Tivoli® NetView® for z/OS™

Customization: Using Assembler

Version 5 Release 1
Customization: Using Assembler

Version 5 Release 1
Tivoli NetView for z/OS Customization: Using Assembler

Copyright Notice

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

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

Trademarks

IBM, the IBM logo, Tivoli, the Tivoli logo, BookManager, IBMLink, MVS/DFP, MVS/ESA, MVS/XA, NetView, OS/390, S/370, Tivoli Enterprise, TME, VTAM, and z/OS are trademarks or registered trademarks of International Business Machines Corporation or Tivoli Systems, Inc. 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 company, product, and service names may be trademarks or service marks of others.

Notices

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

Programming Interfaces

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

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

**Part 1. Writing in Assembler Language** ........................................ 1

**Chapter 1. Getting Started** ................................................... 3
The Benefits of Using Assembler Language ....................................... 3
Testing Your Program .................................................................. 3

**Chapter 2. Designing Assembler Modules** ..................................... 5
Task Structure ........................................................................... 5
General Coding Guidelines .......................................................... 6
  Processing Environment .......................................................... 6
  Coding Requirements ............................................................ 7
Control Blocks ........................................................................... 8
  Control Block Requirements .................................................... 8
  Establishing Addressability ..................................................... 9
Work Block Services ................................................................... 11
Basic Module Services ............................................................... 11
  Standard NetView Buffer Structure ......................................... 11
  Dynamic Storage Services ........................................................ 12
  Releasing Queued Storage ....................................................... 12
  Message Processing .............................................................. 12
  Invoking Commands ................................................................ 17
Scheduling Commands in a Cross-Domain Environment ....................... 19
Additional Services ................................................................. 20
  Loading and Deleting Modules in Virtual Storage ......................... 20
  Posting and Waiting on Event Control Block (ECB) Services ............ 20
  Disk Services .......................................................................... 20
  Parsing .................................................................................. 21
  Command Authorization Checking ........................................... 21
  Named Storage ....................................................................... 22
  Sending and Receiving Data Using the MS Transport ....................... 22
External Function Parameter List ................................................. 102
Argument List ........................................................................ 102
Evaluation Block ..................................................................... 103
Directory for Function Packages .................................................. 104
Format of the Entry in the Directory .............................................. 105
Example of a User Function Directory .......................................... 106

Part 2. Reference ........................................................................ 107

Chapter 7. Control Blocks .............................................................. 109

BUFHDR: Buffer Header ................................................................. 109
  Values for HDRIND Fields ......................................................... 112
  Values for HDRMTYPE Fields ...................................................... 112
  BUFHDR Presentation Extension .................................................. 115
  Example Scenario of BUFHDR Usage .......................................... 116
DSIAIFRO: Automation Vector Extensions ....................................... 116
  Modifications of the AIFR Format ............................................... 117
  Format of the Message Control Object ....................................... 118
  Format of the Source Object ...................................................... 119
DSICBH: Control Block Header ...................................................... 120
DSICWB: Command Work Block .................................................. 121
DSIDSB: Data Services Block ....................................................... 122
DSIDSRB: Data Services Request Block ......................................... 122
DSIDTR: Data Transport Request ................................................ 124
DSIELB: External Logging Block .................................................. 124
DSIIFR: Internal Function Request ................................................ 125
  AIFR—Automation Internal Function Request ............................. 125
  Automation Internal Function Request Routing List ...................... 142
DSILOGDS: NetView Log ............................................................... 143
DSIMVT: Main Vector Table ........................................................ 143
  MVTAIDFT–DSIEX16 Interface Data ............................................ 146
DSIPDB: Parse Descriptor Block ................................................... 147
DSISCE: System Command Entry ................................................ 148
DSISCT: System Command Table ................................................. 149
DSISVL: Service Routine Vector List ............................................. 149
DSISWB: Service Work Block ...................................................... 149
DSITECBR: Branch Table of ECB Processor Load Modules ............ 150
DSITIB: Task Information Block .................................................. 150
DSITVB: Task Vector Block ......................................................... 152
DSIUSE: Installation Exit Parameter List ........................................ 156
DSIXRCMD: RUNCMD Installation Exit Buffer .............................. 160

Chapter 8. Macros ...................................................................... 161

DSIAUTO: Invoke Automation Services .......................................... 161
  Return Codes in Register 15 ....................................................... 162
DSIBAM: Build Automation Message ............................................ 162
DSIBAMKW: Build Automation Message Keyword ....................... 164
DSICBS: Control Block Services .................................................. 165
DSICES: Command Entry Services .............................................. 166
  Return Codes in Register 15 ....................................................... 168
DSICVTHE: Convert to Hexadecimal ............................................. 169
DSIC2T: Code Point Translation Service Reference ....................... 169
  Return Codes in Register 15 ....................................................... 171
DSIDATIM: Date and Time ........................................................... 171
DSIDEL: Delete User-Defined Module ......................................... 172
  Return Codes in Register 15 ....................................................... 172
DSIDKS: Disk Services ............................................................... 172
  Return Codes in Register 15 ....................................................... 174
DSIFIND: Find Long-Running Command Storage ......................... 175
  Return Codes in Register 15 ....................................................... 177
DSIFRE: Free Storage ................................ 177
    Return Codes in Register 15 ............................ 179
DSIFREBS: Call DSIFREBS Service ............................. 180
DSIGET: Get Storage ........................................ 180
    Return Codes in Register 15 ............................ 182
DSIGETDS: Data Queue Manipulation Service ..................... 182
    Return Codes in Register 15 ............................ 184
DSIHREGS: High-Performance Transport Application Registration .... 184
    Return Codes in Register 15 ............................ 188
DSIHSNDS: Send High-Performance Message Unit ...................... 189
    Return Codes in Register 15 ............................ 194
DSIID: Store SYSMOD Level in CSECT ............................. 195
    Return Codes in Register 15 ............................ 195
DSIKVS: Keyword/Value Services ................................ 196
    Return Codes in Register 15 ............................ 197
DSILCS: Obtain/Release Control Blocks ............................ 200
    Return Codes in Register 15 ............................ 200
DSILOD: Load User-Defined Module ................................ 201
    Return Codes in Register 15 ............................ 201
DSIMBS: Message Build Services ................................ 203
    Return Codes in Register 15 ............................ 204
DSIMDS: Message Definition Services ............................. 204
    Defining Messages on Disk. ................................ 206
    User Message Definition Module Examples ...................... 207
DSIMMDBS: Call DSIMMDB Service ................................ 208
DSIMQS: Message Queuing Services ................................ 208
    Return Codes in Register 0 ............................. 212
    Return Codes in Register 15 ............................ 212
DSINOR: Resource Object Data Manager ............................ 213
    Return Codes in Register 15 ............................ 215
DSIPAS: Parameter Alias Services ................................ 216
    Return Codes in Register 15 ............................ 217
DSIPOP: Remove Long-Running Command ................................ 217
    Return Codes in Register 15 ............................ 219
DSIPOS: ECB Post Services .................................... 219
DSIPRS: Parsing Services ...................................... 220
    Return Codes in Register 15 ............................ 220
    Parsing Services Module Examples ............................ 222
DSIPOP: Remove Long-Running Command ................................ 222
    Return Codes in Register 15 ............................ 222
DSIPUSH: Establish Long-Running Command ........................ 227
    Return Codes in Register 15 ............................ 229
DSIOQS: Query Operator Services ................................ 234
    Return Codes in Register 15 ............................ 235
DSIQRS: Query Resource Services ................................ 235
    Return Codes in Register 15 ............................ 237
MNOTES Issued ................................................. 238
DSIRDS: Resource Definition Services ............................ 238
DSIRXEB: Get an EVALBLOK .................................... 238
    Return Codes in Register 15 ............................ 239
DSISRCMV: Search for Subvector or Subfield ......................... 239
    Return Codes in Register 15 ............................ 240
DSISYS: Operating System Indicator .............................. 240
    Return Codes in Register 15 ............................ 241
DSITECBS: Manage a Dynamic ECB List for DSTs ....................... 242
    Return Codes in Register 15 ............................ 242
DSIVARS: Variable Services .................................... 243
    Return Codes in Register 15 ............................ 246
DSIWAT: ECB Wait Services .................................... 247
DSIWCS: Write Console Services .................................. 248
    Return Code in Register 15 ............................ 248
DSIWLS: Write Log Services .................................... 249
    Return Codes in Register 15 ............................ 250
DSIZCMS: CNM Data Services .................................... 251
Preface

This document describes the ways you can tailor or supplement Tivoli® NetView® for z/OS™ to satisfy unique requirements or operating procedures.

This document describes the uses and advantages of user-written programs (exit routines, command processors, and subtasks). It provides instructions that guide the programmer through the mechanics of designing, writing, and installing these programs.

Who Should Read This Document

This document is intended for experienced system programmers who are knowledgeable in assembler language and familiar with the functions that the NetView program provides.

Secondary users of this document include operators, network planners, designers, and system analysts. Secondary users can also include IBM® marketing representatives and instructors.

What This Document Contains

To use this manual most effectively, you should have in mind a specific command processor or installation exit that you want to write in assembler language. For example, you might want to write a command processor to run under a data services task (DST) to store data in a VSAM file, or you might want a command processor to run on an operator station task (OST) to display information about an operator’s screen. The Tivoli NetView for z/OS Customization Guide contains information to help you decide which command processors and installation exits you need to write to build your application, and what the appropriate language is for each of those pieces of code.

"Chapter 1. Getting Started” on page 3 describes the assembler application program interface.

"Chapter 2. Designing Assembler Modules” on page 5 presents information about NetView services available for user-coded command processors, installation exits, and user subtasks.

"Chapter 3. Writing Installation Exit Routines” on page 25 illustrates how to design, code, and install installation exit routines that take advantage of the NetView exits at strategic processing points.

"Chapter 4. Writing Command Processors” on page 61 illustrates how to design, code, and install assembler language command processors for NetView.

"Chapter 5. Writing User Subtasks” on page 83 discusses user subtasks and illustrates how to write, process, and install them.

"Chapter 6. Writing User Function Directories” on page 101 explains how to write function package directories and add them to the NetView function package table.
Chapter 7. Control Blocks on page 109 describes control block fields related to customization interfaces, and describes the source object, which is used with a message data block (MDB), and the message control object that is contained in the AIFR extension.

Chapter 8. Macros on page 161 describes the purpose and coding of NetView macros.

Chapter 9. Called Service Routines on page 277 explains called service routines and lists interfaces for the assembler language.

Appendix A. Assembler Samples on page 283 contains a table of the assembler samples shipped with NetView.

Appendix B. MDB Field to AIFR Cross-Reference Table on page 289 contains a cross-reference table for MDB control block fields.

Appendix C. Writing an Automation Table Function on page 297 illustrates how to design, code, and install an assembler language automation table function (ATF) for NetView.

Appendix D. Assembler Macros and HLL Service Routine Interfaces for NetView on page 303 lists each assembler macro and its corresponding HLL (high-level language) service routine.

