement. The format of this JSON must match the format presented in the Swagger Documentation.

- Syntax Check the Statement

  After an automation table statement has been created, the Validation API allows users to ensure that it has been created 100% correctly. Under the hood, the server will create an INSTORE member in NetView and run an AUTOTBL TEST command on that member. All messages related to this action will be returned so that a user will know exactly what is wrong with their statement if the validation fails.

- Test the statement
  - This function replays the recorded Canzlog message with the same message and its attributes. The API required inputs are as follows:
    - Member name to which the messages was recorded
    - Automation Table Statement
  - The command being driven is AUTOTEST. Please note that no actions are taken on the system, it is all simulated through AUTOTEST command.

- Save the Statement

  Once a user has created a statement which is valid and passes testing, they are able to save the statement in a member. You will need to provide the API with a fully-qualified data set name as well as the statement you wish to save. The API will also allow you to specify whether you'd like to append your statements to the member or replace it entirely with your new statement.

- Enable the Statement

  Insert the automation table member into the list of other active automation table members. This utilizes the AUTOTBL command. The request needs to be accompanied by a JSON that contains the name of a data set that holds the automation table statement that the user has built using the previous APIs. You are also able to specify the position you wish to insert the automation table member at in the list.

- Retrieve the Statement

  This API retrieves a saved automation table member. This allows a user to edit the contents that they've retrieved and then call the Save API to append or replace it.

- Retrieve Automation Table Statistics

  This API retrieves data from the NetView program about the statistics of automation tables and how they are processing message-type (MSG-type) automation statements and Management Services Unit type (MSU-type) automation statements. This API has the following parameters in addition to the normal domain parameter:

  Report Type
  The type of report desired, either MSG, MSU, or both.

  Table Name
  The name of the table that the request is being sent to.

Canzlog

- Retrieve a Canzlog Message
  - Retrieve a message or set of messages from Canzlog. With this URI, only certain parameters should be used based on the information that the request is supplying.
    - For CZID based requests, parameters supplied should be as follows:
      - Requesttype=CZID
      - czid={CZID}

    - nummessages (Optional, default value is 1.)
    - direction (Optional, default value is forward.)
  - For requests that are grabbing the most recent messages, parameters supplied should be as follows:
    - requestType = recent
    - numMessages = (Optional, default is 1)
  - For timestamp-based requests, parameters supplied should be as follows:
    - requesttype=timestamp
    - begin={timestamp} (Format is mm/dd/yy hh:mm:ss.)
    - nummessages (Optional, default value is 1.)
    - direction (Optional, default value is forward.)
  - For time range based requests, parameters supplied should be:
    - requesttype=timerange
    - begin={timestamp} (Format is mm/dd/yy hh:mm:ss.)
    - end={timestamp} (Format is mm/dd/yy hh:mm:ss.)
- The server will interpret the parameters and construct a command to be sent to NetView. It is important that only the required parameters are filled out for the request type the user is sending. Results will not be valid if incorrect parameters are given, assuming the request can be completed.

- Record a Canzlog Message

  Store the Canzlog message and its message attributes in the topmost concatenated DSIASRC DD data set. The recording contains all relevant information about the message that can be retrieved from Canzlog, and much of it is in a non-readable format. This step should be done as soon as the Canzlog message is successfully retrieved, as the message could become unavailable in a short amount of time.

Command

- Run a General NetView Command

  Get the response from a command you send to NetView. There is not a tailored API for every NetView command, so this API is provided as a way for customers to experiment with running NetView commands and retrieving the result of that command. There are limitations to which commands are able to run with this API, including but not limited to: Any command that provides output in a window, any command that drives output to a panel, and commands that run on a VOST. Output for commands will not be formatted in a user-friendly format, it will be provided as a JSON containing a list of strings where each string corresponds to a line of NetView output.

Enterprise

- Retrieve XCF Groups Data

  Retrieve data about which systems are configured to run with the enterprise master (ENT.SYSTEMS). The command NetView program is running is LIST STATUS=XCFGRPS, which returns a list of z/OS XCF groups in which the NetView program participates. Only active NetView instances can be retrieved by the command.

  Note: The DISCOVERY Tower must be enabled to run this API.

Network

- Retrieve DDVIPA Data

  This APIs show the percentage of distributed DVIPA connections sent to a specific TCP/IP stack on an LPAR within the sysplex. The Workload Manager recommended percentage is also displayed. The command has six optional parameters that allow filtering for the command. This would allow a user to

limit the amount of data being returned because the amount of data available can be very large. The parameters are as follows:

Begin Time
Beginning of a time range, format should be `mm/dd/yy hh:mm:ss`. STCK values are also acceptable.

DVIPA
An IP Address for the DVIPA the user is wanting to use data from.

End Time
End of a time range, format should be `mm/dd/yy hh:mm:ss`. STCK values are also acceptable.

Number of DDVIPA records to return

Number of Intervals
Indicates the number of polling intervals to return, starting from the most recent data available. This keyword can not be specified with the keyword TIME. The specified value can not be 0 or negative. If the specified value is greater than the total number of polling intervals that are available, only the available intervals will be returned.

Number of records

Port
DVIPA port number.

System
z/OS system name. Wildcard (*) supported at the end of the string.

TCP Name
TCP/IP stack name. Wildcard (*) supported at the end of the string.

This command should be run at a NetView sysplex master.

Note: Over time, there can be a large amount of data here.

- Retrieve DDVIPA Health Data

  This API retrieves data from the NetView program about DDVIPA targets as well as DDVIPA server health information. To filter the data, this API has the following parameters in addition to the normal domain parameter:

  DVIPA IP Address
  The IP address for the requested DVIPA. If no value is specified, all DVIPAs are returned.

  DVIPA Port
  The port number for the requested DVIPA. The default value is all ports. You can specify multiple port values within parentheses. If you specify multiple port values and one of the values is an asterisk, the rest of the list is ignored.

Task

- Retrieve TASKUTIL Information

  Provide the CPU utilization, storage utilization, message queue, and command running (if available) for active NetView tasks.

NetView REST Server APIs Used for Service Management Unite

Table 12. NetView REST Server APIs that Service Management Unite is using

Service Management Unite Dashboard	Comments	API in Use	NetView Command
Explore NetView Domains		NetView Configuration APIs: /ibm/netview/v1/enterprise/members	LIST STATUS=XCFGRPS
NetView Tasks		NetView Task APIs: /ibm/netview/v1/tasks/utilization NetView General Command APIs: /ibm/netview/v1/command	TASKUTIL STOP TASK=task RECYCLET TASK=task RESET TASK=task
Canzlog		NetView Canzlog APIs: /ibm/netview/v1/canzlog/message	PIPE CZR
Issue Command		NetView General Command APIs: /ibm/netview/v1/command	Any command specified
Automation Statements		NetView General Command APIs: /ibm/netview/v1/command NetView Automation APIs: /ibm/netview/v1/automation/statement/retrieval /ibm/netview/v1/automation/statement/destination	AUTOTBL STATUS AUTOTBL INSERT MEMBER=member name AUTOTBL SWAP MEMBER=member name
Create New Statement		NetView Automation APIs: /ibm/netview/v1/automation/statement/persistence /ibm/netview/v1/automation/statement/validation /ibm/netview/v1/automation/statement/test NetView Canzlog APIs /ibm/netview/v1/canzlog/message/record	AUTOTEST CZRECORD AUTOTBL TEST MEMBER=member name AUTOTEST
N/A		NetView Authentication APIs: /auth/login	N/A

Table 12. NetView REST Server APIs that Service Management Unite is using (continued)

Service Management Unite Dashboard	Comments	API in Use	NetView Command
Network Details		NetView Network APIs: /ibm/netview/v1/network/sysplex/connection/distribution	CNMSDVST

Chapter 12. Using the Trace Facility

Use the PPI trace facility to set up a trace in the PPI for either an individual receiver or for all current and future receivers. The PPI trace facility writes a trace record each time a user defines, deactivates, or deletes a receiver to the PPI and each time a buffer is sent or received.

You can use the PPI trace facility to debug problems or analyze performance. For example, if applications that use the PPI are not communicating correctly, you can enable the PPI trace facility to gather information to help determine the problem in the dialog. The trace facility creates a log of all the buffers that are sent to and received by the applications. You can use the log to verify that the correct dialog occurred between the applications or that the applications correctly received all the buffers.

As another example, you can use the log created by the PPI trace facility to analyze the performance of applications that use the PPI. If congestion occurs at the interface, the PPI trace records show how long buffers are on the PPI buffer queue before being received by the application. Using this trace information, applications can be modified to optimize the flow of information buffers.

Controlling the Trace Facility

You control the PPI trace facility with the TRACEPPI command. You can use the TRACEPPI command to start, stop, modify, or end the PPI trace facility. See NetView online help for more information about the TRACEPPI command.

When you enable a trace for the first time, you can specify whether trace information is written to internal storage or to an external data set, using the generalized trace facility (GTF).

Writing to Internal Storage

If you specify that trace information is written to internal storage, you can specify the size of the storage area for the trace information by using the SIZE parameter when you define the PPI trace. This storage is allocated from the PPI address space.

Note: If the storage area for the trace information is too large, applications using the PPI can encounter out-of-storage conditions.

The PPI address space is contained in the subsystem interface address space. You can access the PPI trace information by dumping the PPI address space and the common storage area (CSA). If the subsystem interface is brought down while the PPI has an active internal trace, the subsystem interface dumps the internal PPI trace information.

Writing to External Storage with GTF

To use the GTF option to collect PPI trace information, you need to install, activate, and enable GTF for PPI traces. If you specify the GTF option when you issue a TRACEPPI command, trace records are written to an external data set. An internal PPI trace table is not created.

When you start GTF, specify the USRP parameter and enter the PPI event ID (X'5EF'). Without the event ID, the trace enters a GTF disabled state, and all records logged during the disabled state are lost. To re-enable the GTF, stop and restart GTF with the USRP parameter and the PPI event ID (X'5EF').

If you start GTF with a user-cataloged procedure that does not have a SYSLIB DD statement, issue the system console commands shown in Table 13 on page 91.

Table 13. Starting the PPI Generalized Trace Facility

You enter...	**GTF responds...**
`START GTF,,,(MODE=EXT)`	*xx* AHL100A SPECIFY TRACE OPTIONS

Table 13. Starting the PPI Generalized Trace Facility (continued)	
You enter...	**GTF responds...**
R *xx*,TRACE=USRP,...	*yy* AHL100A SPECIFY TRACE KEYWORDS--USR=
R *yy*,USR=(5EF)	

While a GTF trace is active, errors can occur such as:

- The GTF address space becomes inactive
- The GTF buffers become full
- A paging error occurs

When an error occurs, the subsystem interface address space issues an MVS console message describing such errors, and the PPI trace is put into GTF disabled state.

For more information about GTF or GTF trace buffers, refer to the MVS library.

Monitoring the Trace Facility

You can use the DISPPI command to monitor the receivers that are traced. Output from the DISPPI command shows information about the receiver's buffer or trace status. Also, the DISPPI command displays information about the PPI trace table.

Refer to the NetView online help for the format of the DISPPI command. See the *IBM Z NetView Troubleshooting Guide* to interpret the results of the PPI trace.

Appendix A. Data Formats for LU 6.2 Conversations

This chapter describes the formats of the following principal data types used by the management services (MS) transport and its NetView applications, including the hardware monitor:

- Multiple-domain support message unit (MDS-MU) header structure
- MDS data types
- MDS error message format

Figure 11 on page 93 is a conceptual view of the format used by the MDS-MU (X'1310') general data stream (GDS) variable. The MDS-MU contains an MDS header and an application program GDS variable.

The entire MDS-MU (MDS header and application program GDS variable) must be less than 32767 (X'7FFF') bytes in length. The MDS header must be 1024 (X'400') bytes in length. The maximum size permitted for the application program GDS variable in the MDS-MU is 31743 (X'7BFF') bytes.

The application program GDS variable contains MS application program data supplied by the MS application program. It can be a control point management services unit (CP-MSU) (X'1212') GDS variable, an SNA condition report (SNACR) (X'1532') GDS variable, or some other GDS variable.

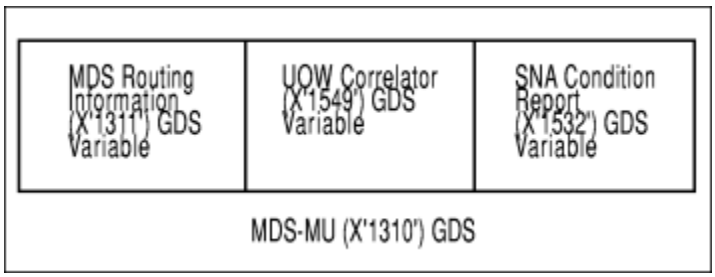


Figure 11. Format of an MDS-MU GDS

MDS Header Structure

Figure 12 on page 93 shows the format of an MDS header. The MDS header is the information used for routing between MS application programs. The MDS header is composed of two GDS variables:

- MDS routing information (X'1311') GDS variable
- Agent unit of work correlator (X'1549') GDS variable

The NetView program checks the syntax of MDS-MU headers received from other nodes. If a syntax error is found, a software alert is generated for the hardware monitor and an MDS error message is written to the system log. The MDS error message contains the first 100 bytes of the failed MDS-MU.