Publications

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

Prerequisite and Related Documents

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

You can find additional product information on these Internet sites:

<table>
<thead>
<tr>
<th>Resource Web sites</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td><a href="http://www.ibm.com/">http://www.ibm.com/</a></td>
</tr>
<tr>
<td>Tivoli Systems</td>
<td><a href="http://www.tivoli.com/">http://www.tivoli.com/</a></td>
</tr>
<tr>
<td>Tivoli NetView for z/OS</td>
<td><a href="http://www.tivoli.com/nv390">http://www.tivoli.com/nv390</a></td>
</tr>
</tbody>
</table>

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

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

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

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

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

Ordering Publications

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

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

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

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

Providing Feedback about Publications

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

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

Contacting Customer Support

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

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

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

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

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

http://www.tivoli.com/nv390

Under Related Documents, select Other Online Sources.

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

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

Keyboard Access

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

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

Conventions Used in This Document

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

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

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

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

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

Platform-specific Information

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

Terminology

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

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

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

**MVS** MVS/ESA™, OS/390, or z/OS operating systems.

**RACF®** RACF is a component of the SecureWay® Security Server for z/OS and OS/390, providing the functions of authentication and access control for OS/390 and z/OS resources and data, including the ability to control access to DB2® objects using RACF profiles. Refer to:
Tivoli Enterprise™ software
Tivoli software that manages large business networks.

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

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

V and R
Specifies the version and release.

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

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

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

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

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

Required Syntax
The command name, required keywords, variables, and operands are always on the main syntax line. Figure 1 on page xiv specifies that the resname variable must be used for the CCPLOADF command.
Keywords and operands are written in uppercase letters. Lowercase letters indicate variables such as values or names that you supply. In Figure 2, MEMBER is an operand and membername is a variable that defines the name of the data set member for that operand.

**Figure 1. Required Syntax Elements**

**Figure 2. Syntax for Variables**

**Optional Keywords and Variables**

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

**DISPREG**

**Figure 3. Optional Syntax Elements**

**Default Values**

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

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

Figure 4 on page xv shows the default keyword STEP above the main line and the rest of the optional keywords below the main line. It also shows the default values for operands MODNAME=* and OPTION=* above and below the main line.
Long Syntax Diagrams

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

Syntax Fragments

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

Required commas and parentheses are included in the syntax diagram. When an operand has more than one value, the values are typically enclosed in parentheses and separated by commas. In Figure 6 on page xvii, the OP operand, for example, contains commas to indicate that you can specify multiple values for the testop variable.
If a command requires positional commas to separate keywords and variables, the commas are shown before the keyword or variable, as in Figure 4 on page xv. For example, to specify the BOSESS command with the \textit{sessid} variable, enter:

\texttt{NCCF BOSESS applid,,sessid}

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

**Highlighting, Brackets, and Braces**

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

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

**Table 3. Syntax Elements Examples**

<table>
<thead>
<tr>
<th>This element...</th>
<th>Looks like this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>CCLOADF</td>
</tr>
<tr>
<td>Variable</td>
<td>\textit{resname}</td>
</tr>
<tr>
<td>Operand</td>
<td>MEMBER=\texttt{membername}</td>
</tr>
<tr>
<td>Default</td>
<td>\texttt{today} or INCL</td>
</tr>
</tbody>
</table>
Abbreviations

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

Tivoli NetView for z/OS enables you to manage complex, multivendor networks and systems from a single point. Before reading this chapter, refer to the information about NetView customization facilities and designing user-written functions in the Tivoli NetView for z/OS Customization Guide.

Also, you should understand the requirements and environments of command processors at the assembler level if you write assembler language subroutines for your high-level language command processors.

The Benefits of Using Assembler Language

Although coding in assembler language requires more effort, this language has greater flexibility than other languages NetView supports. The advantages of assembler language include:

• **Display flexibility.** Although full-screen display support is provided to all languages through the VIEW command, you can take advantage of special features of your workstation, such as wide screens or cursor-dependent functions. By writing in assembler language, you gain direct access to the 3270 data-stream commands.

• **Special data handling.** For example:
  – Reentrant updates to a global data structure
  – Complex functions at operator logoff or abend-reinstatement

• **Privileged functions.** Access to all system functions and macros.

• **Interaction with other commands.** You can examine the status of, or wait on, asynchronous events other than the standard set that the VIEW command provides. These include timed events or completion of commands under another task.

• **Performance.** While high-level language routines supported by NetView usually run faster than interpreted command lists, some kinds of data manipulation are faster with assembler language.

Testing Your Program

Before putting your program into production, develop test scenarios for the major functions. Use the NetView trace facility when testing to analyze errors or abends that the code generates. The format of the TRACE command is found in the NetView online help. Running a trace is particularly helpful for verifying input to installation exit routines, DSIPSS, and DSIMQS. A trace also verifies that your storage is balanced, that is, all the storage your code gets (using DSIGET) is also freed by your code (using DSIFRE).

When testing new command processors, use RES=N in the CMDMDL definition statement to replace the object code without recycling NetView.

To avoid possible conflict between your code and NetView services currently in use, consider running the code on a separate NetView program or test machine, or during a time of day when NetView is not used heavily. To test code, you must
Application Program Interfaces

restart NetView, except for predefined nonresident command processors. If there is little risk of conflict, you can simply add code to the NetView production library currently in use.

When the test system is running, perform the test scenarios and verify that the results and output of your code are correct. If they are correct, put the code into production and begin using it normally.

Note: For information on methods of testing automation, refer to Tivoli NetView for z/OS Automation Guide.
Chapter 2. Designing Assembler Modules

This chapter reviews NetView services available for user-coded command processors, installation exits, and user subtasks. See "Chapter 8. Macros" on page 161 for more information about the NetView service macros described in this chapter. See "Appendix A. Assembler Samples" on page 283 for assembler coding samples using these macros.

Task Structure

The NetView task environment consists of a main task (MNT) and six types of subtasks.

The main task initializes NetView. It provides an environment for the subtasks and oversees their creation and cleanup. The main task also provides limited installation exits that can be used to modify processing under the main task.

The six subtasks are:

- Operator station task (OST)
  An operator station task enables an operator to enter commands and receive messages. An OST is started for each operator at logon.
  An OST is also started for each automated operator (autotask). Autotasks are independent of VTAM and terminals. Only line-mode commands (no full-screen commands) can be run by autotasks, which are started with the AUTOTASK command and the RMTCMD command.

- NetView-NetView task (NNT)
  A NetView-NetView task gives an operator an opportunity to access another NetView program, usually in another domain. The task also enables the operator to send commands and receive messages from the NetView program. The START DOMAIN command creates an NNT in the other domain.

- Hardcopy task (HCT)
  A hardcopy task logs the activity of one or more OSTs using 328x printers. The START HCL command starts an HCT.

- Primary program operator interface task (PPT)
  The primary program operator interface task, receives unsolicited messages from VTAM and message traffic exchanged between the system console and VTAM. The PPT opens a (primary) programmed operator interface to VTAM to receive these messages. Each NetView program has one PPT. The PPT is started when the NetView program starts. When more than one NetView program is running in the same system, the PPT is normally made a secondary POI receiver in all but one of the NetView versions. Timing is no longer done by the PPT, but is now done by the DSITIMMT task.

- Optional tasks (OPT)
  Optional tasks are user-defined subtasks that can provide increased flexibility beyond the subtasks that the NetView program provides. For example, you can use an OPT to provide unique command interfaces or more precise control of work dispatching than standard command processors running under a DST-based task could provide. You can define the OPT to start when the NetView program starts or when you issue a START TASK command.

- Data services task (DST)
Designing Assembler Modules

Data services tasks provide command, message, and exit functions. In addition, DSTs can manage VSAM or communication network management (CNM) facilities. You can create new subtasks to perform any or all of these functions. The DST can be started when NetView program starts or when you issue a START TASK command.

A DST is a subset of an optional task (OPT). If you write code to check the CBHTYPE of a control block under a DST, you find an X'05', which is an OPT.

Note: For more information about the NetView task structure, refer to the Tivoli NetView for z/OS Diagnosis Guide.

General Coding Guidelines

This section contains some general guidelines to follow when coding in the NetView program.

Processing Environment

NetView code and user-written code have control in a problem program state. Both types of code also have control with a user memory protection key. Unless your system programmers set up special provisions, this is key 8 in MVS. NetView and all user-written code given control under the NetView program are authorized. You have the option of invoking privileged system services, entering supervisor state, or changing the memory protection key. However, you should return control to the NetView program in the same program state and memory protection key as when you entered the user code.

Except as noted in Chapter 7, Control Blocks on page 109, all NetView control blocks are accessible with the user key. Unless Chapter 8, Macros on page 161 specifically notes otherwise, you need to invoke all NetView services in problem program state and the user key.

When the NetView program calls user-written code, the TVBINXIT bit can be tested to determine the environment under which the user-written code is processing. When a NetView task is initially dispatched, the TVBINXIT bit is off. If the task calls user-written code at this time, the code processes under the mainline environment.

When the operating system interrupts mainline processing, the NetView program sets the TVBINXIT bit on in the DSITVB control block of the processing task. Any user-written code called at this time is processed under the TVBINXIT=ON environment.

Each task has unique restrictions that limit the types of user-written code it can process. Table 4 indicates the types of code that can run under each task in the mainline environment. The figure also indicates the types of code that can run when the TVBINXIT bit in the TVB is set on. Because you define the commands and functions that run under an OPT, that subtask is not included in Table 4.

Table 4. Task Environment for User-Written Programming

<table>
<thead>
<tr>
<th>Main Task</th>
<th>OST</th>
<th>NNT</th>
<th>HCT</th>
<th>PPT</th>
<th>DST</th>
<th>TVBINXIT Set On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation exit routines</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Regular command processors</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4. Task Environment for User-Written Programming (continued)

<table>
<thead>
<tr>
<th></th>
<th>Main Task</th>
<th>OST</th>
<th>NNT</th>
<th>HCT</th>
<th>PPT</th>
<th>DST</th>
<th>TVBINXIT Set On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate command</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>processors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data services command</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>processors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Coding Requirements

This section describes the general guidelines to consider when writing your code. Control blocks and macros in this section are described in more detail in “Chapter 7. Control Blocks” on page 109, and in “Chapter 8. Macros” on page 161.

Limitations When TVBINXIT Is Active

See Table 5 on page 25 to determine the installation exits that must interrogate the TVBINXIT bit. An installation exit should not change the value of TVBINXIT.

When your code receives control with TVBINXIT set on, the following limitations apply:

• As a general coding guideline, your installation exits should check TVBINXIT and return with return code zero if the installation exit was invoked with TVBINXIT set on.

• Do not code your program to cause a system wait state. A system wait can produce an interlock. For example, if an interruption request block (IRB) exit waits for a resource being used by the mainline routine, the resource is not released, because the IRB exit is interrupting the mainline routine.

  To avoid causing a system wait state:
  – Do not issue macro DSIWAT.
  – Do not issue macro DSIDKS or any other disk services macros.

• Only immediate commands can be called directly. Use the DSIMQS macro to invoke regular commands. For more information, see “Invoking Commands” on page 17.

Names of Variables

Do not use the following prefixes for any variable or message you name in your code:

• Any NetView component prefix, for example:
  - AAU
  - BNT
  - EGV
  - EZL
  - FLB

• Any NetView control block suffix, such as SWB or PDB

Macro Use

When a NetView macro and an operating system macro perform equivalent services, use the NetView macro. This ensures that the source code for your program can be transported across NetView operating systems. If you use an operating system macro, be careful that its function does not conflict with a function the NetView program might be performing. For example, if the operating system macro STIMER is issued under the DSITIMMT, NetView timer services are disrupted.
Register Use
Use the following guidelines when using registers for installation exit routines, command processors, and subtasks:

0–15  Save all registers at entry to the user code, and restore them before returning control to the NetView program.

0, 2–12  Do not rely on the contents of these registers for constant values as entry to the user code. Their contents vary.

0, 1, 14, 15  Do not rely on these registers after issuing a NetView macro. These registers are used during macro expansion.

13  Ensure that this register contains the address of a standard 72-byte save area.

14  Use this register to specify the location in the NetView program to return control to.

15  Set this register with your return code when exiting from your program.

Reentrant Code
Ensure that your code is reentrant to enable the one copy of the program to be used concurrently by more than one task. For example, a command processor can be invoked from two or more tasks simultaneously and execute simultaneously under multiple tasks.

Reentrant code is not required for certain installation exits or for optional tasks. See Table 5 on page 25 for more information.

Addressing and Residency Mode Considerations
The NetView program supports any valid combination of addressing mode (AMODE) and residency mode (RMODE). The NetView program provides the appropriate services, depending on the current mode of your code. In addition, the NetView program provides macro parameters where necessary to enable you to specify 24-bit or 31-bit addressing for your special requirements. (New called-assembler services require 31-bit addressing, as described in “Chapter 9. Called Service Routines” on page 277.) The NetView program also provides appropriate residency for the control blocks your code accesses, such as USE, CWB, and BUFHDR. For example, if your command processor is loaded with AMODE=24, the CWB passed to it resides below 16 megabytes (MB). The Message Queueing Service (DSIMQS) will only ensure that the message buffers are below the 16 MB line if a user-optional task is link-edited with RMODE(24) and AMODE(24).

To conserve storage below the 16 MB line and therefore minimize the resources needed to copy buffers, 31-bit residency is recommended except when limited by your use of interfaces.

Control Blocks
Your code must include the necessary control block mappings for the module you are writing and establish addressability to each of the control blocks. For the specific information you need to write these modules, see one of the following chapters:

- “Chapter 3. Writing Installation Exit Routines” on page 23
- “Chapter 4. Writing Command Processors” on page 61
Control Block Requirements

At source program assembly, include DSECTs for the NetView control blocks that have fields used by the program. The program uses the following control blocks:

- For installation exit routines, include the installation exit parameter list (DSIUSE). Because the installation exit parameter list (USE) contains the addresses of the task vector block (TVB), service work block (SWB), parse descriptor block (PDB), and buffer header (BUFHDR) (defined with the DSITIB DSECT), you include these control blocks.

  \textbf{Note:} Each TVB addresses an associated task information block (TIB) and the main vector table (MVT).

  - For command processors, include the command work block (CWB). Because the CWB contains the addresses of an associated SWB, PDB, and BUFHDR (defined with the TIB DSECT), you also need to include these control blocks. In addition, data services command processors (DSCPs) require the data services request block (DSRB).

  - For user subtasks, include the TVB and any other control blocks necessary for your user-defined task.

Depending on the particular service facilities your program uses, you might need to include other control blocks.

Figure 7 on page 10 illustrates the interconnections among the control blocks. The field displacements suggested in this figure are not representative of the field displacements. The subtasks diagram in this figure shows that for each type of user-written code, the TVB provides access to the MVT and to the SVL.
Use macro DSICBS to include DSECTs for the appropriate control blocks. For example, you can code the DSICBS macro in a command processor as follows:

```
DSICBS DSICWB,DSISWB,DSIMVT,DSIPDB,DSITVB,PRINT=NO
```

This instruction includes the CWB, SWB, MVT, PDB, and TVB. This instruction also specifies, with PRINT=NO, that control block expansions cannot be printed.
Establishing Addressability

Establish addressability to a control block before referencing fields within that control block. The following example shows how you can establish addressability to the MVT for a command processor:

```
USING DSICWB,1
L REG10,CWBTIB
USING DSITIB,REG10
L REG11,TIBTVB
USING DSITVB,REG11
L REG12,TVMVT
USING DSIMVT,REG12
```

Work Block Services

You need the service work block (SWB) when your code invokes NetView service facilities. The CWB is used as command processor input. Macro DSILCS obtains and releases SWBs or CWBs, as shown in the following example:

```
DSILCS CBADDR=MYSWBPTR,SWB=GET
LTR REG15,REG15
BNZ ERRORSWB
L REG2,TVBIB
L REG3,MYSWBPTR
ST REG2,TVBTIB-DSISWB(REG3)
```

```
DSILCS CBADDR=MYSWBPTR,SWB=FREE
```

---

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This statement obtains an SWB and assigns its address to be placed in MYSWBPTR.</td>
</tr>
<tr>
<td>2</td>
<td>This statement tests for a good return code in register 15.</td>
</tr>
<tr>
<td>3</td>
<td>This statement branches to error routine ERRORSWB if the return code is not zero (0).</td>
</tr>
<tr>
<td>4</td>
<td>These three statements initialize the SWBTIB field to your TIB address before you use the SWB, if the macro is successful.</td>
</tr>
<tr>
<td>5</td>
<td>This statement releases the SWB using DSILCS after use.</td>
</tr>
</tbody>
</table>

---

Basic Module Services

This section contains information about the module services provided with the NetView program.

Standard NetView Buffer Structure

All message and command buffers require an initialized buffer header (BUFHDR) structure preceding the message data or command text. BUFHDR is a separate DSECT contained in the task information block (DSITIB) control block. Use DSICBS to include DSITIB when your module references BUFHDR fields. "Chapter 7: Control Blocks" on page 103 illustrates the structure of a buffer header. Figure 19 on page 110 presents an overview of the BUFHDR fields.
Dynamic Storage Services

Storage services enable you to get and to free storage for your program. Use macro DSIGET to get storage and macro DSIFRE to free storage, as shown in the following example:

```
DSIGET LV=4088,A=STORPTR,TASKA=(MYTVBREG),Q=NO
LTR REG15,REG15
BNZ ERRORGET
  :

DSIFRE LV=4088,A=STORPTR,TASKA=(MYTVBREG),Q=NO
```

The preceding example requests that 4088 bytes of storage be obtained. The address of the storage is placed in the fullword named STORPTR. The second and third statements test for a good return code in register 15 before you use the storage. When the storage is obtained, it is cleared to binary zeros. Except when it would cause a new page to be gotten, DSIGET obtains an additional 8 bytes of storage that are used by the NetView program to detect storage overlay conditions.

After use, free the storage using the same value for the Q option as you used previously to get the storage. Also, free non-queued storage; specify the same size that you specified to get the storage (as shown in the example above where DSIGET and DSIFRE both specify LV=4088). If you do not specify the same size on DSIGET and DSIFRE, the NetView program assumes that a storage overlay condition exists and may cause a dump to be taken.

Note: Specify the TASKA keyword because it can help you avoid addressing the wrong TVB. TASKA contains the address of the TVB for the subtask where the code is executing. Use the TASKA parameter for both non-queued (Q=NO) and queued (Q=YES) storage.

Releasing Queued Storage

You can also use the DSIGET Q=YES option to enable the NetView program to release queued storage at logoff or task termination, which facilitates error recovery. The following example illustrates this use:

```
DSIGET LV=4096,A=STORPTR,TASKA=(MYTVBREG),Q=YES,BNDRY=PAGE
LTR REG15,REG15
BNZ ERRORGET
  :

DSIFRE A=STORPTR,TASKA=(MYTVBREG),Q=YES
```

In this example, Q=YES indicates that the NetView program keeps track of the 4096 bytes of storage in an internal queue. To support this internal queue, Q=YES gets 16 more bytes of storage than requested. (Q=NO gets 8 more bytes of storage than requested for storage overlay checking.) If you specify BNDRY=PAGE, the extra bytes are not received, and page alignment is not affected.

Note: The NetView program ignores any storage length indicated by DSIFRE when you specify Q=YES.

Message Processing

Message processing in the NetView program includes:

- Creating messages
- Displaying messages to operators
- Displaying messages to the console
Creating Messages

The message definition service (DSIMDS) macro provides you with a chance to create a load module of user-defined message skeletons. Each defined message skeleton can contain up to nine variable-length text inserts.

The message building service (DSIMBS) macro combines message insert text with the specified message skeleton and returns a completed message buffer, which can be used for displaying the message.

For more information about message macros, see the following references:

- The introductory section of "Chapter 8. Macros" on page 161 for conditions that apply to all NetView macros
- "DSIMBS: Message Build Services" on page 201 for details on using the DSIMBS macro
- "DSIMDS: Message Definition Services" on page 204 for details on using the DSIMDS macro
- The sample library on the product tape for examples of creating user-defined messages with DSIMDS and DSIMBS
Displaying Information to NetView Operators

The primary channels for presenting information to the operator are:

- The terminal screen using macro DSIPSS
- The network log, MVS system log, sequential log, and hardcopy log using macro DSIWLS

Message presentation using the DSIPSS and DSIMQS macros is more complex. Macro DSIPSS can present information in any of the following three screen modes. DSIMQS is limited to standard mode and title-line mode.

Standard Mode

NetView messages are sent to the screen with a user-definable prefix followed by data. The prefix includes fields such as the NetView message type (HDRMTYPE), and the NetView domain name (HDRDOMID), indicating the domain that sent the message. If the message is wider than the screen, it is split between words. The message is continued on the next line, and is indented by a user-customized amount.

A variation on standard mode output is the immediate message. It appears at the bottom of the panel as a single 71-character message with neither prefix nor continuation lines.

To present a message in standard mode, you can use any of these methods:

- Issue DSIPSS TYPE=OUTPUT
- Issue DSIPSS TYPE=IMMED
- Issue DSIMQS to queue a HDRTYPEU message buffer to the OST
- Issue DSIPSS TYPE=FLASH

Refer to sample ADATTIM (CNMS4274) in CNMSAMP for an example of standard mode output.

Title-line Mode

Title-line presentation services send sequences of messages to the operator without allowing other messages to be interspersed. The messages appear on the panel in a tabular format, with one or more title lines. Title-line messages have no prefix and can use the full width of the panel. Messages longer than panel width are truncated. For more information about how to specify this type of output, see "Title-Line Output" on page 13.

Title-line mode messages and system multiline write-to-operator (MLWTO) messages are treated as a single message by:

- DSIEX02A, DSIEX16, and &WAIT in NetView command list language
- TRAP and WAIT in REXX and high-level language command procedures
- The NetView automation table
- The ASSIGN command

When creating your title-line mode messages, ensure that the first line does not conflict with any IBM-supplied message number. A message number format similar to the IBM message number format can be helpful when you use these facilities with your messages.

Refer to sample AMLWTO (CNMS4273) in CNMSAMP for an example of title-line output.