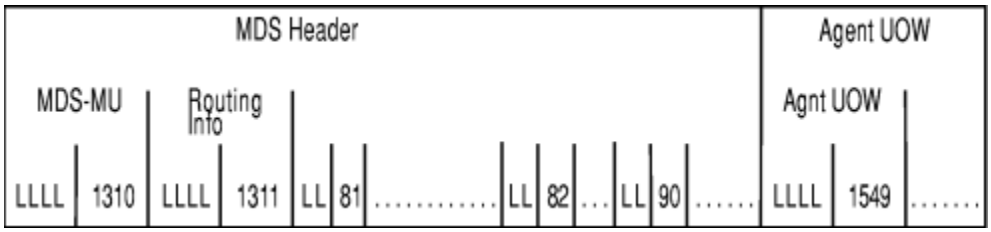


Figure 12. Format of an MDS Header

MDS Routing Information (X'1311') GDS Variable

The routing information GDS variable contains the data required for MDS routing. It consists of the following structures:

- Location names
  - Origin location name (X'81') MS subvector
  - Destination location name (X'82') MS subvector
- Flags (X'90') MS subvector

Location Names

Two location names are contained in the MDS routing information GDS variable: the name of the originating location and the name of the destination location. Each location name MS subvector consists of the network ID (NETID) of the LU, the unqualified LU name, and an MS application program name.

Create the NETID and LU names with the following characteristics:

- Use characters from the set A - Z, 0 - 9, $, #, and @.
- Alphabetic characters must be uppercase and the first character must be non-numeric.
- The characters $, #, and @ are supported for migration purposes only. Do not use them in LU or network names.
- You can include trailing blanks, but they are ignored for comparison purposes.

Origin Location Name (X'81') MS Subvector

The origin location name (X'81') MS subvector consists of the structures in Table 14 on page 94.

Table 14. Contents of Origin Location Name (X'81') MS Subvector

Structure Name	GDS/SV/SF	Description of Contents
NETID	X'01'	The network ID of the origin LU.
LU name	X'02'	The unqualified LU name of the origin.
MS application program name	X'03'	An MS application program name: • A 4-byte architecturally defined name • A 1- to 8-byte installation defined name.

Destination Location Name (X'82') MS Subvector

The destination location name (X'82') MS subvector consists of the structures in Table 15 on page 94.

Table 15. Contents of Destination Location Name (X'82') MS Subvector

Structure Name	GDS/SV/SF	Description of Contents
NETID	X'01'	The network ID of the destination LU.
LU name	X'02'	The unqualified LU name of the destination.
MS application program name	X'03'	An MS application program name: • A 4-byte architecturally defined name • A 1- to 8-byte installation defined name.

Flags (X'90') MS Subvector

This subvector contains the following indicators:

- **Message Type**

  The message type can be one of the following types:

  – MDS request
  – MDS reply
  – MDS error message

  These message types characterize the messages from the perspective of the NetView MS transport. The message type at the application program level can be different.

  Table 16 on page 95 summarizes the relationship between MDS message types and application program-level flows.
- **First MDS Message Indicator**

  This flag is set to 1 when the message is the first (or only) message for the unit of work.
- **Last MDS Message Indicator**

  This flag is set to 1 when the message is the last (or only) message for the unit of work. It is set to zero (0) when additional messages are expected.

Table 16. Settings of MDS Message Type First and Last MDS Message Flags

Application Program-Level Flow	MDS Message Type	First MDS Message Flag	Last MDS Message Flag
Request without reply Unsolicited data without acknowledgment (alert)	MDS request	1	1
Request with reply Unsolicited data with acknowledgment	MDS request	1	0
Reply (not last)	MDS reply	0	0
Last (or only) reply Acknowledgment	MDS reply	0	1
MDS error message	MDS error message	1	1

MDS-MUs contain a correlator field (described under “Agent Unit of Work Correlator (X'1549') GDS Variable” on page 95) that links requests and replies together. Error messages also contain a correlator that identifies failing MDS send requests. If replies are sent with the last indicator off, multiple MDS-MUs can be sent as replies to the same request. When this occurs, the NetView program buffers the replies until the last one is received and then sends all data to the application at the same time. If a timeout occurs before the last reply is received, all previous replies are discarded.

Agent Unit of Work Correlator (X'1549') GDS Variable

This correlator provides a value unique across time for the MS application program that assigns it, allowing MDS-MUs containing requests, replies, and error messages to be correctly correlated by both MDS and the application programs.

Correlator Contents

The contents of the correlator are described in Table 17 on page 96. The structures are broken down by subvector (SV) and subfield (SF).

Table 17. Contents of Agent Unit of Work Correlator (X'1549') GDS Variable		
Structure Name	**GDS/SV/SF**	**Description of Contents**
Requester location name	X'01'	Name of node where the unit of work originated
	X'01'	NETID of originating node
	X'02'	LU name of origination node
Requester agent	X'04'	Name of originating MS application
Sequence number date and time	X'02'	Uniquely identifies unit of work

Sequence Number Date and Time Structure

Table 18 on page 96 describes the contents of the sequence number date and time structure of the agent unit of work correlator (X'1549') GDS variable.

Table 18. Contents of Sequence Number Date and Time (SEQNO DTM) Structure			
Field Name	**Byte Offset**	**Description of Contents**	**Example Values**
Length	0	Length of SEQNO DTM structure	X'11'
Key	1	Key (always X'02')	X'02'
SEQNO	2-5	Unique binary sequence value	X'00000001'
Date		Date of unit of work origination:	X'07DC0B11'
	6, 7	4-digit year, in hexadecimal	
	8	2-digit month, in hexadecimal	
	9	2-digit day of month, in hexadecimal	
Time		Time of unit of work origination:	X'173B3B63'
	10	2-digit hour, in hexadecimal	
	11	2-digit minute, in hexadecimal	
	12	2-digit second, in hexadecimal	
	13	2-digit hundredth of second, in hexadecimal	
Time flag	14	Indication of whether date/time is local or Greenwich Mean Time (GMT): X'E9': Date/time is GMT (no offset). X'4E': Date/time is local (ahead of GMT). X'60': Date/time is local (trails GMT).	X'60'
GMT offset		Present only when date/time is local:	X'0400'
	15	2-digit hour offset in hexadecimal	
	16	l2-digit minute offset in hexadecimal	

The **Example Values** in Table 18 on page 96 describe a sequence number date and time structure of X'11020000000107DC0B11173B3B3B600400'. This hexadecimal string indicates the following information:

- Structure length is 17 (X'11').

- Structure key, which is always 2 (X'02').
- A unique sequence number of X'00000001'.
- Local date is 17 November 2012 (X'07DC0B11').
- Local time is 23:59:59.99. (X'173B3B63').
- Local time trails GMT (X'60').
- Offset from GMT is minus 4 hours (X'0400').

The GMT, also known as Coordinated Universal Time (UTC), is understood from the example data to be 18 November 03:59:59.99.

Accepting an MDS-MU

Hardware and software products in nodes that are not NetView nodes can send a CP-MSU within an MDS-MU to a hardware monitor MS application (ALERT_NETOP X'23F0F3F1') using the NetView MS transport. An MDS-MU has a key of X'1310' for its header.

MDS-MU Example

Figure 13 on page 97 shows an example of an MDS-MU message. The MDS message type is request without reply. It is being sent from a network known as NETA from an LU known as CNM01 and an application known as USERAPPL. It is going to an LU called CNM02 in the same network and to an application named ALERT_NETOP (X'23F0F3F1'). The data content is a CP-MSU containing an alert. See Figure 12 on page 93 for the format of an MDS header.

```
                00B41310 00371311 19810601 D5C5E3C1 0702C3D5 D4F0F10A
MDS Routing --► 03E4E2C5 D9C1D7D7 D3158206 01D5C5E3 C10702C3 D5D4F0F2
Information     060323F0 F3F10590 00C00000 33154916 010A01D5 C5E3C140     Agent Unit
                4040400A 02C3D5D4 F0F14040 400A04E4 E2C5D9C1 D7D7D30F ◄-- of Work
                02000000 03005B07 020A1413 00E90046 12120042 00000B92     Correlator
                00000121 01000000 01101000 0D110E0A 0040F1F2 F3F4F540
CPS-MSU ------► 40110303 0109D5C1 D4C5F140 4040E3E8 D7F10693 10011023
                0C960601 10221023 04813110
```

Figure 13. MDS-MU Message

MDS Data Types

The application GDS variable can be one of the following MDS data types:

- A CP-MSU
- An SNA condition report (SNACR)
- A routing report
- A network management vector transport (NMVT)
- A routing and targeting instruction (R&TI)

These data types are used by the MS transport and its NetView applications, including the hardware monitor. A major vector is a GDS variable. These data types are provided to allow operations management served applications to send architected operations management commands to remote systems for processing and receive commands from remote systems.

CP-MSU Format

Figure 14 on page 98 shows the format of a CP-MSU. For operations management served applications the CP-MSU (X'1212') consists of data that includes the R&TI information. The CP-MSU can also contain an SNA condition report (SNACR).

The maximum length allowed for the CP-MSU is 31743 (X'7BFF') bytes.

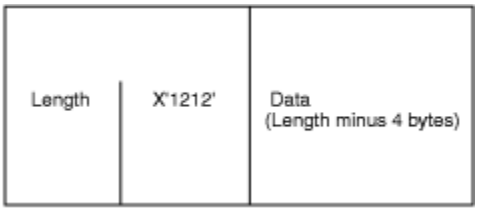


Figure 14. Format of a CP-MSU

Accepting a CP-MSU

Programs in the same node as the NetView program (regardless of address space) can use the PPI with function code 12 to send an NMVT or CP-MSU with a valid hardware monitor major vector to the hardware monitor.

The following major vectors are valid:

- X'0000' Alert
- X'0001' Link event
- X'0002' Resolve
- X'0025' PD statistics
- X'000F' ISDN/CMIP statistics
- X'132E' RECFMS envelope

CP-MSU has a key of X'1212'.

These major vectors are the expected major vectors. However, the ALERT_NETOP function does not limit the major vectors to those listed. If an unacceptable major vector is presented to the hardware monitor, it can later cause an error.

An exception is made for the first occurrence of the parameter major vector R&TI (X'154D'). This parameter major vector is not processed and does not cause an error. That R&TI is attached behind the next major vector, if one exists, when that next major vector is copied in a CP-MSU.

Multiple Major Vectors in a CP-MSU

The number of major vectors in a CP-MSU has no limit.

The multiple major vectors to be processed are separated using first-in, first-out (FIFO) queuing and processed individually. A CP-MSU with multiple major vectors is divided into that number of CP-MSUs, each with one of the major vectors that was in the input CP-MSU. The remaining parts of the hardware monitor process the new CP-MSUs.

If the initial CP-MSU arrives in an MDS-MU from the MS transport, each new CP-MSU has an image of the MDS header from that MDS-MU prefixed to it before being processed. Therefore, if the initial input was a CP-MSU, the NetView automation table, for an alert or resolve, examines a single major vector in a CP-MSU. The argument applied to automation is a single major vector in a CP-MSU within an MDS-MU. Prepare the user's automation table accordingly.

For operations management, a CP-MSU can have only one operations management architected major vector.

Routing Report Format

Routing reports are special CP-MSUs used by operations management to report routing errors at the application level. A special null subvector in the CP-MSU, X'00040077', identifies it as a routing report. The routing report contains an R&TI and an SNA condition report. The SNA condition report format is different from the format used in MDS error messages, which contain only an MDS header and SNA condition report, not an SNA condition report within a CP-MSU. The SNA condition report used in the

routing report, contains a structure report, as defined by SNA architecture. Figure 15 on page 99 shows the structure of the routing report.

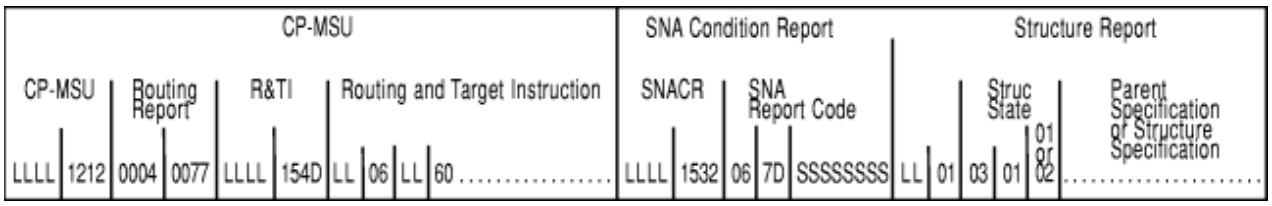


Figure 15. Format of a Routing Report

NMVT Format

Figure 16 on page 99 shows the format of a network management vector transport (NMVT). An NMVT contains a header and *B* bytes of data. The length of *B* plus 8 is a parameter outside the NMVT.

Refer to *Systems Network Architecture Formats* for more information about data type formats.