Full-Screen Mode

You can present a full panel of information using 3270 data streams. Full-panel command processors, which run under an OST, are the only
type of user-written code (except for installation exit DSIEX12) that can use full-screen mode. This topic is addressed in “Writing a Full-Screen Command Processor” on page 67.

Refer to sample APSSFULL (CNMS4279) in CNMSAMP for an example of full-screen output.

Title-Line Output: Title-line output is best suited to message groups that can be presented on a single panel. Figure 9 shows an example of title-line output.

![Figure 9. Example of Title-Line Output](image_url)

The first line that the NetView program generates is a user-customized NetView message prefix on a line by itself. This prefix is the same as the one defined under “Standard Mode” on page 14. In Figure 9, the equal sign (=) in “NETV1” is the message type and NETV1 is the domain ID. The separator is followed by the title, which can be one to six lines. The title is followed by the data lines. The panel wraps around until all the data is displayed. The title is redisplayed at the top of each new panel.

Using Title-line Output: To use title-line output, format and send one message buffer for each line of information as follows:

- Set the HDRMTYPE field in the BUFHDR to HDRTYPEL (=). See “Values for HDRMTYPE Fields” on page 113 for instructions.

- As you build each message buffer, do the following:
  - For a line that is part of the title, set HDRIND to HDRLNLBL. Your title can contain from one to six lines.
  - For a data line, set HDRIND to HDRLNDAT.
  - For the last line of data, set HDRIND to HDRLNEND.

- When the message buffer is ready for presentation, do one of the following:
Designing Assembler Modules

- From any NetView subtask, use macro DSIMQS to queue the message buffer to the desired OST. This method is appropriate to use from the destination OST. However, the output is not displayed until you return control to the OST.

- From an OST or NNT, issue macro DSIPSS TYPE=OUTPUT to send the message buffer. You cannot issue this macro to send a multiline message from a routine running with TVBINXIT set on.

In either case, the output is not displayed until the data-end line is sent.

The NetView groups all title-line buffers at the OST until a buffer marked as data end (HDRLNEND) is received. Upon receipt of the end message, the title lines are sent to the panel. These lines appear directly under the previous message. Each data line appears one line at a time. If the message sequence fills one panel and begins another, the title lines are repositioned at the top of the panel, followed by the next data lines. This process continues until all the messages appear on the panel.

Each buffer sent to the panel contains at least one character. To print a blank line, place a blank character (X'40') in the buffer. If a line of title-line output is longer than the width of the panel, the line is truncated to panel width.

Displaying Messages to the System Console

The write-to-console service (DSIWCS) macro displays a message buffer at the system console. The message buffer requires an initialized NetView BUFHDR, and the text is truncated to 120 characters when displayed on the console.

Logging Messages

The write-to-log service (DSIWLS) macro records information in the network log, the MVS system log, a hardcopy log, an external log, or a sequential log.

Network Log, the System Log, and Hardcopy Task: DSIWLS logs data to the network log, the system log, and the hardcopy task. The DEFAULTS and OVERRIDE commands determine the general logging environment. The current logging actions are applied to the message buffer passed to the DSIWLS macro. If the message is passed in an AIFR buffer, the system checks the AIFR setting to determine which logs contain the messages.

External Logging: DSIWLS provides for the logging of data to the NetView-defined external logging task (DSIELTSK). You can define the external logging task at installation to record to the system management facilities (SMF) log or to a user-defined data set. If you use a user-defined data set, you must code the XITXL exit to actually perform the logging of the data.

External logging is a NetView-implemented data services task (DST). See Chapter 8, "Macros" on page 161 for information about DSTs. The XITXL exit is described in "XITXL: External Logging" on page 59.

Sequential Logging: DSIWLS also provides for logging of data to a sequential logging task. The sequential logging task records the data using the basic sequential access method (BSAM). You can define multiple sequential logging tasks at installation.

See sample CNMS4275 in Appendix A, "Assembler Samples" on page 283 for an example of writing to a sequential log.
Note: Refer to the sample library on the product tape for an example of writing to the logs. Refer to the Tivoli NetView for z/OS Installation: Configuring Additional Components for information about installing the external logging task. Refer to the Tivoli NetView for z/OS Application Programmer’s Guide for information about installing sequential logs.

Invoking Commands

You can call commands directly, or you can schedule them. Calling a command directly requires the following:

- You must initialize the command environment (required control blocks must be acquired, and so on).
- You must invoke the command entry service (DSICES) macro to look up the address of the command processor in the NetView command table. Then a branch is performed to the command processor.

To schedule a command, you must use the message queuing service (DSIMQS) macro. Scheduling a command under a task places a command buffer on one of the task’s message queues. Other command buffers can be ahead of the scheduled command’s buffer. When the scheduled command’s buffer is processed off the message queue, the command processor is invoked.

Calling a Command Directly

All of the following steps in this section are required to directly invoke a command processor or command list.

Obtaining a Command Work Block (CWB): A command processor requires a CWB for use as a parameter list, a save area, and a work area. You can build your own CWB or you can obtain a CWB by issuing the DSILCS macro. Before invoking the command processor, you need to store the TIB address in the CWBTIB field of the associated CWB.

Obtaining a Service Work Block (SWB): The routine that invokes a command processor provides an SWB. You can preallocate or obtain the SWB with the DSILCS macro, or the SWB can be one the invoker has passed. This control block must also have its SWBTIB field pointing to the TIB of the task. The SWB address must be stored in CWBSWB.

Building a Command Buffer: A command buffer containing a verb and optional operands invokes each command. The command buffer is prefixed with the BUFHDR. Each BUFHDR field must be initialized except the message command extension HDRMCEXT. The address of this command buffer is stored in CWBBUF.

Obtaining a Parse Descriptor Block (PDB) and Parsing the Command: The routine obtains storage for a PDB so the command processor can parse the command. You can obtain the size of the storage for the PDB by issuing the DSIPRS macro with the PDBSIZE option in the program. See “Parsing” on page 21 for more information.

Identifying the Command Processor Address: After the command is parsed, you can find the command in the NetView system command table (SCT). You can look up the command using DSICES in one of three ways:

- With the parsed command in the PDB
- With the unparsed command in the command buffer (CWBBUF)
- By command processor module name (the module name is known but the verb name can change, as in a synonym)
The DSICES macro returns the address of the command’s system command table entry. The macro returns the address in an area passed on the DSICES macro as the SCTADDR parameter. This address points to an SCT entry, mapped by the system command entry (SCE), and the address is stored in the PDBCMDA field.

When the DSICES macro returns to the caller, the return code indicates whether the command is immediate, regular, or both. When TVBINXIT is on, the PDBIMMED bit must be set on if the DSICES return code indicates that the command to be called was defined as TYPE=I (immediate) or TYPE=B (both).

You can call regular commands only when TVBINXIT is off and only from task types OST, NNT, and PPT. You can call immediate commands only when TVBINXIT is on and only from task types OST and NNT. Unless otherwise noted in this book, do not call NetView-written TYPE=D commands.

**Calling the Command Processor:** When calling the command processor, some of the registers should have the following assignments:

1. Points to the CWB, which next points to the PDB, SWB, TIB, and the command buffer.
13. Points to a save area. It is probably already set there, because a save area is required for the service macros.
14. Contains the return point address.
15. Contains the command processor’s entry point address (found in SCE).

The command processor entry point address is stored in the SCECADDR field of the SCE entry pointed to by the PDBCMDA field. SCECADDR contains the address of the linkage assist routine called DSICMDLD. You should branch to DSICMDLD with a branch and set mode (BSM) or a branch and save and set mode (BASSM). DSICMDLD resolves any residency requirements for data areas being passed. For example, if the caller is in AMODE=31, passes data areas (command buffer, CWB, and so forth) above-the-line, and is calling a command processor which resides below-the-line, DSICMDLD copies all the needed areas into below-the-line storage before calling the target command processor.

For example, to pass a command to VTAM while running under an OST, NNT, or PPT, prepare the input described above. Call the NetView command processor identified by DSICES for your VTAM command.

See sample CNMS4280 in **Appendix A. Assembler Samples** on page 283 for an example of calling a command directly.

**Scheduling Commands Using DSIMQS**

You use the DSIMQS macro to schedule the invocation of commands. The simplest way for user-written code to invoke regular commands to run under an OST or NNT is to simulate commands entered from a terminal. Include the command in the text of a standard buffer with an initialized buffer header, as described in **BUFHDR: Buffer Header** on page 109. In BUFHDR, set HDRMTYPE to HDRTYPET. You can use HDRTYPEB as well, but commands with a HDRMTYPE set to HDRTYPEB are not logged or echoed.

Use macro DSIMQS to queue the buffer to the desired OST or NNT, where the command is processed as though it had been entered from a terminal.
Choosing Command Priority: You can schedule commands as high priority. This enables the command to preempt any existing queued messages or other work that is scheduled at lower priority. Commands scheduled with high priority also preempt command procedures that are already executing. If you do not want to preempt work that may already be queued, including command procedures that are already executing, schedule your command at low priority. See Message Queuing Services on page 208 for more information about priority.

Passing Commands to a Subtask in the Same Domain: To pass commands to a subtask in the same domain, build an IFRCODCR and queue it to the desired subtask. An IFRCODCR (see DSIIFR: Internal Function Request on page 125) is an internal function request (IFR) to invoke a command. IFRCODCR is intended for requesting or conveying data. The command driven by this type of buffer does not present data to the operator by full-screen or line mode, and it does not create or remove a long-running command. These actions can be disruptive because the operator can be engaged in unrelated activity.

The IFR requires:
- An initialized BUFHDR with command extension
- An IFRCODE set to IFRCODCR

The command and its operands follow the IFRCODE.

In BUFHDR, HDRMLENG must reflect the length of the command and its parameters, and the length of the IFRCODE field. Set HDRTDISP to the offset of the IFRCODE field, which must be X'24', the offset of the HDRMSG field in the BUFHDR. Set HDRMTYPE to HDRTYPEI.

Use macro DSIMQS to send the buffer to any subtask that can process a command. When the buffer is received, the NetView program increases HDRTDISP by 2 to address the command and its parameters. The NetView program decreases HDRMLENG by 2, because the IFRCODE is not included in the command text.

For more details about the fields and settings of BUFHDR and IFR, see BUFHDR: Buffer Header on page 109 and DSIIFR: Internal Function Request on page 123.

See sample CNMS4283 in Appendix A. Assembler Samples on page 283 for an example of scheduling a command.

Scheduling Commands in a Cross-Domain Environment

You can schedule commands on a remote system using the RMTCMD command. You can enhance the power of this cross-domain scheduling by using NetView pipelines with the RMTCMD command.

The DSIPSS macro also provides for the scheduling of commands in a cross-domain (OST or NNT) environment.

You can forward a command from one domain to another by doing either of the following (providing the route has been previously established using the START DOMAIN command):
- Building a buffer with the desired command and issuing macro DSIPSS with TYPE=XSEND to transmit the command to the cross-domain task (NNT) in another NetView program.
Designing Assembler Modules

- Calling the ROUTE command. (See "Scheduling Commands Using DSIMOS" on page 18) The ROUTE command routes a command to a specified NetView domain. For more information, refer to the NetView online help.