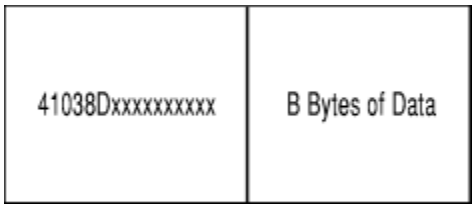


Figure 16. Format of an NMVT

R&TI Format

Figure 17 on page 99 shows the format of an R&TI. The R&TI format includes the following structures:

R&TI header
4 bytes; 2-byte length field of entire variable (X'154D')

Name list header
2 bytes; 1-byte length field (2-bytes and subfield lengths) (X'06')

Destination application
2-byte header plus destination application name (length + X'50')

Origin application
2-byte header plus destination application name (length + X'60')

Destination instance identifier
2-byte header plus destination instance used for task name (length + X'70')

Origin instance identifier
2-byte header plus origin instance used for task name (length + X'80')

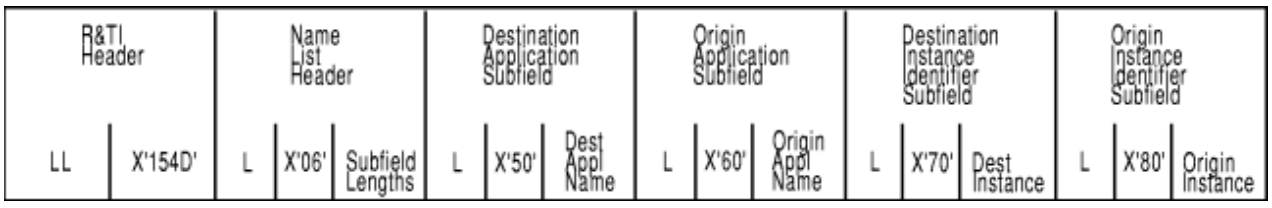


Figure 17. Format of an R&TI

MDS Error Message Format

The MDS error message is the vehicle for reporting errors detected by an MDS router at any point along the path of a transaction. It is also used by MS application programs under certain circumstances. Figure 18 on page 100 shows a high-level composition of an MDS error message.

Note: The MDS error message is not a unique GDS variable. It is an MDS-MU with a message type of MDS error message. It always contains the SNA condition report (X'1532') MDS variable as its application program GDS variable.

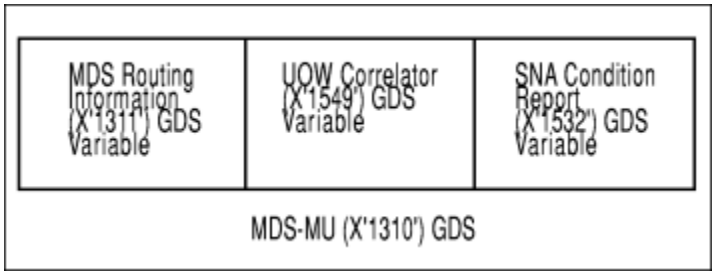


Figure 18. Format of an MDS Error Message

An MDS error message is created for two categories of errors:

- Routing errors in which an individual MDS-MU cannot be delivered successfully to its destination application program. However, no error report is created for nondelivery of an MDS error message.
- Transaction failure in which the error does not pertain to an individual MDS-MU, but to the sequence of such message units in a transaction. For example, it is a transaction failure if the NetView program is able to determine, from either the receipt of session outage notification or the expiration of the timer, that no reply is forthcoming for a request that expected one.

The MDS error message has the following characteristics:

- It is sent either from an MDS router in the node that detected the error or from one of the communicating MS application programs. The MDS router is a function of the MS architecture implemented by the NetView program.
- The agent unit of work correlator (X'1549') GDS variable in the MDS header identifies the MDS-MU that cannot be delivered or the transaction that failed.
- The application program data is an SNA condition report (X'1532') GDS variable, which carries an SNA report code (SNA sense data), identifying the precise error that was detected. An MDS error message created by MDS also identifies the LU and application program names for the other application program that was involved in the transaction, but not the LU and MS application programs that are the destination of the MDS error message. The LU and MS application programs are identified in the SNA condition report (X'1532') GDS variable because the origin of the MDS error message is the MDS router, not the partner application program.

Table 19 on page 101 shows the contents of an MDS error message created by an MDS router.

Table 19. Contents of an MDS Error Message (X'1310') Created by MDS Router

Structure Name	GDS/SV/SF	Description of Contents
MDS routing information	X'1311'	
• Origin location name	• X'81'	•
NETID	X'01'	NETID of reporting node
LU name	X'02'	LU name of reporting node
Application ID	X'03'	X'23F0F1F0' (MDS router)
• Destination location name	• X'82'	•
NETID	X'01'	NETID of destination node
LU name application ID	X'02'	LU name of destination program
Application ID	X'03'	Destination MS application program
• Flags	• X'90'	•
MDS message type		MDS error message (X'02')
First MDS message indicator		1-first message for unit of work
Last MDS message indicator		1-last message for unit of work
Agent unit of work correlator	X'1549'	Correlator of failed transaction
• Requester location name	• X'01'	• Name of node where work originated
Requester NETID	X'01'	NETID of node
LU name	X'02'	LU name of node
• Requester agent	• X'04'	• Name of originating MS application
• Sequence number date and time structure	• X'02'	• Unique work unit identifier
SNA condition report	X'1532'	
• SNA report code	• X'7D'	• Sense data indicating nature of error
• Reported on destination prefix	• X'08'	• (delimiter)
• Reported on location name	• X'09'	• Name of node
Reported on NETID	X'01'	NETID of node
Reported on node ID	X'02'	LU name of node
• Reported on destination suffix	• X'0B'	• (delimiter)
• Reported on agent	• X'04'	• MS application name

MDS Error Message Example

Figure 19 on page 101 shows the MDS router at CNM02 generating an error message to send to USERAPPL at CNM01 because ALERT_NETOP (X'23F0F3F1') was not registered at CNM02. See Figure 15 on page 99 for the format of an SNA condition report.

```
00911310 00371311 15810601 D5C5E3C1 0702C3D5 D4F0F206
0323F0F1 F0198206 01D5C5E3 C10702C3 D5D4F0F1 0A03E4E2
C5D9C1D7 D7D30590 02C00000 33154916 010A01D5 C5E3C140
4040400A 02C3D5D4 F0F14040 400A04E4 E2C5D9C1 D7D7D30F
02000000 03005B07 020A1413 00E90023 1532067D 08A80003
02080F09 0601D5C5 E3C10702 C3D5D4F0 F2020B06 0423F0F3
F1
```

Figure 19. MDS Error Message

Application Program-Level Error Reporting

The MDS error message is not the only method for reporting application program-detected errors. Other errors include:

- Command reject
- Parsing exception
- Function not supported

These errors are reported with application program-defined techniques such as reply major vectors, but under some circumstances, an MS application program must be able to end an outstanding MDS transaction unconditionally.

For example, an MS application program might start a timer when an MDS request is sent to another MS application program. If no reply has been received when the timeout period has elapsed, the sending MS application program can conclude that something is wrong with the destination application program, causing it not to respond. Because the sending MS application program will not wait for the reply any longer, it must end the outstanding MDS transaction with an MDS error message.

The format of an MDS error message sent by an MS application program is similar to that shown in Table 19 on page 101, except that the SNA condition report (X'1532') GDS variable in this MDS error message contains just the SNA report code containing the sense data. The reported-on location name and reported-on agent structures are not used because the partner application program is fully identified in the destination of the MDS error message.

Appendix B. Program-to-Program Interface Return Codes

This appendix describes the return codes generated by the program-to-program interface (PPI) and the hexadecimal equivalents for each request type. See “Building the Request Buffer” on page 13 for more information on the request parameter buffers.

Table 20. Return Codes Generated by Program-to-Program Interface Request Types

Ret Code	Hex Value	Description	Request Type										
			1	2	3	4	9	10	12	14	22	23	24
0	X'0'	The request completed successfully.			X	X	X	X	X	X	X	X	X
4	X'4'	The specified receiver is not active. The PPI has received a copy of the NMVT, CP-MSU, or data buffer.							X	X			
10	X'A'	The PPI is available to process user requests.	X										
14	X'E'	The receiver program is active.		X									
15	X'F'	The receiver program is inactive.		X			X	X					
16	X'10'	The receiver program is already active.				X							
18	X'12'	The receiver ECB is not zero.											X
20	X'14'	The request code is not defined.											
22	X'16'	The program issuing this request is not running in primary addressing mode.		X	X	X	X	X	X	X	X	X	X
23	X'17'	The user program is not authorized.							X	X			
24	X'18'	The PPI is not active.	X	X	X	X	X	X	X	X	X	X	X
25	X'19'	The ASCB address is not correct.					X	X			X	X	
26	X'1A'	The receiver program is not defined.		X			X	X	X	X	X	X	

Table 20. Return Codes Generated by Program-to-Program Interface Request Types (continued)

Ret Code	Hex Value	Description	Request Type										
			1	2	3	4	9	10	12	14	22	23	24
28	X'1C'	An active subsystem interface address space was found, but an active PPI address space was not found.	X	X	X	X	X	X	X	X	X	X	X
30	X'1E'	No data buffer in the receiver buffer queue.									X	X	
31	X'1F'	The receiver buffer is not large enough to receive the incoming data buffer.									X		
32	X'20'	No NetView storage is available.				X			X	X			
33	X'21'	The buffer length is not valid.							X	X	X		
35	X'23'	The receiver buffer queue is full.							X	X			
36	X'24'	ESTAE recovery cannot be established as requested.				X	X	X	X	X	X	X	
40	X'28'	SENDER-ID or RECEIVER-ID is not valid.		X		X	X	X	X	X	X		
90	X'5A'	A processing error has occurred.	X	X		X	X	X	X	X	X	X	X

Appendix C. Network Asset Management

Use this appendix to write NetView command lists. It contains general-use programming interface and associated guidance information.

This appendix provides information about the following items:

- Vital product data (VPD) returned from the VPDCMD command
- The sample network asset management command lists
- The record formats used by the sample network asset management command lists

Use the information provided in this appendix as a reference when interpreting messages returned from the VPDCMD command, when modifying the sample network asset management command lists, or when writing your own network asset management command lists.

Vital Product Data Descriptions

Messages that are returned in response to the VPDCMD command contain the following types of VPD:

- Answering node configuration data
- Product data
- DCE data, which, beginning with NetView for z/OS V5R4, is deprecated
- Link configuration data, which, beginning with NetView for z/OS V5R4, is deprecated
- Sense data
- Attached device configuration data
- Product set attributes

Different devices support different fields of the subvectors that return the VPD. A message is built using all the fields supported by an answering device. The message reflects all the VPD that is supplied by the device, including any supported fields that the device returns with values of blanks or zeros. Some devices can return values of blanks or zeros for fields that have no meaning to the device. You can obtain more information on some of the fields from the applicable hardware and software publications.

The layout of the subvectors formatted by network asset management are in this section. For more information about the subvectors and their major vectors, refer to *Systems Network Architecture Formats.*

Answering Node Configuration Data

Answering node configuration data describes the node that answered the VPD request. Answering node configuration data is returned in the following messages:

- DWO100I
- DWO103I

Messages DWO100I and DWO103I contain the same fields. Message DWO103I is issued instead of message DWO100I if the configuration reported by VTAM includes more names than the routine can manage. This situation can occur when the amount of storage allocated during initialization is not sufficient. You specify the amount of storage on the VPDSTOR operand of the VPDINIT statement. Because of the storage restriction, message DWO103I does not contain information about the complete configuration. The higher level node identification is not included.

Messages DWO100I and DWO103I are two-part messages. One part of the message contains information about the node where the VPD request originated. The other part of the message shows the configuration of the originating node.

The format of messages DWO100I and DWO103I is:

REQID *reqid* : ORIG *nodetype nodename* CNFG *nodetype nodename* [...]

Where:

reqid
Is the ID that correlates this reply to a specific request.

ORIG *nodetype nodename*
Describes the node where the VPD request originated.

nodetype can have one of the following values:

PU
The physical unit name

LSN
The link station name

LU
The logical unit name

LINK
The link name

CH/LINK
The channel-link name

nodename is the name of the originating node.

CNFG *nodetype nodename*
Describes the configuration of the originating node.

nodetype can have one of the following values:

PU
The physical unit name

LSN
The link station name

LU
The logical unit name

LINK
The link name

CH/LINK
The channel-link name

The *nodenames* are the names that make up the configuration of the originating node.

Product Data (Subvectors X'10' and X'11')

Message DWO102I contains data about the type of product of a particular node. The data is returned in subvectors X'10' and X'11'.

The format for message DWO102I is:

REQID *reqid* : *prodid vpdfield=xxx* [...]

Where:

reqid
Is the ID that correlates this reply to a specific request.

prodid
Identifies the type of product. The *prodid* variable can have one of the following values:

IBM-HW
IBM hardware

MIX-HW
IBM or non-IBM hardware (not distinguished)

OEM-HW
Non-IBM hardware

IBM-SW
IBM software

MIX-SW
IBM or non-IBM software (not distinguished)

OEM-SW
Non-IBM software

vpdfield=xxx
Represents a combination of various fields and values that describe the type of product of a particular node. The message reflects all the data returned from the device. Vital product data is device dependent. Each device builds a different combination of fields. The NetView program formats all the data that the device returns, even if the data contains blanks or zeros.

The following fields can be contained in DWO102I:

Field
Value

M/T
System type of the hardware product

MDL
System model number of the hardware product

MFG
Plant of manufacture of the hardware product

S/N
Sequence number of the hardware product

EM/T
System type of the emulated product

EMDL
System number of the emulated product

COMPID
Software serviceable component identifier

REL
Software serviceable component release level

VER
Software product common version identifier

RLS
Software product common release identifier

MOD
Software product common modification identifier

SPROD
Software common product name

NODEID
Software product customization identifier

PPN
Software program product number

CSD
Software product customization date in YY/DDD format

CST
Software product customization time in HH:MM format

ECL
Microcode EC level

HPROD
Hardware product common name

VID
Vendor identification

DCE Data

Beginning with NetView for z/OS V5R4, the DCE Data for Modems (Subvector X'50') and DCE Data for DSUs/CSUs (Subvector X'50') functions are deprecated.

Link Configuration Data (Subvector X'52')

Beginning with NetView for z/OS V5R4, this function is deprecated.

Sense Data (Subvector X'7D')

Message DWO111I contains SNA sense data supplied by a node that cannot satisfy a VPD request. The data is returned in subvector X'7D'.

The format of message DWO111I is:

REQID *reqid* : SNS *sensecode*

Where:

reqid
Is the ID that correlates this reply to a specific request.

sensecode
Is the 8-character hexadecimal sense code of the node that cannot satisfy a VPD request.

Attached Device Configuration Data (Subvector X'82')

Message DWO101I contains data about the configuration of a device attached to the node that reports the VPD. The data is returned in subvector X'82'.

The format of message DWO101I is:

REQID *reqid* : *vpdfield=xxx* [...]

Where:

reqid
Is the ID that correlates this reply to a specific request.

vpdfield=xxx
Represents a combination of fields and values that describe the attached device. The message reflects all the data returned from the device. Depending on the device, different fields are used to build this portion of the message. All the fields returned from the device are used, even the fields that contain blanks or zeros.

The following fields can be contained in DWO101I:

Field
Value

PORT
Is the port number where this device is attached.

PWROS
Is the power-on status of this device.

PWROL
Specifies whether this device has been powered on since the last time a solicitation for VPD was issued.

Product Set Attributes (Subvector X'84')

Message DWO105I transports additional attributes describing the product set. The data is returned in subvector X'84'.

The format of message DWO105I is:

REQID *reqid* : *vpdfield=xxx* [...]

Where:

reqid
Is the ID that correlates this reply to a specific request.

vpdfield=xxx
Represents a combination of fields and values that describe the attached device.

The following fields can be contained in DWO105I:

Field
Value

PHI
From 1–50 characters in character set 640 identifying the physical location of the product set.

LAN
Six bytes identifying the LAN universally assigned address. The 6 bytes are unique across all LAN adapters whose addresses are controlled by the Institute of Electrical and Electronics Engineers (IEEE).

Additional Product Set Attributes (Subvector X'86')

Message DWO106I reports the user-defined data. The message text is dynamically built from subvector X'86' that was embedded in a delivery RU from VTAM.

The format of message DWO106I is:

REQID *reqid* : *vpdfield=xxx* [...]

Where:

reqid
Is the ID that correlates this reply to a specific request.

vpdfield=xxx
Represents a combination of fields and values that describe the attached device.

The following fields can be contained in DWO106I:

Field
Value

LBL
From 1–25 characters in character set 640 identifying the text of a label for a product set attribute.

UDT
Characters 1–118 contain the data for a product set attribute.

Note: The X'10' subfield of subvector X'86' contains the additional product data with maximum length 224. The first 118 characters are written to the UDT field and the last 106 characters are truncated.

Network Asset Management Command Lists

You can use the sample command lists as they are shipped to perform basic vital product data (VPD) collection. If the sample command lists do not suit your particular network asset management requirements, you can modify the existing command lists, or you can write your own.

Using the Sample Command Lists

The sample command lists assign a record type number to each VPD record logged into an external file. The record type number is set in the common global variable SMFVPD by using the CNMSTUSR or C*xx*STGEN member. For information about changing CNMSTYLE statements, see *IBM Z NetView Installation: Getting Started*.

The sample command lists set the record type number to the default of 37 and continue processing. Record type 37 is used for hardware monitor log records, and in this case, for vital product data.

The following list shows the sample network asset management command lists:

VPDLOGC
Generates and logs start and end records when called by the VPDALL command.

VPDPU
Collects and logs VPD from a specified PU (and optionally its ports) in your domain.

VPDDCE
Collects and logs VPD from all of the DCEs existing in a direct path between a specified NCP and PU in your domain.

VPDXDOM
Enables a focal point to collect VPD from devices attached to another NetView program, when you use it in conjunction with the VPDLOGC, VPDPU, and VPDDCE command lists and a NetView automation definition. VPDXDOM sets the record type number of the NetView program that is soliciting for VPD to that of the focal point NetView program. This ensures that all the collected VPD is logged with the same record type number under the focal point NetView program.

Writing Command Lists

If you plan to write command lists to collect and log VPD:

1. Define the format of each type of log record you want to create.
2. Assign a record type number from 128–255 to each type of log record created.
3. Install an external logging facility if you want to log VPD into an external file. After logging data into an external file, you can manipulate the data using tools such as Service Level Reporter (SLR) or Information/Management for MVS.

Network Asset Management Record Formats

This section describes the format of the common prefix and the different subrecords. Each record written by a sample command list through the NetView external logging facility consist of a common prefix (header), followed by variable data. The variable data is contained in one of the following subrecords:

- Start (subtype S)
- End (subtype E)
- PU hardware (subtype P)
- PU software (subtype F)
- DCE hardware (subtype M)
- Time-out (subtype T)
- User data (subtype U)
- Error (subtype W)

Each subrecord contains a collect identifier field. This field correlates the records written during the same data collection. The last two digits of the collect identifier indicate the method used to collect the data.

Common Record Prefix

Each record created by the sample command lists is preceded by a common record prefix, or header, which contains identifying information about the record. Table 21 on page 111 is a format of this common prefix.

Table 21. Format of Common Record Prefix

Record Offset	Length in Bytes	Description
000	2	Record length
002	2	Reserved
004	1	Reserved
005	1	Record number
006	4	DATE (00yydddf)
010	4	TIME (sec/100)
014	4	System ID
018	4	Subsystem ID (set to VPD)
022	2	Subsystem record number (set to 22)

Start Subrecords

Start subrecords (subtype S) contain information about the start of data collection. Start subrecords are written by VPDLOGC, which is called by the VPDALL command at the start of data collection.

Table 22. Format of Start Subrecords

Record Offset	CMD Proc Offset	Length in Bytes	Description
024	001	1	Record subtype (S)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	8	Operator ID
053	030	8	Req (set to VPDALL)
061	038	3	Trailer (set to VPD)

End Subrecords

End subrecords (subtype E) contain information about the end of data collection. End subrecords are written by VPDLOGC, which is called by the VPDALL command at the end of data collection.

Table 23. Format of End Subrecords

Record Offset	CMD Proc Offset	Length in Bytes	Description
024	001	1	Record subtype (E)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)

Table 23. Format of End Subrecords (continued)			
Record Offset	**CMD Proc Offset**	**Length in Bytes**	**Description**
045	022	8	Operator ID
053	030	12	End of collect (mmddyyhhmmss)
065	042	8	Record counter [1]
073	050	8	Number of calls to VPDPU [2]
081	058	8	Reserved
089	066	8	Number of calls to VPDDCE [2]
097	074	8	Reserved
105	082	3	Trailer (set to VPD)

Notes:

1. This field contains the number of records, generated since the START subrecord, that can be written to an external logging facility. This field does not represent the number of records that were successfully written to the external logging facility.
2. This field contains the total number of calls made to the VPDPU or VPDDCE command list. It does not represent the number of successful completions of the command list.

PU Hardware Subrecords

PU hardware subrecords (subtype P) contain information about the hardware characteristics of a PU. PU hardware subrecords are written by the VPDPU command list.

Table 24. Format of PU Hardware Subrecords			
Record Offset	**CMD Proc Offset**	**Length in Bytes**	**Description**
024	001	1	Record subtype (P)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	5	System type
050	027	3	System model
053	030	3	Manufacturer ID
056	033	7	Sequence number
063	040	10	EC level
073	050	8	LU name
081	058	8	PU name
089	066	8	Link name
097	074	8	PU type 4 or 5 name
105	082	6	Attached port number
111	088	1	Current power on status
112	089	1	Power on status since last solicitation
113	090	12	Universal LAN adapter address

Table 24. Format of PU Hardware Subrecords (continued)			
Record Offset	**CMD Proc Offset**	**Length in Bytes**	**Description**
125	102	8	Reserved
133	110	3	Trailer (set to VPD)
136	113	50	Physical location [1]
Note: You can set this field.			

PU Software Subrecords

PU software subrecords (subtype F) contain information about the software characteristics of a PU. PU software subrecords are written by the VPDPU command list.

Table 25. Format of PU Software Subrecords			
Record Offset	**CMD Proc Offset**	**Length in Bytes**	**Description**
024	001	1	Record subtype (F)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	9	Component ID
054	031	3	Release level
057	034	6	Customization date
063	040	5	Customization time
068	045	5	Reserved
073	050	8	LU name
081	058	8	PU name
089	066	8	Link name
097	074	8	PU type 4 or 5 name
105	082	3	Trailer (set to VPD)

DCE Hardware Subrecords

DCE hardware subrecords (subtype M) contain information about the hardware characteristics of a DCE. DCE hardware subrecords are written by the VPDDCE command list.

Table 26. Format of DCE Hardware Subrecords			
Record Offset	**CMD Proc Offset**	**Length in Bytes**	**Description**
024	001	1	Record subtype (M)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	5	System type
050	027	3	System model
053	030	7	Serial number

Table 26. Format of DCE Hardware Subrecords (continued)

Record Offset	CMD Proc Offset	Length in Bytes	Description
060	037	1	Card EC level
061	038	1	Packaging EC level or micro code EC level
062	039	11	Reserved
073	050	8	Reserved
081	058	8	PU name
089	066	8	Link name
097	074	8	PU type 4 or 5 name
105	082	1	Total link segment level
106	083	1	Current link segment level
107	084	2	DCE address
109	086	1	DCE position [1]
110	087	3	Trailer (set to VPD)
113	090	20	Link station attribute [2]
133	110	Varied	DCE features [2]

Notes:

1. This field contains a character that identifies the sequence in which the DCEs are connected. The possible values are 1, 2, 3, and 4, with 1 being the DCE closest to the NCP that solicited the VPD.
2. These fields can be defined by the user.

Time-Out Subrecords

Timeout subrecords (subtype T) contain information about a VPD request that timed out before it was completed. Timeout subrecords are written by the VPDPU or VPDDCE command lists when the you specify the ERROR option.

Table 27. Format of Time-Out Subrecords

Record Offset	CMD Proc Offset	Length in Bytes	Description
024	001	1	Record subtype (T)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	8	Operator ID
053	030	8	Request specified
061	038	8	Node name 1
069	046	8	Node name 2
077	054	3	Trailer (set to VPD)

User Data Subrecords

User data subrecords (subtype U) contain additional product set data for a specified PU. The user data subrecords are written by the VPDPU command list.

Table 28. Format of User Data Subrecords

Record Offset	CMD Proc Offset	Length in Bytes	Description
024	001	1	Record subtype (U)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	25	User's label
070	047	118	User's data (see note)
188	165	3	Trailer (set to VPD)

Note: See message DWO106I for more information.

Error Subrecords

Error subrecords (subtype W) contain information about a VPD request that failed. Error subrecords are written by the VPDPU or VPDDCE command lists when you specify the ERROR option.

Table 29. Format of Error Subrecords

Record Offset	CMD Proc Offset	Length in Bytes	Description
024	001	1	Record subtype (W)
025	002	8	NetView program domain ID
033	010	12	Collect identifier (mmddyyhhmm99)
045	022	8	Operator ID
053	030	8	Request specified
061	038	8	Node name 1
069	046	8	Node name 2
077	054	8	Error message number
085	062	3	Trailer (set to VPD)

Appendix D. External Log Record Formats

Use this appendix to write NetView application programs. It contains general-use programming interface and associated reference information. It also provides the various log record formats written to external logs. These external logs can be SMF logs, or user-written logs.

External Log Record Type 37

The hardware monitor writes record type 37, subtype 4 records to the external log. Each record is made up of data sections preceded by an external log record header and a data descriptor section. The BNJTBRF macro maps the hardware monitor external log record.

See "Network Asset Management Record Formats" on page 110 describing VPD that can be written as external log record type 37 (subtype 22).

Note: In the following tables, a type of BinCD means binary coded decimal, which is a numeric value coded in binary format.