The commands you invoke can return data to the originating domain by issuing DSIPSS TYPE=OUTPUT for a buffer with HDRMTYPE=HDRTYPEU or HDRMTYPE=HDRTYPEL.

The maximum length of text that you can send as a cross-domain command is 240 bytes, which corresponds to three 80-character input lines. Use multiple commands to chain data for applications that require a larger data transfer.

Additional Services

The following sections describe other services provided by the NetView program.

Loading and Deleting Modules in Virtual Storage

You can dynamically load and delete modules that are used infrequently by using the DSILOD and DSIDEL macros. Use DSILOD to load the module and DSIDEL to delete the module.

See sample CNMS4271 in "Appendix A. Assembler Samples" on page 283 for an example of using DSILOD and DSIDEL.

Posting and Waiting on Event Control Block (ECB) Services

You can post and wait on ECBs using the DSIPOS (post) and DSIWAT (wait) macros. DSIWAT enables you to wait on a single ECB or on a list of ECBs.

See the optional task template (CNMS4277) in "Appendix A. Assembler Samples" on page 283 for an example of waiting on an ECB list.

Disk Services

The disk services macro retrieves data from partitioned data sets. The macro DSIDKS is linked to a data set. This macro locates a specific member or file and reads the records there, as illustrated in the following example:

```
DSIDKS SWB=(REG2),DSBWORD=DISKADDR,TYPE=CONN,NAME=DSIPRF 1

DSIDKS SWB=(REG2),DSBWORD=DISKADDR,TYPE=FIND,NAME=MEMNAME 2

DSIDKS SWB=(REG2),DSBWORD=DISKADDR,TYPE=READ 3

DSIDKS SWB=(REG2),DSBWORD=DISKADDR,TYPE=READ,INCL=YES 3

DSIDKS SWB=(REG2),DSBWORD=DISKADDR,TYPE=DISC,NAME=DSIPRF 4

DSIPRF DC CL8' DSIPRF '

MEMNAME DC CL8' MEMNAME '
```

Key Explanation
This statement initializes the disk service control blocks and input buffer, returning the address of the DSB in DISKADDR. The NAME parameter specifies the DDNAME (DSIPRF in this example).

This statement finds the member name MEMNAME and reads the first record using the returned DSBWORD. Because FIND reads the first record of the file, you can code INCL=YES on a FIND request to indicate that if the first record is a %INCLUDE card, it is to be processed.

These two statements read the next two sequential records. INCL=YES specifies that %INCLUDE cards are to be processed.

This statement frees the control blocks and the input buffer.

See sample CNMS4276 in “Appendix A. Assembler Samples” on page 283 for an example of using DSIDKS to read a NetView initialization file.

Parsing

You can parse NetView command and message buffers (containing the standard NetView BUHDR structure) using the DSIPRS macro. The DSIPRS macro requires a parse descriptor block (PDB). The returned PDB includes the PDDBUF pointer to the command buffer, the parse elements, and the number of parse elements.

To determine the size of the PDB and parse a buffer you need to:

1. Issue the DSIPRS macro with the PDBSIZE option specified. This returns the required size of the block.
2. Build the control block header (CBH) and set CBHID to the value defined by symbol CBHPDB after you obtain the storage (from preallocated storage or with DSIGET).
3. Set CBHTYPE to zero (0).
4. Store the PDB length in the CBHLENG field.
5. Invoke DSIPRS with the supplied PDB to parse the buffer. DSIPRS fills in the PDB with the delimiter and parse information.

Note: The DSIPRS macros are issued in pairs.

See sample CNMS4280 in “Appendix A. Assembler Samples” on page 283 for an example of using DSIPRS to calculate the size and initialize a parse descriptor block.

Command Authorization Checking

Command authorization checking verifies that a command or operand is available to an operator or group of operators.

You can use the command entry services (DSICES) macro to determine whether an operator is authorized to use a given command, and you can use the keyword and value service (DSIKVS) macro to determine whether an operator is authorized to use a given keyword or keyword and value combination.

You can use the parameter alias service (DSIPAS) macro to derive the original keyword and value for a command that is entered with parameter aliases. Parameter aliases are defined with the PARMSYN statement in DSIPARM member DSICMD.
Named Storage

You can create and retrieve a storage environment across multiple command processor calls using named storage.

You can use the DSIPUSH macro to create a named storage pointer, as shown in the following code segment:

```
L R3,LOCLSTOR BUFFER AREA FOR PUSH LIST
USING SWBLRCPL,R3
XC 0(SWBLRCPU,R3),0(R3)
MVC SWBLRCLN(4),=A(SWBLRCPU) LENGTH OF PUSH LIST
MVC SWBLRCNM(16),MYNAME UNIQUE NAME OF LRC
MVC SWBLRCST(4),DYNASTOR QUEUED STORAGE OBTAINED FOR LRC
MVC SWBLRCRE(8),ZEROS NO RESUME FOR NAMED STORAGE
MVC SWBLRCAB(8),MYABEND ADD ABEND MODULE NAME
MVC SWBLRCGL(8),=C’DSILRCR8’ (FOR EXAMPLE)
* SWBLRCFG (FLAGBITS) IGNORED FOR NAMED STORAGE
L R4,CWBSWB AVAILABLE SWB
DSIPUSH SWB=(R4),LIST=(R3),ROLL=NO
```

Then you can use the DSIFIND macro to retrieve the named storage pointer, as shown in the following code segment:

```
L R3,LOCLSTOR BUFFER AREA FOR PUSH LIST
USING SWBLRCPL,R3
XC 0(SWBLRCFI,R3),0(R3)
MVC SWBLRCLN(4),=A(SWBLRCFI) LENGTH OF FIND LIST
MVC SWBLRCNM(16),MYNAME UNIQUE NAME OF LRC
L R4,CWBSWB AVAILABLE SWB
DSIFIND SWB=(R4),LIST=(R3)
LR R3,R1 ADDRESS OF MY NAMED STORAGE
USING MYDSECT,R3
```

For more details on the coding of these macros, see "DSIPUSH: Establish Long-Running Command" on page 229 and "DSIFIND: Find Long-Running Command Storage" on page 175.

Sending and Receiving Data Using the MS Transport

The following macros send and receive data using the MS transport:

**DSIGETDS**

Retrieves messages or management services units (MSUs) on the initial data queue when the assembler command processor is driven from the automation table.

**DSI6REGS**

Registers an MSU application with the MS transport, or a served application with operations management.

**DSI6SNDs**

Enables MS applications or operations management served applications on
the NetView program to send data to a specified target in the same domain, or to a logical unit (LU) or VTAM CP name in any domain. Use of the VTAM CP name applies to a Version 5 Release 1 NetView program running under VTAM Version 4 Release 1 or later.

**Sending and Receiving Data Using the High-Performance Transport**

The following macros send and receive data using the high-performance transport:

- **DSIHREGS**
  Registers an application that wants to send data to or receive data from another application using the high-performance transport. This macro also deregisters applications.

- **DSIHSNDS**
  Enables NetView applications to send data to a specified target using the high-performance transport. These applications must be registered through DSIHREGS.

**Passing an MSU through the NetView Automation Table**

DSIAUTO macro provides a way for a program, executing within the NetView address space in MVS to pass an MSU directly to the active automation table. For details about DSIAUTO, see "[DSIAUTO: Invoke Automation Services](#)" on page 161.

**Numeric Code Point Translation**

You can use the DSIC2T macro for numeric code point translation. The code point translation service routine is available to the NetView program in REXX, PL/I, C, and assembler languages. The function performed is the same, regardless of the language you choose.

The NetView code point translation service is provided to enable you to translate the numeric code points received in alerts into readable text. This in turn enables you to use NetView Bridge with a problem-management database to open problem records when NetView alerts are received.

**Specifying Tokens in Alert Automation**

The DSIBAM macro specifies the `keyword=data` tokens included in the alert automation message generated by CNMS4284.

**Note:** Issue this macro ONLY from CNMS4284.

**Returning a Command to an Originating Domain**

For a command running under an NNT to invoke a command in the originating domain, the command must issue DSIPSS TYPE=OUTPUT for a buffer with HDRM.TYPE=HDRTYPEX. The buffer must contain the desired command verb and parameters. The verb must be delimited from the data or parameters with a blank and must be a defined command in the receiving domain.

For example, if data formatting is required, you can build a buffer with HDRM.TYPE=HDRTYPEX and a command in the buffer text. In this case, the command verb identifies a user-defined presentation services command processor and the parameters are the data to be presented. When the receiver of the OST’s cross-domain message gets the command buffer, the OST calls the command processor with the data.
Designing Assembler Modules

Applications are limited to sending 256 bytes of data. Use multiple commands to chain data for applications that require a larger data transfer.

Resource Span Checking

You might want a command processor to use NetView span checking facilities to limit operators to particular sets of resources. This section describes the macros and logic required to implement resource span checking using DSIQRS.

The DSIQRS macro simplifies resource span checking. DSIQRS returns span information for all operators that are currently defined, including dynamically added operators. Therefore, you should use the DSIQRS macro for all resource span checking.

Note: For more information about span checking, refer to the Tivoli NetView for z/OS Security Reference.

Resource Span Checking Using DSIQRS

The DSIQRS macro simplifies the implementation of resource span checking. Use DSIQRS to determine whether a resource is in an operator’s span of control and whether an operator is allowed access to a particular resource. For example, if you want to determine whether a particular operator is authorized to issue commands for a particular resource, DSIQRS might be specified as follows:

DSIQRS SWB=(REG2),OPID=OPERID,RESOURCE=RESNAME

Applications supply the operator ID and the VTAM resource name to the DSIQRS macro. DSIQRS issues one of the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The NetView operator has access to the resource</td>
</tr>
<tr>
<td>64</td>
<td>The NetView operator does not have access to the resource</td>
</tr>
<tr>
<td>101</td>
<td>RODM query failure</td>
</tr>
</tbody>
</table>

Data Services Task (DST) Unique Services

Access to communication network management (CNM) data and VSAM files is provided by the CNM service (DSIZCSMS) and VSAM service (DSIZVSMS) macros. You can use these macro services only under DSTs. DSTs and their macro services are described in “Chapter 8. Macros” on page 161.
Chapter 3. Writing Installation Exit Routines

This chapter illustrates how to design, code, and install installation exit routines that take advantage of NetView exits.

Overview of Installation Exit Routines

You can write installation exit routines to view, delete, or replace data flowing to, from, or through NetView. You can also write installation exit routines to perform special functions. NetView enables you to code installation exits for your specific processing needs. Some examples of specific processing needs include accounting, routine processing, security verification, and any data manipulation requirements. Installation exit routines handle a specific event with nonstandard processing and automate processes based on message and management services unit (MSU) information. For example, your code can examine the messages and MSUs passing through the NetView program, record relevant data, and initiate work requests based on the data.

NetView provides two types of installation exits for which you can write routines:

- Global installation exits (DSIEXnn), which apply to all NetView tasks. The global installation exit routines are loaded when NetView starts.
- Data services task (DST) installation exits (XITnn and BNJPALEX), which apply only to DSTs. The DST installation exit routines are loaded when the associated DST starts. Each DST can have its own set of installation exit routines.