Table 30. External Log Record Header Format for the Hardware Monitor

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	BRFRLEN	Length of record including this header	BinCD
002	002	2	BRFRSEG	Segment descriptor	BinCD
004	004	1	BRFRFLG	System indicator	Hex
005	005	1	BRFRRTY	Record type [37 (X'25')]	BinCD
006	006	4	BRFRTME	Time since midnight, in hundredths of a second, record was moved into SMF buffer	Binary
010	00A	4	BRFRDTE	Date when the record was moved into the SMF buffer, in the form *00yydddF* or *0cyydddF* (where *c* is 0 for 19xx and 1 for 20xx, *yy* is the current year (0–99), *ddd* is the current day (1–366), and *F* is the sign)	Packed
014	00E	4	BRFRSID	System ID	Char
018	012	4	BRFRWID	Subsystem ID (set to 'NETV')	Char
022	016	2	BRFRSUBT	Record subtype (set to decimal 4)	BinCD

Table 31. Hardware Monitor Data Descriptor Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	4	BRFPRODI	Displacement from start of external log header to section	BinCD
004	004	2	BRFPROLN	Length of product section	BinCD
006	006	2	BRFPRONO	Number of product sections (0 or 1)	BinCD

Table 31. Hardware Monitor Data Descriptor Format (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
008	008	4	BRFALLDI	Displacement of common report from start of SMF header to section	BinCD
012	00C	2	BRFALLLN	Length of common section	BinCD
014	00E	2	BRFALLNO	Number of common sections (0 or 1)	BinCD
016	010	4	BRFEVTDI	Displacement of event report from start of SMF header to section	BinCD
020	014	2	BRFEVTLN	Length of event section	Num
022	016	2	BRFEVTNO	Number of event sections (0 or 1)	BinCD
024	018	4	BRFSTADI	Displacement of statistical report from start of SMF header to section	BinCD
028	01C	2	BRFSTALN	Length of statistical section	BinCD
030	01E	2	BRFSTANO	Number of statistical sections (0 or 1)	BinCD
032	020	4	BRFMODDI	Displacement of LPDA-1 modem report from start of SMF header to section **Note:** Beginning with NetView for z/OS V5R4, this function is deprecated.	BinCD
036	024	2	BRFMODLN	Length of LPDA-1 modem section **Note:** Beginning with NetView for z/OS V5R4, this function is deprecated.	BinCD
038	026	2	BRFMODNO	Number of LPDA-1 modem sections (0 or 1) **Note:** Beginning with NetView for z/OS V5R4, this function is deprecated.	BinCD
040	028	4	BRFLPDDI	Displacement of LPDA-2 report from start of SMF header **Note:** Beginning with NetView for z/OS V5R4, this function is deprecated.	BinCD
044	02C	2	BRFLPDLN	Length of LPDA-2 section **Note:** Beginning with NetView for z/OS V5R4, this function is deprecated.	BinCD
046	02E	2	BRFLPDNO	Number of LPDA-2 sections **Note:** Beginning with NetView for z/OS V5R4, this function is deprecated.	BinCD

Table 31. Hardware Monitor Data Descriptor Format (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
048	030	4	BRFLANDI	Displacement of local area network report from start of SMF record	BinCD
052	034	2	BRFLANLN	Length of local area network section	BinCD
054	036	2	BRFLANNO	Number of local area network sections	BinCD
056	038	4	BRFGENDI	Displacement of generic event report	BinCD
060	03C	2	BRFGENLN	Length of generic event section	BinCD
062	03E	2	BRFGENNO	Number of generic event sections (0 or 1)	BinCD
064	040	4	BRFETHDI	Displacement of ETHERNET LAN DATA report	BinCD
068	044	2	BRFETHLN	Length of ETHERNET LAN DATA section	BinCD
070	046	2	BRFETHNO	Number of ETHERNET LAN DATA sections	BinCD
072	048	4	BRFSELDI	Displacement of self-defined text message report	BinCD
076	04C	2	BRFSELLN	Length of self-defined text message section	BinCD
078	04E	2	BRFSELNO	Number of self-defined text message sections	BinCD
080	050	4	BRFDETTI	Displacement of detailed data network subfield report	BinCD
084	054	2	BRFDETLN	Length of detailed data network subfield section	BinCD
086	056	2	BRFDETNO	Number of detailed data network subfield sections	BinCD

Table 32. Product Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	BRFSUBTY	Record subtype (set to 4)	BinCD
002	002	2	BRFRELVL	NetView program release level (set to 32)	Char
004	004	4	BRFPRONM	Product name (set to 'NETV')	Char
008	008	8	BRFTIMST	Time stamp in form: 00YYDDDFHHMMSS0S	Packed

Table 33. Alert Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	8	BRFDOMNM	Domain name	Char
008	008	8	BRFFLRNM	Failing resource name	Char
016	010	4	BRFFLRTY	Failing resource type	Char
020	014	8	BRFHINM(1)	Resource level name 1	Char
028	01C	4	BRFHITY(1)	Resource level type 1	Char
032	020	8	BRFHINM(2)	Resource level name 2	Char
040	028	4	BRFHITY(2)	Resource level type 2	Char
044	02C	8	BRFHINM(3)	Resource level name 3	Char
052	034	4	BRFHITY(3)	Resource level type 3	Char
056	038	8	BRFHINM(4)	Resource level name 4	Char
064	040	4	BRFHITY(4)	Resource level type 4	Char
068	044	8	BRFHINM(5)	Resource level name 5	Char
076	04C	4	BRFHITY(5)	Resource level type 5	Char
080	050	1	BRFCPL	Complex link indicator (0=no 1=yes)	Hex
081	051	1	BRFALT	Alert indicator (0=no 1=yes)	Hex

Table 34. Generic Event Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	1	BRFETYPE	Event type	Char
001	001	9	BRFPROID	Product ID	Char
010	00A	4	BRFALTID	Alert ID number	Char
014	00E	40	BRFDESC	Event description	Char
054	036	40	BRFCAUS1	First probable cause	Char
094	05E	8	BRFCDPTS	Probable-cause code points 2 to 5	Hex
102	066	2	BRFFLAGS	Generic flags	Char
104	068	2	BRFEDCP1	Event description code point	Hex
106	06A	2	BRFPCCP1	First probable cause code point	Hex

Table 35. Event Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	1	BRFALRTT	Event type for event records	Hex
001	001	1	BRFGENCA	General cause for event records	Hex
002	002	1	BRFSPECA	Specific cause for event records	Hex

Table 35. Event Report Format (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
003	003	2	BRFBLKID	Block ID	Hex
005	005	1	BRFUACD	Action code	Hex
006	006	8	BRFUAQL1	Detail qualifier 1	Char
014	00E	8	BRFUAQL2	Detail qualifier 2	Char
022	016	8	BRFUAQL3	Detail qualifier 3	Char
030	01E	48	BRF48TXT	Error description: probable cause	Char
078	04E	2	BRFDBKID	Detail block ID	Hex
080	050	1	BRFDUACD	Detail action code	Hex
081	051	1	BRFNMJTY	NMVT type: **X'00'** NMVT 0000 **X'01'** NMVT 0001 **X'02'** NMVT 0025 **X'0F'** Miscellaneous NMVT **X'FF'** Non-NMVT	Hex

Table 36. Statistical Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	4	BRFTRFFC	Total traffic	BinCD
004	004	2	BRFTEMPS	Total temporary errors	BinCD

Table 37. Local Area Network Report

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	6	BRFLMADR	Local mac address	Hex
006	006	6	BRFRMADR	Remote mac address	Hex
012	00C	18	BRFROUTI	Routing information	Hex
030	01E	6	BRFUPADR	Mac address of upstream member	Hex
036	024	6	BRFDNADR	Mac address of downstream member	Hex
042	02A	6	BRFSMADR	Single mac address	Hex
048	030	16	BRFSMNAM	Single mac name	Char
064	040	2	BRFRIBID	Ring or BUS ID	Hex

Table 38. ETHERNET LAN Data Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	6	BRFLMADE	Local mac address	Hex
006	006	6	BRFRMADE	Not used	Hex
012	00C	18	BRFROUTE	Not used	Hex
030	01E	6	BRFUPADE	Not used	Hex
036	024	6	BRFDNADE	Not used	Hex
042	02A	6	BRFSMADE	Not used	Hex
048	030	16	BRFSMNAE	Not used	Char
064	040	1	BRFMCTPE	Mac type	Hex

Table 39. Self-defining Text Message Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	275	BRFTEXT	Self-defining text message	Char

Note: If the major vector contains multiple X'31' self-defining text subvectors, this field contains only the self-defining text from the first X'31' subvector.

Table 40. Detailed Data Network Alert Report Format

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	002	BRFDENUM	Number of detailed data network subfields	BinCD
002	002	253	BRFDEDAT	Detailed data network subfields	Char

Notes:

1. Only detailed data subfields not associated with qualified message data are supported.
2. The BRFDEDAT field can contain more than one subfield with each subfield preceded by a field containing the length of the subfield (see Table 41 on page 122 for the structure of the BRFDEDAT field).

Table 41. BFRDEDAT Mapping

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	BRFDATLN	Number of bytes of data contain in BRFDATTX	BinCD
002	002	*n*	BRFDATTX	Detailed data network subfield, where *n* is the value stored in BRFDATLN (see notes)	Char

Table 41. BFRDEDAT Mapping (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type

Notes:

1. This structure is repeated *z* times in BRFDEDAT, where *z* is the value stored in BRFDENUM (see Table 40 on page 122).
2. The length of the detailed data network subfields can vary; the length value is stored in the BRFDATLN field associated with each subfield.

External Log Record Type 38

This section describes the content and format of external log record type 38 (X'26'). Three subtypes are documented:

Subtype 1
Command authorization table

Subtype 2
Task resource utilization data

Subtype 3
Span authorization table

Subtype 4
Command statistics data

NetView Command Authorization Table External Log Record

The command authorization table external log record type 38, subtype 1 contains a common header, a specific data descriptor section, and other information sections that are defined by the data descriptor section:

For format of...	See...
Common header	Table 42 on page 124
Common product section	Table 43 on page 125
Data descriptor section	Table 44 on page 126
General section	Table 45 on page 126
Data section	Table 46 on page 127

Subtype 1 external log record type 38 is generated by auditing the NetView command authorization table, which is controlled globally by the NetView DEFAULTS CATAUDIT command or specifically by using the AUDIT keyword on the PERMIT and EXEMPT statements in the NetView command authorization table. Refer to the *IBM Z NetView Administration Reference* for a description of the PERMIT and EXEMPT statements and the NetView online help for the syntax of the DEFAULTS command.

NetView Task Resource-Utilization-Data External Log Record

The task resource-utilization-data external log record type 38, subtype 2 contains a common header, a specific data descriptor section, and other information sections defined by the data descriptor section:

For format of...	See...
Common header	Table 42 on page 124
Common product section	Table 43 on page 125

For format of...	See...
Data descriptor section	Table 47 on page 128
General section	Table 48 on page 128
Data section	Table 49 on page 130

NetView Span Authorization Table External Log Record

The span authorization table external log record type 38, subtype 3 contains a common header, a specific data descriptor section, and other information sections defined by the data descriptor section:

For format of...	See...
Common header	Table 42 on page 124
Common product section	Table 43 on page 125
Data descriptor section	Table 50 on page 134
General section	Table 51 on page 135
Access section	Table 52 on page 135
Resource/View name section	Table 53 on page 137
Operator section	Table 54 on page 137
Matching information section	Table 55 on page 137

NetView Command Statistics External Log Record

The Command Statistics External Log Record 38, subtype 4 contains common header, common product section, a specific data descriptor section, and other information sections that are defined by the data descriptor section:

For format of...	See...
Common header	Table 42 on page 124
Common product section	Table 43 on page 125
Data descriptor section	Table 56 on page 138
General section	Table 57 on page 138
Command data section	Table 58 on page 138

Subtype 4 external log record type 38 is generated at fixed intervals by the Command Statistics function for all the monitored NetView commands, which can be specified by the NetView CMDMON.INIT.LOGSMF statement in CNMSTYLE or from the CMDMON command.

Record Header and Section Formats

The following common record header and product information sections make up an external log record type 38.

Table 42. Format of Record Type 38 Header

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
000	000	2	S38LENG	Length of record	Binary

Table 42. Format of Record Type 38 Header (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
002	002	2	S38SEGD	SMF segment descriptor	Binary
004	004	1	S38SYSI	System indicator	Binary
005	005	1	S38RECT	Record type [38 (X'26')]	Binary
006	006	4	S38TIME	Time since midnight, in hundredths of a second, record was moved into SMF buffer	Binary
010	00A	4	S38DATE	Date when the record was moved into the SMF buffer, in the form *00yydddF* or *0cyydddF* (where *c* is 0 for 19xx and 1 for 20xx, yy is the current year (0-99), *ddd* is the current day (1-336), and *F* is the sign)	Packed
014	00E	4	S38SYID	System identification	Char
018	012	4	S38SUBS	Subsystem identification: "NETV"	Char
022	016	2	S38SUBT	Record subtype: **X'0001'** Command authorization table record **X'0002'** Task resource utilization data **X'0003'** Dynamic span table record **X'0004'** Command Statistics External Log Record	Binary
024	018	—	—	Start of variable length data	—

Table 43. Format of Common Product Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
000	000	2	S38CVER	Record version number	Binary
002	002	4	S38CPNM	Product name (NETV)	Char
006	006	2	S38CPVR	Product version and release, in the format *vr*, where *v* is version and *r* is release.	Char

Specific Subtype 1 Section Formats

The following tables contain specific command authorization table external log record (subtype 1) formats.

Table 44. Format of Subtype 1 Data Descriptor Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
000	000	2	S38TRNUM	Total number of triplets	Binary
002	002	2	—	Reserved	
004	004	4	S38PROFF	Offset to Product section	Binary
008	008	2	S38PRLEN	Length of Product section	Binary
010	00A	2	S38PRNUM	Number of Product sections	Binary
012	00C	4	S38GOFF	Offset to General section	Binary
016	010	2	S38GLEN	Length of General section	Binary
018	012	2	S38GNUM	Number of General sections	Binary
020	014	4	S38COFF	Offset to Command section	Binary
024	018	2	S38CLEN	Length of Command section	Binary
026	01A	2	S38CNUM	Number of Command sections	Binary
028	01C	4	S38KOFF	Offset to Keyword section	Binary
032	020	2	S38KLEN	Length of Keyword section	Binary
034	022	2	S38KNUM	Number of Keyword sections	Binary
036	024	4	S38VOFF	Offset to Value section	Binary
040	028	2	S38VLEN	Length of Value section	Binary
042	02A	2	S38VNUM	Number of Value sections	Binary
044	02C	4	S38IOFF	Offset to command identifier	Binary
048	030	2	S38ILEN	Length of command identifier	Binary
050	032	2	S38INUM	Number of command identifier sections	Binary
052	034	4	S38UOFF	Offset to User ID section	Binary
056	038	2	S38ULEN	Length of User ID section	Binary
058	03A	2	S38UNUM	Number of User ID sections	Binary
060	03C	4	S38CAOFF	Offset to Caller section	Binary
064	040	2	S38CALEN	Length of Caller section	Binary
066	042	2	S38CANUM	Number of Caller sections	Binary

Table 45. Format of Subtype 1 General Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
000	000	8	S38CTNM	Name of the NetView command authorization table	Char
008	008	8	S38CDOM	Domain where the NetView command authorization table is loaded	Char

Table 45. Format of Subtype 1 General Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
016	010	17	S38CTTM	Time that this NetView command authorization table was loaded, in the format:*MM/DD/YY HH:MM:SS*	Char
033	021	3	S38CHPA	If the authorization decision was PASS, this field describes how the PASS decision was authorized: **PER** by PERMIT statement **EXE** by EXEMPT statement	Char
036	024	4	S38CDEC	Authority decision, "PASS" or "FAIL"	Char
040	028	8	S38CMTY	Match type: **SPECIFIC** The command is explicitly protected by a command identifier that exactly matches the command being checked. **GENERIC** The command is protected by a command identifier that contains a generic character in one or more *command*, *keyword*, or *value* field.	Char

Table 46. Format of Subtype 1 Data Section

Located By	Name	Type	Description
S38COFF	S38CCOM	Char	The command that is being protected by a command identifier
S38KOFF	S38CKEY	Char	The keyword that is being protected by a command identifier
S38VOFF	S38CVAL	Char	The value that is being protected by a command identifier
S38IOFF	S38CCI	Char	The command identifier that caused this authorization check to fail or pass. The format is: *netid.luname.cmd.keyword.value*
S38UOFF	S38CUSER	Char	The user ID that is being checked
S38CAOFF	S38CCALR	Char	The user ID of the CALLER of the table authority check (see notes)

Notes:

1. This section is the variable length data that follows the S38CMTY field of the general section for the command authorization table (subtype 1) external log record type 38.
2. S38CCALR is present only when the CALLER differs from the user ID that is in S38CUSER. The CALLER can differ from S38CUSER when AUTHCHK=SOURCEID checking is in effect.

Specific Subtype 2 Section Formats

The following tables contain specific task resource-utilization-data external log record (subtype 2) formats.

Note: For additional help understanding this data, refer to the CNME1101 NetView REXX sample.

Table 47. Format of Subtype 2 Data Descriptor Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0000	0000	2	S38TRNUM2	Total number of triplets (X'0003')	Binary
0002	0002	2	—	Reserved	
0004	0004	4	S38PROFF2	Offset to product section	Binary
0008	0008	2	S38PRLEN2	Length of product section (X'0008')	Binary
0010	000A	2	S38PRNUM2	Number of product sections (X'0001')	Binary
0012	000C	4	S38GOFF2	Offset to general section	Binary
0016	0010	2	S38GLEN2	Length of general section (X'0034')	Binary
0018	0012	2	S38GNUM2	Number of general sections (X'0001')	Binary
0020	0014	4	S38DOFF2	Offset to data section	Binary
0024	0018	2	S38DLEN2	Length of data section (X'0060')	Binary
0026	001A	2	S38DNUM2	Number of product sections (X'0001')	Binary

Table 48. Format of Subtype 2 General Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0000	0000	2	S38TUrver	Record version number	Binary

Table 48. Format of Subtype 2 General Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0002	0002	2	S38TUevent	One of the following event codes: **1** LOGOFF or task ended abnormally **2** Session ended, task ended abnormally, and reinstated **3** Task stopped by STOP UNCOND **4** Task statistics at CLOSE NORMAL checkpoint **5** Task statistics at CLOSE STOP checkpoint **6** Task statistics at CLOSE IMMED checkpoint **7** Task statistics at CLOSE ABEND checkpoint **8** Task statistics at LOGTSTAT checkpoint **9** Task start events **10** Task statistics at interval	Binary
0004	0004	8	S38TUopid	Owning or billable operator ID The task name or operator ID (TVBOPID).	Char
0012	000C	8	S38TUluname	LU name The task name or terminal name connected to the task (TVBLUNAM).	Char
0020	0014	8	S38TUdomain	NetView domain name The NetView domain name in which the task ran.	Char

Table 48. Format of Subtype 2 General Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0028	001C	8	S38TUunique	NetView domain session correlation Each NetView program has a different value for this field, and each time the NetView program is started, this value is changed. Records with the same value came from the same NetView program and ran in the same address space.	Binary
0036	0024	8	S38TUsessid	NetView subtask session correlation Each task has a different value for this field, and each time the task starts, this value is changed. Records with the same value came from the same task and ran in the same "session." Recovery of an abnormal end is treated as the same session.	Binary
0044	002C	8	S38TUstck	Current STCK value for data The internal hardware clock at the time the data was recorded. (The store clock was shifted 12 bits to the right.)	Binary

Table 49. Format of Subtype 2 Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0000	0000	2	S38TUdataVer	Record version number	Binary
0002	0002	2	S38TUmaxCPU	Maximum processor percentage (hundredths of percent) The maximum measured processor during a 10 second interval since the task began, or since the last LOGTSTAT RESETMAX or LOGTSTAT INTERVAL command.	Binary
0004	0004	4	S38TUsessSec	Task session time in seconds	Binary

Table 49. Format of Subtype 2 Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0008	0008	4	S38TUsessFrac	Task time fraction of one second in microseconds The elapsed time the task has run in seconds plus microseconds. The microseconds field provides accurate timing for short intervals. **Note:** These two fields are not a double-word pair containing microseconds. Refer to sample CNME1101 for algorithms in REXX for timing arithmetic.	Binary
0012	000C	4	S38TUcpuSec	Processor units used in seconds	Binary
0016	0010	4	S38TUcpufrac	Processor time fraction of one second in microseconds The amount of processor time charged to this task by MVS. The microseconds field provides accurate timing for short intervals. **Note:** These two fields are not a double-word pair containing microseconds. See sample CNME1101 for algorithms in REXX for timing arithmetic.	Binary
0020	0014	4	S38TUpenSec	Penalty time in seconds	Binary
0024	0018	4	S38TUpenFrac	Penalty time fraction of one second in microseconds The number of seconds that this task has waited because of MAXMQIN, AVLSLOW, SLOWSTG, MAXCPU, MAXMQOUT, or MAXIO penalties. The microseconds field provides accurate timing for short intervals. **Note:** These two fields are not a double-word pair containing microseconds. See sample CNME1101 for algorithms in REXX for timing arithmetic.	Binary
0028	001C	2	S38TUavgCPU	Average processor utilization in hundredths of percent The percentage of a processor this task has used. The ratio of Used CPU to Session Seconds.	Binary

Table 49. Format of Subtype 2 Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0030	001E	2	S38TUpnPct	Average penalty imposed in hundredths of percent The percentage of elapsed time that this task has waited for penalties. The ratio of Penalty Seconds to Session Seconds.	Binary
0032	0020	4	S38TUmaxStg	Maximum storage used The largest usage of storage for this task since the task was started or since the last LOGTSTAT RESETMAX or LOGTSTAT INTERVAL command.	Binary
0036	0024	4	S38TUgetRate	Total DSIGET usage rate in KB per minute The average rate (for the life of the task) at which storage was obtained by DSIGET, in KB per minute.	Binary
0040	0028	4	S38TUfreRate	Total DSIFRE usage rate in KB per minute The average rate (for the life of the task) at which storage was released by DSIFRE, in KB per minute.	Binary
0044	002C	4	S38TU24gRate	A 24-bit DSIGET usage rate in KB per minute The average rate (for the life of the task) at which storage was obtained by DSIGET, in KB per minute (24-bit storage only).	Binary
0048	0030	4	S38TU24fRate	A 24-bit DSIFRE usage rate in KB per minute The average rate (for the life of the task) at which storage was released by DSIFRE, in KB per minute (24-bit storage only).	Binary
0052	0034	4	S38TUmxmiRate	Maximum inbound DSIMQS rate in KB per minute The maximum rate, over a one-minute period, at which messages were queued to this task by DSIMQS, in KB per minute. This is the maximum rate since the task started, or since the last LOGTSTAT RESETMAX or LOGTSTAT INTERVAL command.	Binary

Table 49. Format of Subtype 2 Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0056	0038	4	S38TUmqiRate	Average inbound DSIMQS rate in KB per minute The rate, over the life of the task, at which messages were queued to this task by DSIMQS, in KB per minute.	Binary
0060	003C	4	S38TUmxmoRate	Maximum outbound DSIMQS rate The maximum rate (for a 1 minute period) at which messages were sent by this task by DSIMQS, in KB per minute. This is the maximum rate since the task started, or since the last LOGTSTAT RESETMAX or LOGTSTAT INTERVAL command.	Binary
0064	0040	4	S38TUmqoRate	Average outbound DSIMQS rate The rate, over the life of the task, at which messages were sent by this task by DSIMQS, in KB per minute.	Binary
0068	0044	4	S38TUmqiTot	Number of inbound DSIMQS messages The rate, over the life of the task, at which messages were sent to this task by DSIMQS, in KB per minute.	Binary
0072	0048	4	S38TUmqoTot	Number of outbound DSIMQS messages The number of messages sent by this task over the life of the session.	Binary
0076	004C	4	S38TUioTot	Number of DASD I/Os The total number of I/Os done by NetView services on this task for the life of this task.	Binary
0080	0050	4	S38TUmxiorate	Maximum I/O rate (I/Os per minute) The maximum rate of I/Os per minute in a 1 minute interval since the task was started or since the last LOGTSTAT RESETMAX or LOGTSTAT INTERVAL command.	Binary
0084	0054	4	S38TUioRate	Average I/O rate (I/Os per minute) The average rate of I/Os per minute for the life of the task.	Binary
0088	0058	4	S38TUmqiPNs	Penalties caused by task (bytes per second)	Binary

Table 49. Format of Subtype 2 Data Section (continued)					
Offset Dec.	**Offset Hex.**	**Length in Bytes**	**Name**	**Description**	**Type**
0092	005C	4	S38TUmqiPNm	Penalties caused by the task, fraction of 1 second (bytes per microsecond) The total number of penalty seconds that this task caused other tasks to wait because of MAXMQIN, SLOWSTG, or AVLSLOW limit being exceeded for this task. A penalty time is served when a DSIMQS from another task is sent to the task that is over any of these limits. The microseconds field provides accurate timing for short intervals. **Note:** These two fields are not a double-word pair containing microseconds. See sample CNME1101 for algorithms in REXX for timing arithmetic.	Binary

Specific Subtype 3 Section Formats

The following tables contain specific span authorization table external log record (subtype 3) formats.

Table 50. Format of Subtype 3 Data Descriptor Section				
Offset Dec.	**Offset Hex.**	**Length in Bytes**	**Description**	**Type**
0000	0000	2	Total number of triplets (X'0006')	Binary
0002	0002	2	Reserved	
0004	0004	4	Offset to product section	Binary
0008	0008	2	Length of product section (X'0008')	Binary
0010	000A	2	Number of product sections (X'0001')	Binary
0012	000C	4	Offset to general section	Binary
0016	0010	2	Length of general section (X'0022')	Binary
0018	0012	2	Number of general sections (X'0001')	Binary
0020	0014	4	Offset to access section	Binary
0024	0018	2	Length of access section (X'0014')	Binary
0026	001A	2	Number of access sections (X'0001')	Binary
0028	001C	4	Offset to resource/view name section	Binary
0032	0020	2	Length of resource/view name section	Binary
0034	0022	2	Number of resource/view name sections (X'0001')	Binary
0036	0024	4	Offset to operator section	Binary
0040	0028	2	Length of operator section	Binary

Table 50. Format of Subtype 3 Data Descriptor Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0042	002A	2	Number of operator sections (X'0001')	Binary
0044	002C	4	Offset to matching information section	Binary
0048	0030	2	Length of matching information section	Binary
0050	0032	2	Number of matching information sections (X'0001')	Binary

Table 51. Format of Subtype 3 General Section

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0000	0000	8	Span table name	Char
0008	0008	8	NetView domain name where the span table is loaded	Char
0016	0010	17	Date/time when this span table is loaded in *MM/DD/YY HH:MM:SS* format	Char
0033	0021	1	Reserved	

Table 52. Format of Subtype 3 Access Section

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0000	0000	4	Access decision PASS or FAIL	Char
0004	0004	4	Request origin: **CMD** Request from commands **VIEW** Request from NetView management console views	Char
0008	0008	4	Type of name: **RESC** Resource name **VIEW** NetView management console view name	Char

Table 52. Format of Subtype 3 Access Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0012	000C	8	Reason for PASS or FAIL: **NOACTIVE** Operator has no active span **NO MATCH** No match found **RESCSTOP** Resource has been stopped **GEN NAME** For CTL=GENERAL, name defined generically to another span **DEF NAME** For CTL=GENERAL, name defined specifically to another span **RDMQFAIL** RODM query failed **RDMNONAM** No name defined to RODM object **INVALLEN** For a VIEW request, the resource or view name has a zero length **GLOB MAT** Operator has CTL=GLOBAL **GLOBVTAM** Operator has CTL=GLOBAL and the command entered is a VTAM command **SPEC MAT** Name matches a specifically defined name **GENR MAT** Name matches a generically defined name **GLSA MAT** Name matches a leading single asterisk name **GLDA MAT** Name matches a leading double asterisk name **NMNOTDEF** For CTL=GENERAL (PASS), name not defined to the NetView program **NOSPNDEF** For CTL=SPECIFIC (FAIL) and CTL=GENERAL (PASS), no span definition in use **DBCSNAME** No specific match found for a DBCS name; no generic match done for a DBCS name	

Table 53. Format of Subtype 3 Resource/View Name Section

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0000	0000	2	Length of the resource or view name that follows; can be 0	Binary
0002	0002		If the length is nonzero, the resource or view name. If the operator has CTL=GLOBAL and reason for matching is GLOBVTAM, this field can have a group of resource names, separated by commas.	Char

Table 54. Format of Subtype 3 Operator Section

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0000	0000	8	Operator ID, padded with blanks; an operator ID of *BYPASS* indicates the VTAM command is issued from a system console where no logon has been performed.	Char
0008	0008	4	Operator CTL setting: **GLOB** Global **SPEC** Specific **GENL** General	Char
0012	000C	2	Number of active spans that the operator has; can be 0	Binary
0014	000E		If the number of active spans is nonzero, the operator's active spans; each span is 8-bytes long padded with blanks	Binary

Table 55. Format of Subtype 3 Matching Section

Offset Dec.	Offset Hex.	Length in Bytes	Description	Type
0000	0000	4	Reserved for NetView internal use	Char
0004	0004	8	Span name in which the match is found; *GLOBAL* means the operator has CTL=GLOBAL; blanks indicate no match was found	Char
0012	000C	2	Length of the resource or view name that matches; 0 indicates failure for CTL=SPECIFIC	Binary
0014	000E		If the length is nonzero, the resource or view name that was matched	Char

Specific Subtype 4 Section Formats

The following tables contain command statistics external log record (subtype 4) formats.