Each installation exit handles a particular event, such as the reception of data from the system console. When that event occurs, NetView passes control to the appropriate installation exit routine for processing. After processing, the installation exit routine returns control and passes a return code to NetView. Optionally, you can concatenate up to ten DST exit routines. If you concatenate routines, the first DST exit routine returns control to NetView. If the first exit does not indicate USERDROP, NetView calls the next exit in the sequence. This process continues until the last DST exit has returned control to NetView or USERDROP is indicated.

Writing a routine for each installation exit is not necessary. Installation exits written to process frequently executed functions degrade performance. See Table 5 for a list of installation exits and their sample numbers. These samples are meant to be used as examples of how to code a NetView installation exit. You can customize them for your individual needs.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Description</th>
<th>Applicable Tasks</th>
<th>TVBINXIT</th>
<th>REENT</th>
<th>NetView Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIEX01</td>
<td>Input from the operator</td>
<td>OST with VTAM terminal</td>
<td>On</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX02A</td>
<td>Message output this domain or message output cross-domain</td>
<td>NNT, OST, PPT NNT, OST, CNMCCSSIR</td>
<td>On or Off</td>
<td>Yes</td>
<td>CNMS4271</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CNMS4283</td>
</tr>
<tr>
<td>DSIEX03</td>
<td>Input before command processing</td>
<td>NNT, OST, PPT</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX04</td>
<td>Log output for buffers not processed by DSIEX02A</td>
<td>Main task or any subtask</td>
<td>On or Off</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
### Table 5. Installation Exit Environments (continued)

Superscript numbers refer to the corresponding notes at the end of the table.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Description</th>
<th>Applicable Tasks</th>
<th>TVBINXIT</th>
<th>REENT</th>
<th>NetView Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIEX05</td>
<td>Before VTAM command invocation</td>
<td>NNT, OST, PPT</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX06</td>
<td>Solicited VTAM messages</td>
<td>NNT, OST, PPT</td>
<td>Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIEX07</td>
<td>Cross-domain command send</td>
<td>NNT, OST</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX09</td>
<td>Output to the system console</td>
<td>Main task or any subtask</td>
<td>On or Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX10</td>
<td>Input from the system console</td>
<td>DSIWTOMT</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>DSIEX11</td>
<td>Unsolicited VTAM messages</td>
<td>PPT</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>DSIEX12</td>
<td>Logon validation</td>
<td>NNT, OST</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX13</td>
<td>OST/NNT message receiver</td>
<td>NNT, OST, PPT</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX14</td>
<td>Before logoff</td>
<td>NNT, OST</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX16</td>
<td>Post-NetView automation table exit for messages</td>
<td>NNT, OST, PPT</td>
<td>On or Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX16B</td>
<td>Post-NetView automation table exit for MSUs</td>
<td>DST, OST, PPT</td>
<td>On or Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DSIEX17</td>
<td>Monitors MVS system messages and delete operator messages (DOMs)</td>
<td>NNT, OST, PPT</td>
<td>Off</td>
<td>Yes</td>
<td>CNMS4297</td>
</tr>
<tr>
<td>DSIEX18</td>
<td>Network LOG BROWSE installation exit</td>
<td>LOG BROWSE TASK</td>
<td>On</td>
<td>No</td>
<td>CNMS4298</td>
</tr>
<tr>
<td>DSIEX19</td>
<td>Service point application authorization checking during RUNCMD processing</td>
<td>NNT, OST, PPT</td>
<td>Off</td>
<td>Yes</td>
<td>CNMS4307</td>
</tr>
<tr>
<td>DSIEX20</td>
<td>SAW record filtering</td>
<td>DST (AAUTSKLP)</td>
<td>Off</td>
<td>Yes</td>
<td>CNMS4308</td>
</tr>
<tr>
<td>DSIEX21</td>
<td>DSITCPRF encryption</td>
<td>CNMTAMEL, TCP/IP OST</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>BNJPALEX</td>
<td>Screen 4700 loop alerts</td>
<td>DST (BNJDSE36)</td>
<td>Off</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XITBN</td>
<td>BSAM empty file</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>XITBO</td>
<td>BSAM output</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>XITCI2</td>
<td>CNM data input</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td>CNMS4284 CNMS0057</td>
</tr>
<tr>
<td>XITCQ2</td>
<td>CNM interface output</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>XITDG2</td>
<td>DST initialization</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>XITVI</td>
<td>VSAM input</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>XITVN</td>
<td>VSAM empty file</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td>CNMS4270</td>
</tr>
<tr>
<td>XITVO</td>
<td>VSAM output</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>XITXL</td>
<td>External logging</td>
<td>DST</td>
<td>Off</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
Designing and Coding an Installation Exit Routine

Installation exit routines must adhere to the guidelines for user-written programming described in "General Coding Guidelines" on page 6. In addition, installation exit routines needs to conform to the special requirements described in this section. After coding your installation exit routine, follow the instructions in "Installing an Installation Exit Routine" on page 60.

Input

When you enter the installation exit routine, the registers contain the following information:

<table>
<thead>
<tr>
<th>Register Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  The address of the installation exit parameter list (USE control block). The USE contains the following:</td>
</tr>
<tr>
<td>• The address of a service work block (SWB) to be used as a work area or to request services from the NetView program (USERSWB). If you use the SWB for your save area or for dynamic variables, you must obtain another SWB when invoking NetView macros.</td>
</tr>
<tr>
<td>• The address of the message buffer USERMSG.</td>
</tr>
<tr>
<td>• The address of the task vector block (TVB).</td>
</tr>
<tr>
<td>13 The address of a standard 72-byte save area used to store the caller’s registers.</td>
</tr>
<tr>
<td>14 The return address.</td>
</tr>
<tr>
<td>15 The entry address of the installation exit.</td>
</tr>
<tr>
<td>0, 2-12 Unspecified.</td>
</tr>
</tbody>
</table>

Do not change any input, particularly the USERMSG buffer in the USE control block.

In general, the installation exits process data as it is received. That is, no specific translation to upper case is performed.

When data is received from the keyboard, NetView calls DSIEX01. At entry to DSIEX01, no translation to upper case has been performed on the message buffer. However, when DSIEX01 returns control to mainline code, the message buffer is translated to upper case. Therefore, all subsequent installation exits that process this message buffer process a buffer that is in upper case.
When the data received from the keyboard contains the characters NETVASIS as the beginning of the text, DSIEX01 receives an untranslated message buffer as above. However, when DSIEX01 returns control to mainline code, the message buffer is not translated to upper case. In this case, all subsequent installation exits that process this message buffer receive the buffer in its untranslated form.

Output

Most installation exit routines pass the following return codes to NetView in register 15 to indicate that the messages or MSUs are to be unchanged, deleted, or replaced. Exceptions to the following return codes are noted with the specific exits. The registers are restored without change, except for register 15 (and register zero (0) if USERSWAP is returned).

**USERASIS (0)**
Use the message or MSU as presented to the installation exit; do not delete or replace it.

**USERDROP (4)**
Delete the message or MSU from the terminal and from the network log, system log, and hardcopy log; stop processing before the message or MSU appears on the screen.

**USERSWAP (8)**
Replace the message or MSU with the message or MSU addressed in register zero (0).

See specific installation exit descriptions in "Summary of Installation Exits" on page 31 for additional return code considerations.

Deleting Messages and MSUs

To delete a message or MSU entirely, use return code USERDROP. NetView frees the buffers using DSIFRE. Therefore, the installation exit should not free the buffers.

When NetView receives a USERDROP return code, no further exit routines are called. If you have concatenated DST exit routines, a USERDROP return code prevents the next exit routine from being called.

When processing a single line of a title-line message (HDRTYPEJ, HDRTYPEK, or HDRTYPEL), do not delete a CONTROL, LABEL, or END line unless the entire message is deleted. When processing a title-line message formatted as an IFRCODAI, you can delete any line. For example, if DSIEX06 deletes message IST314I END, processing of the entire title-line message (of which this is the END line) is disrupted. However, if DSIEX02A deletes IST314I END, the remainder of the title-line message is displayed normally.

Replacing Messages and MSUs

If you want to replace a message or MSU, the new message or MSU must be one of the following:
- A static message
- An MSU
- A buffer in a reentrant storage area, such as the SWBADATD or SWBPLIST areas of the USE control block’s USERSWB.

Only the text portion of the buffer is swapped. Also, ensure that the HDRMLENG of the new message or MSU is less than or equal to HDRMLENG of the original message or MSU, unless the message or MSU is formatted as an IFRCODAI.
If you want to replace a title-line message, do not change the HDRMTYPE or HDRIND fields in the buffer header. For more information about title-line messages, see “Title-Line Output” on page 13.

When processing a single line of a title-line message (HDRTYPEJ, HDRTYPEK, or HDRTYPEL), do not replace any lines or the sequence and format might be lost. When processing a title-line message formatted as an IFRCODAI, you can replace any line.

Installation exit DSIEX02A provides a more flexible interface for replacing messages including title-line and MLWTO (multiline write-to-operator) type messages. See “DSIEX02A: Output to the Operator” on page 33.

You can concatenate DST installation exit routines when replacing messages. In this case, the buffer containing the replacement message or MSU becomes the input for the subsequent DST installation exit routine.

**Considerations for IFRCODAI**

If you want a message or MSU logged but not displayed, you can set the appropriate display and logging flags in the IFRCODAI (see “DSIIFR: Internal Function Request” on page 125 in the appropriate installation exit (for example, DSIEX16 and DSIEX16B).

Several NetView installation exits are invoked to process automation internal function requests (function requests with IFR code IFRCODAI). IFRCODAI buffers chain other buffers off them. If your installation exit replaces any of these buffers, use separate DSIGETs for each buffer. For each DSIGET, specify nonqueued subpool zero storage.

**Note:** Subdivision of storage into multiple buffers results in storage management problems such as lost storage or an abend.

**Message Flow and Interception Points**

The following topics summarize the sequence of decisions made by NetView in handling messages. This information can be useful in determining what forms of message processing are most efficient to meet your performance objectives.

The NetView automation table cannot automate messages that are written only to the network log.

**Message Handling by OST/NNT:**

1. A message is solicited from VTAM through the program operator interface (POI). One of the following actions is taken:
   - For messages that the status monitor uses to update network status, VTAM messages are not processed by the status monitor, with the exception of IST608I and IST1274I.
   - If exit DSIEX06 (solicited access method message input) exists, the exit is invoked. Deleted messages are not processed further.

2. Exit DSIEX02A is called, if one exists. If the exit indicates that the message is to be deleted, it is deleted.

3. The message is checked to see if it satisfies an &WAIT condition in a NetView command list or a TRAP condition in a REXX or high-level language command procedure. With the SUPPRESS option, the message is marked for deletion unless DSIEX16 specifies otherwise.
Writing Installation Exit Routines

4. The NetView automation table begins processing. Table actions are reflected in the buffer structure given to DSIEX16. The automation internal function request (AIFR) buffers have fields that are set to indicate what actions the automation table has scheduled. These actions are carried out after DSIEX16 returns to NetView.

5. DSIEX16 is called.

6. Logging, display, routing, and command actions are processed as specified in the buffer in combination with the current DEFAULTS and OVERRIDE settings.