Table 56. Format of Subtype 4 Data Descriptor Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0000	0000	2	S38TRNUM4	Total number of triplets	Binary
0002	0002	2	—	Reserved	
0004	0004	4	S38PROFF4	Offset to product section	Binary
0008	0008	2	S38PRLEN4	Length of product section (X'0008')	Binary
0010	000A	2	S38PRNUM4	Number of product sections (X'0001')	Binary
0012	000C	4	S38GOFF4	Offset to general section	Binary
0016	0010	2	S38GLEN4	Length of general section (X'000C')	Binary
0018	0012	2	S38GNUM4	Number of general sections (X'0001')	Binary
0020	0014	4	S38DOFF4	Offset to data section	Binary
0024	0018	2	S38DLEN4	Length of data section (X'0048')	Binary
0026	001A	2	S38DNUM4	Number of data sections	Binary

Table 57. Format of Subtype 4 General Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0000	0000	1	S38GMODE	Status of Command Statistics Function. The status may be one of the following codes: **1** Command statistics are collected for all monitored commands. **2** Command statistics are collected only for primary commands.	Binary
0001	0001	3	—	Reserved	Binary
0004	0004	8	S38GDOM	NetView Domain where the NetView command statistics data is collected	Char

Table 58. Format of Subtype 4 Command Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Name	Description	Type
0000	0000	8	S38DCMDN	Command Name	Char
0008	0008	8	S38DALTN	Command Alternate Name	Char
0016	0010	8	S38DPRNT	Parent Name	Char
0024	0018	8	S38DTSK	NetView Task Name	Char
0032	0020	8	S38DSTCK	Start Date and Time	STCK
0040	0028	8	S38DETCK	End Date and Time	STCK

Table 58. Format of Subtype 4 Command Data Section (continued)					
Offset Dec.	**Offset Hex.**	**Length in Bytes**	**Name**	**Description**	**Type**
0048	0030	8	S38DCPU	CPU Time in Microseconds	Binary
0056	0038	4	S38DSTG	Storage High Water Mark in Bytes	Binary
0060	003C	4	S38DIOC	Total I/O Count	Binary
0064	0040	8	S38DAUSR	Authorized User Name	Char

External Log Record Type 39

The session monitor writes log record type 39 to an external log under the following conditions:

- When an operator or command list issues a RECORD command with the STRGDATA parameter, the NetView program writes a counter record type 39 (X'27') to the external log.
- When the network accounting and availability measurement function is active, it writes an external log record type 39 (X'27') when a session is started, a session is ended, or a RECORD command with the SESSTATS parameter is issued.
- When the response time data function is active, it writes an external log record type 39 (X'27') when you issue the COLLECT command with the LOG parameter, or at session end for an LU attached to a PU with the RTM feature.

Each record is made up of data sections preceded by an external log record header and a data descriptor section.

The AAUTLOGR macro maps the session monitor external log record.

Record Subtypes

The session monitor writes one of the following type 39 (X'27') records to the external log:

- RTM collection record
- Session end record
- Session start record
- Accounting and availability data collection record
- Combined session start-end record
- BIND failure record
- INIT failure record
- Storage and event counters

Each record is divided into sections. The functions described in the following sections are shown in more detail in “Record Section Formats” on page 141.

RTM Collection Record (Subtype X'0001')

With the response time measurement function active, the NetView program writes an RTM collection record to the external log. This happens whenever an operator or a command list issues a COLLECT command with the LOG parameter. The RTM collection record has 5 data sections:

- Product ID section
- Session configuration data
- Session route data
- Session response time data

- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

Session End Record (Subtype X'0002')

When the session monitor network accounting and availability measurement function is active, the NetView program writes a session end record to the external log when a session ends. The session end record has 6 data sections:

- Product ID section
- Session configuration data
- Session route data
- Session response time data
- Accounting and availability data
- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

The session response time data section is provided only if response time monitoring is active.

Session Start Record (Subtype X'0003')

When the session monitor network accounting and availability measurement function is active, the NetView program writes a session start record to the external log when a new session starts. The session start record has 5 data sections:

- Product ID section
- Session configuration data
- Session route data
- Accounting and availability data
- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

Accounting and Availability Data Collection Record (Subtype X'0004')

When the session monitor network accounting and availability measurement function is active, the NetView program writes an accounting and availability data collection record to the external log. This occurs whenever an operator or a command list issues the RECORD command with the SESSTATS parameter. The accounting and availability data collection record has 4 data sections:

- Product ID section
- Session configuration data
- Accounting and availability data
- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

Combined Session Start-End Record (Subtype X'0005')

When the session monitor network accounting and availability measurement function or the response time monitor function is active, the NetView program writes a combined session start-end record. This record is written when a session ends before the NetView program has a chance to write the session start record for that session, or when the response time monitor function is active and the network accounting and availability function is inactive. The combined session start-end record has 6 data sections:

- Product ID section
- Session configuration data
- Session route data
- Session response time data
- Accounting and availability data
- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

The session response time data section is provided only if response time monitoring is active.

BIND Failure Record (Subtype X'0006')

When the session monitor network accounting and availability measurement function is active, the NetView program writes a BIND failure record to the external log whenever a session setup fails during the BIND flow. The BIND failure record has 4 data sections:

- Product ID section
- Session configuration data
- Session route data
- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

INIT Failure Record (Subtype X'0007')

When the session monitor network accounting and availability measurement function is active, the NetView program writes an INIT failure record to the external log whenever a session setup fails during the INIT flow. The INIT failure record has 4 data sections:

- Product ID section
- Session configuration data
- Session route data
- Advanced Peer-to-Peer Networking route data, if this is an Advanced Peer-to-Peer Networking session

Storage and Event Counter Record (Subtype X'0008')

The NetView program writes a counter record to the external log whenever an operator or a command list issues a RECORD command with the STRGDATA parameter. The counter record has 5 data sections:

- Product ID section
- Event counter data
- Session awareness counter data
- Resource counter data
- Storage data

Record Section Formats

The NetView program builds the external log records by adding combinations of the following data sections to the external log record header section and the data descriptor section:

- Product ID section
- Session route data section
- Session configuration data section
- Response time data section
- Accounting and availability data section
- Event counter data section
- Session awareness counter data section
- Resource counter data section
- Storage data section
- Advanced Peer-to-Peer Networking route data section

The detailed format of each section follows:

Header and Data Descriptor Data Sections

This topic describes the formats of the session monitor external log record header section and data descriptor sections. Each external log record has a header section and a data descriptor section. The following two data descriptor section formats are used:

- Data for Subtypes X'0001' through X'0007' (see Table 60 on page 142)
- Storage and Event Counter Data (see Table 61 on page 143)

Table 59. Format of the External Log Record Header Format for the Session Monitor

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LOGRLENG	Record length	Binary
002	002	2	LOGRSEGD	Segment descriptor	Binary
004	004	1	LOGRSYSI	System indicator: (X'04' for MVS)	Binary
005	005	1	LOGRRECT	Record type (X'27')	Binary
006	006	4	LOGRTIME	Time since midnight, in hundredths of a second, record was moved into SMF buffer	Binary
010	00A	4	LOGRDATE	Date when the record was moved into the SMF buffer, in the form *01yydddF* where yy is the current year (0–99), *ddd* is the current day (1–336), and *F* is the sign)	Packed
014	00E	4	LOGRSYID	System identification (taken from SID parameter)	Char
018	012	4	LOGRSUBS	Subsystem ID 'NETV'	Char
022	016	2	LOGRSUBT	Record subtype [1]	Binary

Note: For LOGRSUBT field values, see “Record Subtypes” on page 139. For more information about the fields in the external log record header, refer to the MVS library.

Table 60. Format of the Data Descriptor Section for Subtypes X'0001' Through X'0007'

Offset Dec.X''	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	4	LHDRPRDO	Offset of product section [1]	Binary
004	004	2	LHDRPRDL	Length of product section	Binary
006	006	2	LHDRPRDN	Number of product sections [2]	Binary
008	008	4	LHDRSESO	Offset of session configuration section [1]	Binary
012	00C	2	LHDRSESL	Length of session configuration section	Binary
014	00E	2	LHDRSESN	Number of session configuration sections [2]	Binary
016	010	4	LHDRRTEO	Offset of route data section [1]	Binary

Table 60. Format of the Data Descriptor Section for Subtypes X'0001' Through X'0007' (continued)

Offset Dec.X''	Offset Hex.	Length in Bytes	Field Name	Description	Type
020	014	2	LHDRRTEL	Length of route data section	Binary
022	016	2	LHDRRTEN	Number of route data sections [2]	Binary
024	018	4	LHDRRTMO	Offset of response time data section [1]	Binary
028	01C	2	LHDRRTML	Length of response time data section	Binary
030	01E	2	LHDRRTMN	Number of response time data sections [2]	Binary
032	020	4	LHDRACCO	Offset of accounting and availability data section [1]	Binary
036	024	2	LHDRACCL	Length of accounting and availability data section	Binary
038	026	2	LHDRACCN	Number of accounting and availability data sections [2]	Binary
040	028	4	LHDRARTO	Offset of Advanced Peer-to-Peer Networking route data section	Binary
044	02C	2	LHDRARTL	Length of Advanced Peer-to-Peer Networking route data section	Binary
046	02E	2	LHDRARTN	Number of Advanced Peer-to-Peer Networking route data sections	Binary

Notes:

1. The offset of the first section of this type. All offsets are relative to the beginning of the record.
2. The number of sections of this type in the record.

Table 61. Format of the Data Descriptor Section for Storage and Event Counter Data

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	4	LCNTPRDO	Offset of product section [1]	Binary
004	004	2	LCNTPRDL	Length of product section	Binary
006	006	2	LCNTPRDN	Number of product sections [2]	Binary
008	008	4	LCNTEVNO	Offset of event counters [1]	Binary
012	00C	2	LCNTEVNL	Length of event counter data section	Binary
014	00E	2	LCNTEVNN	Number of event counter data sections [2]	Binary
016	010	4	LCNTSAWO	Offset of SAW counters [1]	Binary
020	014	2	LCNTSAWL	Length of SAW data section	Binary
022	016	2	LCNTSAWN	Number of SAW counter data sections [2]	Binary
024	018	4	LCNTARBO	Offset of ARB counters [1]	Binary
028	01C	2	LCNTARBL	Length of ARB data section	Binary

Table 61. Format of the Data Descriptor Section for Storage and Event Counter Data (continued)					
Offset Dec.	**Offset Hex.**	**Length in Bytes**	**Field Name**	**Description**	**Type**
030	01E	2	LCNTARBN	Number of ARB data sections [2]	Binary
032	020	4	LCNTSTGO	Offset of storage counters [1]	Binary
036	024	2	LCNTSTGL	Length of STRG data section	Binary
038	026	2	LCNTSTGN	Number of STRG data sections [2]	Binary

Notes:

1. The offset of the first section of this type. All offsets are relative to the beginning of the record.
2. The number of sections of this type in the record

Product Data Section

The product data section is used with all session monitor external log record subtypes.

Table 62. Format of the Product Data Section					
Offset Dec.	**Offset Hex.**	**Length in Bytes**	**Field Name**	**Description**	**Type**
000	000	2	LPRDSUBT	Record subtype [1]	Binary
002	002	2	LPRDVERN	NetView program release level	Char
004	004	4	LPRDNAME	Product name (set to 'NETV')	Char

Note: The LPRDSUBT field is the same as the LOGRSUBT field in the log record header section. For valid LPRDSUBT field values, see “Record Subtypes” on page 139.

Session Configuration Data Section

The session configuration data section is used with all session monitor external log record subtypes, except subtype 8 (storage and event counter record). Most of the information for this record comes from VTAM.