7. If the message is displayed and copies have been assigned by the ASSIGN command with the COPY option, a copy is sent to each assigned operator. The NetView automation table cannot automate the copied messages.

Note: The NetView automation table and DSIEX16 are called only once for each unique message in a NetView domain. Any copies of the message made by the ASSIGN command or the NetView automation table do not result in a call to the NetView automation table or DSIEX16 in this NetView domain.

Note: Refer to the Tivoli NetView for z/OS Automation Guide for more information about message handling by the OST/NNT.

Message Handling by the PPT Including Messages to the Authorized Receiver:

1. When the message is from VTAM through the POI:
   a. If this is one of the messages that the status monitor uses to update network status, it is processed by the status monitor.
   b. If there is an unsolicited message installation exit (DSIEX11) or an exit for a message solicited by a VTAM command issued under the primary POI task (DSIEX06), the proper exit is invoked. Deleted messages are not processed further.
   c. VTAM commands and messages received under the VTAM start option PPOLOG=YES (such as those entered from the system console and echoed to NetView) are only logged. They are not automated.

2. When the message has been assigned using the ASSIGN command with the PRI and SEC options, each assigned operator is sent a copy of the message (if an operator was logged on in the PRI list). These messages then proceed through the OST/NNT flow for those particular operators, but the secondary (SEC) copies are not processed by the NetView automation table or DSIEX16.

3. When the message has not been assigned, it is sent to the authorized message receiver and proceeds through the OST/NNT flow for that operator.

4. When the message has not been assigned and no authorized message receiver is logged on to NetView, the flow continues as follows:
   a. Exit DSIEX02A is called, if one exists. If the exit specifies that the message is to be deleted, it is deleted.
   b. The NetView automation table begins processing. Table actions are reflected in the buffer structure given to DSIEX16. The AIFR buffers have fields that are set to indicate what actions the automation table has scheduled. These actions are carried out after DSIEX16 returns to NetView.
   c. DSIEX16 is called. The NetView automation table and DSIEX16 are called only once for each unique message in a NetView domain. Any copies of the message made by the ASSIGN command or the NetView automation table do not result in a call to the NetView automation table or DSIEX16 in this NetView domain.
   d. Logging, routing, and command actions are processed as specified in the buffer in combination with the current DEFAULTS and OVERRIDE settings.
Writing Installation Exit Routines

e. If the message is to be displayed, it is written to the system console.

Note: Refer to the Tivoli NetView for z/OS Administration Reference for information about determining the authorized message receiver. Refer to the Tivoli NetView for z/OS Automation Guide for more information about message handling by the PPT.

Control Blocks

The service facilities used in your installation exit routine dictate the control blocks you must include in your routine. However, the following control blocks are required:

- The installation exit parameter list (USE)
- The main vector table (MVT)
- The service work block (SWB)

Use macro DSICBS to include these and any additional control blocks your routine needs. For more information, see "DSICBS: Control Block Services" on page 165 and the control block descriptions in "Chapter 7. Control Blocks" on page 104.

Summary of Installation Exits

The following sections describe each installation exit provided by NetView, including examples of use and coding requirements.

BNJPALEX: Alert Generation Exit Routine

Description: BNJPALEX is an exit for the 4700 Support Facility. When BNJPALEX is given control, register 1 points to a single element parameter list that points to the data structure shown in Table 6.

Table 6. BNJPALEX Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL4</td>
<td>Address of 1st extended statistical counter</td>
</tr>
<tr>
<td>AL4</td>
<td>Address of an 8-byte user text area</td>
</tr>
<tr>
<td>CL8</td>
<td>Controller name</td>
</tr>
<tr>
<td>XL8</td>
<td>Extended statistical counter threshold exceeded bit map (nth bit on indicates that the nth extended statistical counter exceeded the threshold)</td>
</tr>
<tr>
<td>XL2</td>
<td>Loop basic counter-2 threshold</td>
</tr>
<tr>
<td>XL2</td>
<td>Extended statistical counter threshold</td>
</tr>
<tr>
<td>XL1</td>
<td>Loop basic counter-2 counter value</td>
</tr>
<tr>
<td>XL1</td>
<td>Count of extended counters reported (maximum 64)</td>
</tr>
<tr>
<td>CL1</td>
<td>Alert type byte</td>
</tr>
<tr>
<td></td>
<td>X'01' Extended statistical counters exceed threshold</td>
</tr>
<tr>
<td></td>
<td>X'02' Loop basic counter exceeds threshold</td>
</tr>
<tr>
<td></td>
<td>X'03' Both counter types exceed thresholds</td>
</tr>
</tbody>
</table>

When BNJPALEX returns control to the 4700 Support Facility, register 15 return codes indicate what can be done with the alert. The following are the only acceptable values for register 15:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Issue the alert.</td>
</tr>
<tr>
<td>4</td>
<td>Issue the alert and include the eight bytes of user text located in the user text area.</td>
</tr>
</tbody>
</table>
Writing Installation Exit Routines

Do not issue the alert.

Coding Considerations: Each time the 4700 Support Facility receives data from the 4700 network, it analyzes that data with respect to user-defined error thresholds. Whenever a threshold is exceeded, the 4700 Support Facility issues an alert message to the NetView authorized receiver. The 4700 Support Facility program exit provides for screening the loop error alerts (either loop basic counter 2 or extended statistical counter alerts). The optional user-written exit routine can add up to eight bytes of user text in the alert message, or it can completely suppress the alert.

If the exit is not included in the system, the exit must be linked into a load library specified on the STEPLIB statement of the NetView startup procedure. The name of the load module and its entry point are BNJPALEX. You need to write the exit as a reentrant control section (CSECT).

The BNJPALEX exit routine runs under the BNJDSE36 DST. If BNJPALEX is not coded, a message is displayed when the BNJDSE36 task is started indicating that the LOAD has failed. Ignore the message if you did not install this exit.

Note: For an example of an exit routine, refer to the IBM 3600/4700 library.

DSIEX01: Input from the Operator

Description: NetView calls DSIEX01 when an operator provides standard-mode input to an OST or when an NNT receives cross-domain input. DSIEX01 runs after device-dependent data has been removed from the input buffer but before syntax or command verbs are analyzed and before the message is logged. All commands issued under an operator OST, regardless of type, that are issued from the command facility command line are passed to DSIEX01. The exception to this is when certain error conditions occur such as a failure to parse the command. For an NNT, DSIEX01 is called when a command is routed from the owning task.

The exit is not called if any of the following occur:
• Commands are issued under an autotask or a RMTCMD distributed autotask.
• The Enter key is pressed with no data.
• The Clear key is pressed.

If a PF or PA key is pressed, the key definition is resolved first, then DSIEX01 is called.

All regular (TYPE=R) commands, including those from the command facility, the hardware monitor, or the threshold analysis and remote access feature, are also passed to DSIEX03.

Example of Use: You can use DSIEX01 to support short cut commands by translating an operator’s single character entries into extended command strings.

Coding Considerations: DSIEX01 follows the coding guidelines given in “Limitations When TVBINXIT Is Active” on page 7. DSIEX01 does not apply to unattended operator tasks and MVS console operator tasks.

Return Codes: Return codes for OST input are:
USERASIS (0)
The original command is processed and added to the RETRIEVE stack.
The original command is not processed and not added to the RETRIEVE stack.

The command buffer returned by the exit pointed to by register zero (0) is processed. The original command is added to the RETRIEVE stack.

**DSIEX02A: Output to the Operator**

**Description:** DSIEX02A is called under the OST, NNT, PPT, or CNMCSIR task. The NetView program calls DSIEX02A for standard output to an operator’s terminal (DSIPSS TYPE=OUTPUT, DSIPSS TYPE=IMMED, or FLASH) before &WAIT and WAIT message processing, and before the NetView automation table is scanned to determine processing actions. For example, messages from user-written command processors that issue DSIPSS TYPE(OUTPUT) result in calls to DSIEX02A. DSIEX02A processes a single message or multiline message at a time.

NetView automation table processing occurs before DSIEX16 is called, and the resultant actions are scheduled immediately after this exit.

The single message that was processed by DSIEX02A can become a chain of messages representing the actions that the NetView automation table produced.

**Example of Use:** DSIEX02A provides you with an exit that can process the messages before &WAIT (or WAIT) and the NetView automation table take effect.

You can use this exit to modify the processing options for a message and specify or substitute new logging, display, command, or routing actions independently of one another.

DSIEX02A can reformat messages by removing buffers, changing buffers, and adding new buffers to the original message. The exit can prevent OVERRIDE command options from taking effect for messages.

**Note:** If your messages are to be translated to Japanese, for example, changes to the message text can affect the translations. Refer to the [Tivoli NetView for z/OS Installation: Configuring Additional Components](#) for more information.

**Coding Considerations:** TVBINXIT is on when called for DSIPSS TYPE=IMMED. If TVBINXIT is on, DSIEX02A follows the coding guidelines given under “Limitations When TVBINXIT Is Active” on page 7.

Do not use macro DSIPSS in this installation exit routine. If a message is required, use DSIMQS to queue the message to the subtask.

Because NetView does not check the syntax of messages that are sent to a terminal, DSIEX02A does not receive a parse descriptor block (PDB).

NetView automation is invoked after this exit routine is called; therefore, any changes made for messages in this installation exit can affect NetView automation. NetView automation is not invoked for a message that is deleted by this exit routine.

DSIEX02A provides the following additional features not available in other exits:

- The NetView buffer passed to this exit is an AIFR. You need to reference the IFR control block rather than the BUFHDR.
Writing Installation Exit Routines

- For single-line messages, multiline messages, and title-line messages, this buffer points to the entire chain of buffers that comprise the message.

You can replace all chained buffers by using DSIGET for nonqueued subpool zero storage for new buffers and copying or replacing all the data in the old buffers. Separate DSIGET invocations should be used for each buffer chained off the IFRCODAI internal function request buffer. Any original buffers passed to the exit should be either freed or passed back to NetView on the return. You must free the unused buffers using DSIFRE for nonqueued subpool zero storage. Initialize all necessary fields in all buffers and copy any reserved or unused header information from each of the buffers.

Notes:

1. NetView frees only those buffers that are returned in the IFRCODAI format.
2. For information about freeing NetView buffers see "Free NetView Buffers Service Routine (DSIFREBF)" on page 277.
3. Messages issued by DSIPSS with TYPE=FLASH cannot be processed by user exit DSIEX02A because the exit is not called for FLASH messages.

The IFRCODAI contains control information and system data that can be accessed only through the provided NetView automation table and command list interfaces. Unauthorized modification of these fields can cause processing, logging, or display loops.

Return Codes: In DSIEX02A, you can swap or delete messages by direct manipulation of the buffers that are located by the USERMSG field of the DSIUSE parameter list. When the return code is USERASIS, NetView uses the AIFR that is returned in the USERMSG field.

Use the DSIGET macro to obtain new buffers and the DSIFRE macro to release buffers you have removed. To release all the buffers and the AIFR buffer, use the DSIFREBF callable service and set USERMSG to zero.

The return codes are as follows:

USERASIS (0)
NetView continues processing with the AIFR found in the USERMSG field. If you return a zero in the USERMSG field, you are responsible for freeing the AIFR and all associated buffers.