Table 63. Format of the Session Configuration Data Section					
Offset Dec.	**Offset Hex.**	**Length in Bytes**	**Field Name**	**Description**	**Type**
000	000	2	LSESREVL	Revision level: X'0003'	Binary
002	002	8	LSESPNAM	Primary resource name[3]	Char
010	00A	8	LSESPPUN	PU of primary resource	Char
018	012	8	LSESPLNK	Primary link name [1]	Char
026	01A	8	LSESPSAP	Primary subarea PU	Char
034	022	8	LSESPDOM	Primary NetView domain name	Char
042	02A	8	LSESSNAM	Secondary resource name [3]	Char
050	032	8	LSESSPUN	PU of secondary resource	Char
058	03A	8	LSESSLNK	Secondary link name [1]	Char

Table 63. Format of the Session Configuration Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
066	042	8	LSESSSAP	Secondary higher level PU name	Char
074	04A	8	LSESSDOM	Secondary NetView domain name	Char
082	052	8	LSESPCLS	Performance class name	Char
090	05A	8	LSESCOST	Class of service name	Char
098	062	2	LSESERN	Explicit route number [2]	Binary
100	064	2	LSESRERN	Reverse explicit route number [2]	Binary
102	066	2	LSESVRN	Virtual route number [2]	Binary
104	068	2	LSESTPF	Transmission priority [2]	Binary
106	06A	8	LSESPCID	Unique session ID	Binary
114	072	1	LSESTYPE	Session type: **1** LU-LU **2** SSCP-LU **3** SSCP-PU **4** SSCP-SSCP **5** CP-CP	Char
115	073	1	LSESXNET	Cross-network session (Y or N)	Char
116	074	1	LSESCODE	BIND failure or UNBIND reason code; see the following offsets: • Offset 7 into BINDF • Offset 15 into CDSESSF • Offset 1 into UNBIND. For more information, see *Systems Network Architecture Formats*.	Binary
117	075	8	LSESPRNT	Primary endpoint CP network ID	Char
125	07D	8	LSESPRNM	Primary endpoint CP name	Char
133	085	8	LSESSCNT	Secondary endpoint CP network ID	Char
141	08D	8	LSESSCNM	Secondary endpoint CP name	Char
149	095	8	LSESCOSA	Advanced Peer-to-Peer Networking class of service name	Char
157	09D	2	LSESTPFA	Advanced Peer-to-Peer Networking transport priority	Binary
159	09F	1	LSESFQLN	Length of fully qualified PCID CP name	Binary

Table 63. Format of the Session Configuration Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
160	0A0	17	LSESFQNM	Fully qualified PCID CP name	Char
177	0B1	4	LSESSCOD	Sense code	Binary
181	0B5	8	LSESLOGM	Logmode name	Char-

Notes:

1. This field contains the link name, channel unit address name, or dependent LU requestor name.
2. If no data is available from VTAM, the default is X'FF'.
3. For configurations with hierarchies having more than 4 resources on either primary or secondary sides, (a "virtual/logical" PU/LINK coded), this field can contain the name of a link, channel unit address, or line group.

Session Route Data Section

The session route data section is used with all session monitor external log record subtypes, except subtype 4 (accounting and availability data collection record) and subtype 8 (storage and event counter record).

Table 64. Format of the Session Route Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LRTEREVL	Revision level: X'0001'	Binary
002	002	2	LRTENUME	Total number of nodes in session path	Binary
004	004	2	LRTENUMT	Number of route elements in LRTEETAB. This is the number of NCP V3 or later nodes in the session path.	Binary
006	006	—	LRTEETAB	Route element table (see note)	—

Note: A 10-byte entry exists for each route element in the table. See Table 65 on page 146 for the format of this 10-byte entry.

Table 65. Format of the Ten-Byte Route Element Entry

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	8	LRTEENAM	Route element name	Binary
008	008	2	LRTEETGO	Transmission group (out) number	Binary

Note: Fields LRTEENAM and LRTEETGO are part of the LRTEETAB array of structures (see Table 64 on page 146).

Accounting and Availability Data Section

The accounting and availability data section is used with session monitor external log record subtypes 2, 3, 4, and 5.

Table 66. Format of the Accounting and Availability Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LACCREVL	Revision level: X'0002'	Binary
002	002	2	—	Reserved	—
004	004	8	LACCBEGT	Collection period begin time stamp [1]	Binary
012	00C	8	LACCENDT	Collection period end time stamp [1]	Binary
020	014	4	LACCPCBC	Number of control PIUs sent from primary to secondary[3]	Binary
024	018	4	LACCPCCC	Number of control characters sent from primary to secondary[3]	Binary
028	01C	4	LACCSCBC	Number of control PIUs sent from secondary to primary[3]	Binary
032	020	4	LACCSCCC	Number of control characters sent from secondary to primary[3]	Binary
036	024	4	LACCPTBC	Number of text PIUs sent from primary to secondary[3]	Binary
040	028	4	LACCPTCC	Number of text characters sent from primary to secondary[3]	Binary
044	02C	4	LACCSTBC	Number of text PIUs sent from secondary to primary[3]	Binary
048	030	4	LACCSTCC	Number of text characters sent secondary to primary[3]	Binary

Notes:

1. The first four bytes of the time stamp are local time in store-clock format. The last four bytes are the conversion factor from GMT to local time. (Example: 982B5412 FFFFCA5B.) This time stamp yields an approximate resolution of 1.04 seconds.
2. The session monitor uses the indicators in the first byte of the RH to select control and text PIUs. BSC connections do not have control PIUs.
3. If SESSTATS=AVAIL, this field is zero.

Session Response Time Data Section

The session response time data section is used with session monitor external log record subtypes 1, 2, and 5.

Table 67. Format of the Session Response Time Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LRTMREVL	Revision level: X'0001'	Binary
002	002	8	LRTMCOLB	Collection period begin time stamp [1]	Binary
010	00A	8	LRTMCOLE	Collection period end time stamp [1]	Binary
018	012	2	LRTMOBJP	Objective percentage (default is 0)	Binary

Table 67. Format of the Session Response Time Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
020	014	2	LRTMOBJB	Objective counter number (default is 1)	Binary
022	016	1	LRTMDEF	Response time definition (default is F (first))	Char
023	017	1	LRTMOBJF	Objective met indicator: **Y** Yes **N** No	Char
024	018	4	LRTMTRAN	Number of transactions measured	Binary
028	01C	4	LRTMTOTT	Total response time [2]	Binary
032	020	16	LRTMBNDS	Four 4-byte fields containing counter boundaries [2]	Binary
048	030	20	LRTMBKTS	Five 4-byte fields with contents of counters	Binary
068	044	4	LRTMOBJT	Objective response time [2]	Binary

Notes:

1. For LRTMCOLB and LRTMCOLE, the first four bytes of the time stamp are the local time in store-clock format, and the last four are the conversion factor from GMT to local time. (Example: 982B5412 FFFFCA5B.) This time stamp yields an approximate resolution of 1.04 seconds.
2. LRTMTOTT, LRTMBNDS, and LRTMOBJT are in tenths of seconds.

Event Counter Data Section

The event counter data section is used only with session monitor external log record subtype 8 (storage and event counter record).

Table 68. Format of the Event Counter Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	8	LEVNSTIM	Start of the recording period [1]	Binary
008	008	8	LEVNETIM	End of the recording period [1]	Binary
016	010	8	LEVNDMID	Domain ID (NCCF ID)	Binary
024	018	4	LEVNPIUB	PIU trace buffers processed	Binary
028	01C	4	LEVNPIUS	PIUs processed [2]	Binary
032	020	4	LEVNSAWB	SAW buffers processed [3]	Binary
036	024	4	LEVNSESS	Session start notifications	Binary
040	028	4	LEVNSESE	Session end notifications	Binary
044	02C	4	LEVNSESR	Sessions recorded to VSAM	Binary

Table 68. Format of the Event Counter Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type

Notes:

1. For LEVNSTIM and LEVNETIM, the first four bytes of the time stamp are local time in store-clock format. The last four bytes are the conversion factor from GMT to local time. (Example: 982B5412 FFFFCA5B.) This time stamp yields an approximate resolution of 1.04 seconds.
2. Session awareness (SAW) is a VTAM-session monitor interface through which information about network session activity is exchanged.
3. Refer to *Systems Network Architecture Formats* for more information about PIU.

Session Awareness (SAW) Counter Data Section

The session awareness (saw) counter data section is used only with session monitor external log record subtype 8 (storage and event counter record).

Table 69. Format of the Session Awareness Counters Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LSAWREVL	Revision level: X'0003'	Binary
002	002	2	—	Reserved	—
004	004	4	LSAWASBC	Number of nonfiltered sessions	Binary
008	008	4	LSAWASBM	Session highwater mark	Binary
012	00C	4	LSAWFLTC	Sessions being filtered [1]	Binary
016	010	4	LSAWFLTM	Filter highwater mark [2]	Binary
020	014	4	LSAWHSTE	Sessions with an endpoint in the host	Binary
024	018	4	LSAWRTMC	Sessions keeping RTM data	Binary
028	01C	4	LSAWXNTC	Sessions keeping cross-network data	Binary
032	020	4	LSAWDOMC	Sessions keeping domain data	Binary
036	024	4	LSAWACTC	Sessions keeping accounting data	Binary
040	028	4	LSAWSSCP	Number of SSCP-SSCP sessions	Binary
044	02C	4	LSAWSCPM	SSCP-SSCP highwater mark	Binary
048	030	4	LSAWSPPU	Number of SSCP-PU sessions	Binary
052	034	4	LSAWSPUM	SSCP-PU highwater mark	Binary
056	038	4	LSAWSPLU	Number of SSCP-LU sessions	Binary
060	03C	4	LSAWSLUM	SSCP-LU highwater mark	Binary
064	040	4	LSAWLULU	Number of LU-LU sessions	Binary
068	044	4	LSAWLULM	LU-LU highwater mark	Binary
072	048	4	LSAWRCDQ	Sessions waiting to be recorded	Binary
076	04C	4	LSAWRDQM	Record queue highwater mark	Binary
080	050	4	LSAWPSPU	Number of SSCP-PU pseudo sessions	Binary

Table 69. Format of the Session Awareness Counters Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
084	054	4	LSAWPSPM	SSCP-PU pseudo sessions highwater mark	Binary
088	058	4	LSAWCPCP	Number of CP-CP sessions	Binary
092	05C	4	LSAWCPCM	CP-CP highwater mark	Binary
096	060	4	LSAWRSCV	Sessions keeping RSCV data	Binary

Notes:

1. LSAWFLTC is the sum of the current sessions-filtered counts from session monitor and VTAM. SAW=NO filtering should be done at VTAM rather than at the NetView program.
2. LSAWFLTM is the sum of the sessions-filtered high-water counts from session monitor and VTAM since the last RECORD STRGDATA request.

Resource (ARB) Counter Data Section

The resource (ARB) counter data section is used only with session monitor external log record subtype 8 (storage and event counter record). The session monitor creates active resource control blocks (ARBs) for all resources involved in sessions known to the session monitor.

Table 70. Format of the Resource Counter Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	4	LARBCNT	Number of resource control blocks	Binary
004	004	4	LARBMAX	ARB highwater mark	Binary
008	008	4	LARBSSCP	Number of SSCP resource control blocks	Binary
012	00C	4	LARBSCPM	SSCP ARB highwater mark	Binary
016	010	4	LARBPU	Number of PU resource control blocks	Binary
020	014	4	LARBPUMX	PU ARB highwater mark	Binary
024	018	4	LARBLU	Number of LU resource control blocks	Binary
028	01C	4	LARBLUMX	LU ARB highwater mark	Binary
032	020	4	LARBLNK	Number of link/CUA ARBs	Binary
036	024	4	LARBLNKM	Link/CUA highwater mark	Binary

Storage Counter Data Section

The storage counter data section is used only with session monitor external log record subtype 8 (storage and event counter record).

Table 71. Format of the Storage Counter Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	4	LSTGRTM	The number of bytes of storage used for response time monitor (RTM) data.	Binary
004	004	4	LSTGPARM	The number of bytes of storage used for session parameter (PARM) data.	Binary
008	008	4	LSTGTRCE	The number of bytes of storage used for session trace (TRACE) data. This includes PIU trace, boundary function trace, and gateway trace.	Binary
012	00C	4	LSTGASB	The number of bytes of storage used for active session block (ASB) control blocks.	Binary
016	010	4	LSTGARB	The number of bytes of storage used for active resource block (ARB) control blocks.	Binary
020	014	4	LSTGACCT	The number of bytes of storage used for accounting (ACCT) data.	Binary
024	018	4	LSTGRSCV	The number of bytes of storage used for route selection control vector (RSCV) data.	Binary
028	01C	2	LSTGREVL	Revision level: X'0002'	Binary

Advanced Peer-to-Peer Networking Route Data Section

For Advanced Peer-to-Peer Networking, this data section is used with all session monitor external log record subtypes, except subtype 8 (storage and event counter record).

Table 72. Format of the Advanced Peer-to-Peer Networking Route Data Section

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LARTREVL	Revision level: X'0001'	Binary
002	002	2	LARTNUMT	Number of route elements in LARTTABL. This is the number of nodes in the Advanced Peer-to-Peer Networking subnetwork, including the primary CP.	Binary

Table 72. Format of the Advanced Peer-to-Peer Networking Route Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
004	004	1	LARTRVFL	RSCV flags: **Bit** **Meaning** **0-1** TG not valid **00** No IN-TG **01** IN-TG at end **10** IN-TG at start **11** IN-TG at start and end **2** First RSCV error (VTAM or NCP storage problem) **0** No first RSCV error **1** First RSCV error **3** First RSCV flag **0** First RSCV not present **1** First RSCV present **4** Second RSCV error (VTAM or NCP storage problem) **0** No second RSCV error **1** Second RSCV error **5** Second RSCV flag **0** Second RSCV not present **1** Second RSCV present **6-7** Reserved	Binary

Table 72. Format of the Advanced Peer-to-Peer Networking Route Data Section (continued)

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
005	005	—	LARTTABL	An array of Advanced Peer-to-Peer Networking route elements. See Table 73 on page 153 for format of an Advanced Peer-to-Peer Networking route element.	—

Table 73. Format of an Advanced Peer-to-Peer Networking Route Element

Offset Dec.	Offset Hex.	Length in Bytes	Field Name	Description	Type
000	000	2	LARTTGNU	Transmission group (TG) number	Binary
002	002	8	LARTTGNE	TG partner network name	Char
010	00A	8	LARTTGNA	TG partner node name	Char
018	012	1	LARTTGFL	TG flag descriptor	Binary

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road

Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Programming Interfaces

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM Z NetView.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml .

Adobe is a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

SC27-2870-03