USERDROP (4)
NetView frees the AIFR in the USERMSG field.

USERSWAP (8)
NetView continues processing on the AIFR found in register zero (0). This is similar to USERASIS, except that you return the AIFR in register zero (0) instead of USERMSG.

Note: AIFR that is originally passed to the exit in the USERMSG field becomes the responsibility of the exit. If the same AIFR is not passed back to NetView in register zero (0), the exit is responsible for freeing the original AIFR.

General Notes for DSIEX02A: When IFRCODAI buffers have commands, the DISPLAY, logging, BEEP, and HOLD options are not referenced during the command execution.
DSIEX02A has access to user bit and character fields in the IFRCODAI structure that enable signaling between DSIEX02A, DSIEX03, DSIEX16, user-written tasks, or user-written commands. Actions (including HOLD and BEEP) specified in the IFRCODAI structure supersede the DSIMVT defaults. DSIEX02A cannot determine the overrides but can mark buffers that are not affected by the overrides. When DEFAULT BEEP=DISABLE or HOLD=DISABLE, the option is not processed by NetView when specified by DSIEX02A, DSIEX16, or the NetView automation table, unless the IFRCODAI buffer has the corresponding force flag on in IFRAUTA4 and the action flags in IFRAUTA1 or IFRAUTA2 contain a nondefault value.

If a force flag is set on in IFRAUTA4 before NetView automation table processing (such as in DSIEX02A or before), the corresponding action is not considered by the NetView automation table processing even if the NetView automation table specifies a value for it. This enables DSIEX02A or NetView to define buffers that represent a subset of the logging or display actions without accidental reversals within the table processing.

Because the force action bits specify display and logging actions, their presence implies that no command action processing occurs when the NetView automation table is searched. On the other hand, buffers that contain commands (IFRAUCMD=ON) cause NetView to ignore the display and logging action indicators, including the action force flags.

Because logging actions are serviced under the task under which the NetView automation table is invoked, buffers marked as forced nondisplay are not routed to other tasks by a matching NetView automation table entry.

**DSIEX03: Input Before Command Processing**

**Description:** DSIEX03 is called for all regular commands. Regular commands include commands:

- Issued by a command procedure
- Received from another subtask
- Used to start the hardcopy log at logon
- Used as the initial command
- Resulting from the NetView automation table
- Entered for an MVS console operator task
- Entered from a terminal
- Received as HDRTYPEX messages from an NNT
- Queued with EXCMD
- Executed by a PIPE stage

Before running, all regular commands are passed to DSIEX03. Regular commands (TYPE=R) entered from the command facility command line are passed to both DSIEX01 and DSIEX03. DSIEX03 is also called for TYPE=B commands that are not entered from the command facility command line. Note that a TYPE=B command can requeue itself and therefore appear to drive both DSIEX01 and DSIEX03. Commands running under the PPT that are entered as responses to messages DS1802A and DS1803A are not passed to any installation-defined exit.

**Example of Use:** You can use DSIEX03 to restrict use of particular regular commands if your conditions are more complex than those provided for by command authorization checking.

**Coding Considerations:** NetView OST task processing, including autotasks, removes any suppression characters before invoking DSIEX03. If the command
buffer presented to DSIEX03 has a suppression character removed, the
HDRMTYPE of the command buffer is HDRTYPEB.

If two suppression characters precede the command (for example, a “quiet”
command), the HDRMTYPE is HDRTYPEQC.

If a command buffer is swapped by DSIEX03, HDRMTYPE is set to HDRTYPES
when the buffer is swapped by the exit, as described in “Replacing Messages and
MSUs” on page 28.

Return Codes: The return codes for DSIEX03 can be set to the following:

USERASIS (0)
The original command is processed.

USERDROP (4)
The original command is not processed.

USERSWAP (8)
The command buffer whose address was returned by the exit in register
zero (0) is processed.

DSIEX04: Log Output

Description: The NetView program calls DSIEX04 during the logging process.
DSIEX04 is located within log services and applies to messages logged in the
network log, the system log, and the hardcopy log. DSIEX04 runs before the
message is reformatted and sent to the log.

DSIEX04 is not called for messages logged to the external trace data set
(DSITRACE).

DSIEX04 is not called if DSIEX02A and DSIEX16 are called, because you can
specify logging options in these exits and in the NetView automation table. For
example, in a user-written command, if the macro DSIWLS is issued for a buffer
that has not been automated, DSIEX04 is called. In contrast, the same buffer sent to
the terminal with DSIPSS is also logged, but DSIEX02A, the NetView automation
table, and DSIEX16 are called.

Note: DSIEX16 is generally a better exit to reformat log buffers than exit
DSIEX02A because the NetView table options are already processed and can
be analyzed in DSIEX16. DSIEX16 enables creation of logging buffers that
are separate from the display buffers. DSIEX04 is used only for messages not
processed by DSIEX16 (messages not subject to automation).

Example of Use: You can use DSIEX04 to edit information sent to the network
log, the system log, and the hardcopy log. You can send certain messages to a
specific log or to no log at all.

Coding Considerations: If TVBINXIT is on, DSIEX04 follows the coding
guidelines given under “Limitations When TVBINXIT Is Active” on page 7.

Do not use macro DSIWCS or DSIWLS in this installation exit routine.

Because DSIEX04 can run under any subtask that issues macro DSIWLS, be sure
that any service facilities you request are supported by the subtask under which
the routine is running. For example, DST is limited to VSAM or CNM interface
services.
Because NetView does not check the syntax of messages that are sent to a log, DSIEX04 does not receive a parse descriptor block (PDB). See [DSIPDB: Parse Descriptor Block on page 147] for information about how to parse the messages in DSIEX04.

If USERPDB is not 0, a PDB is available for the installation exit. When USERPDB is 0, the installation exit must obtain its own PDB (using DSIGET) if a PDB is desired, and release it (using DSIFRE) before returning to NetView from the installation exit.

**Return Codes:** DSIEX04 can pass the following return codes:

**USERASIS** (0)  
Log the original message.

**USERDROP** (4)  
Do not log the message in any of the available logs (network, hardcopy, or system).

**USERSWAP** (8)  
Log the message whose address was returned by the exit in register zero (0) in the available logs.

**USERLOG** (12)  
Write the message to the network or system log only.

**USERLOGR** (16)  
Write the substituted message to the network or system log only. The address of the buffer containing the new message is in register zero (0).

**USERHCL** (20)  
Write the message to the hardcopy log only.

**USERHCLR** (24)  
Write the substituted message to the hardcopy log only. The address of the buffer containing the new message is in register zero (0).

**USERNSL** (28)  
Do not write to the system log.

**USERNSLR** (32)  
Do not write to the system log. Use the substituted message to write to the other logs. The address of the buffer containing the new message is in register zero (0).

**DSIEX05: Before VTAM Command Invocation**

**Description:** The NetView program calls DSIEX05 when preparing to pass a command to VTAM through the POI. Domain qualifiers have been removed and all span checking has been completed.

**Example of Use:** You can use DSIEX05 to verify that an operator is authorized to issue a particular command.

**Coding Considerations:** Code the routine to handle both the OST and PPT control block structures.

This exit applies only to commands entered directly (not using the MVS prefix) that are passed through the NetView POI.
Writing Installation Exit Routines

Note: Commands passed to DSIEX05 have already been processed under DSIEX03 and, possibly, DSIEX01.

Return Codes: DSIEX05 can pass the following return codes:

USERASIS (0)
Pass the original command to VTAM.

USERDROP (4)
Do not pass the command to VTAM.

USERSWAP (8)
Pass the command whose address was returned by the exit in register zero (0) to VTAM.

DSIEX06: Solicited VTAM Messages

Description: The NetView program calls DSIEX06 when it receives a solicited VTAM message through the POI, which is generated in response to a VTAM command the user issued or the PPT issued. The message is not processed or logged.

Example of Use: You can use DSIEX06 to change the message number or text of a VTAM message or to process VTAM messages.

Coding Considerations: Code the routine to handle both the OST and PPT control block structures.

This exit applies only to messages received through the NetView POI in response to commands entered directly (not using the MVS prefix).

NetView automation is invoked after this installation exit routine has been called. Any changes made for messages in this installation exit can affect NetView automation. NetView automation is not invoked for a message that has been deleted by this installation exit routine.

Note: Messages processed (and not deleted) by DSIEX06 are then processed by DSIEX02A.

Return Codes: DSIEX06 can pass the following return codes:

USERASIS (0)
Process the message returned by VTAM.

USERDROP (4)
Do not process the message returned by VTAM.

USERSWAP (8)
Process the message whose address was returned by the exit in register zero (0).

DSIEX07: Cross-Domain Command Send

Description: The NetView program calls DSIEX07 before commands are sent cross-domain to an NNT (DSIPSS TYPE=XSEND).

Example of Use: You can use DSIEX07 to monitor cross-domain traffic through the network.
Coding Considerations: Do not use DSIPSS TYPE=XSEND in this installation exit routine. Also, avoid directly calling commands that route a command to another domain, such as ROUTE, DISPLAY, or VARY. If necessary, you can queue such commands for execution. Refer to "Scheduling Commands Using DSIMOS" on page 18 and "Passing Commands to a Subtask in the Same Domain" on page 19 for more information.

DSIEX07 does not receive a PDB. The cross-domain NetView parses the messages after they are received. Refer to "DSIPDB: Parse Descriptor Block" on page 147 for more information about parsing the messages in DSIEX07.

Return Codes: DSIEX07 can pass the following return codes:

USERASIS (0)
Send the message to the NNT.

USERDROP (4)
Do not send the message to the NNT.

USERSWAP (8)
Send to the NNT the message whose address was returned by the exit in register zero (0).

DSIEX09: Output to the System Console

Description: The NetView program calls DSIEX09 when a message is written to the system console operator using macro DSIWCS. The message has not been formatted for transmission.

Example of Use: You can use DSIEX09 to edit messages sent to the system console.

Coding Considerations: If TVBINXIT is on, DSIEX09 follows the coding guidelines given in "Limitations When TVBINXIT Is Active" on page 7.

DSIEX09 is called as a result of DSIWCS macro calls. The output of the MVS console operator task OST is processed in DSIEX02A instead of DSIEX09.

Do not use macros DSIWCS or DSIMQS in this installation exit routine. If you need to send a message to the system console from this exit routine, use system macros instead.

Because Netview does not check the syntax of messages that are sent to the system console, DSIEX09 does not receive a PDB. Refer to "DSIPDB: Parse Descriptor Block" on page 147 for more information about parsing the messages in DSIEX09.

Return Codes: DSIEX09 can pass the following return codes:

USERASIS (0)
Send the message to the system console.

USERDROP (4)
Do not send the message to the system console.

USERSWAP (8)
Send to the system console the message whose address was returned by the exit in register zero (0).
Writing Installation Exit Routines

DSIEX10: Input from the System Console

Description: The NetView program calls DSIEX10 when input is received from the system console operator. The exit is called after the command has been entered but before it is invoked or logged.

Example of Use: You can use DSIEX10 to enable the system console operator to enter command abbreviations and synonyms. These can be expanded in the installation exit routine.

Coding Considerations: DSIEX10 is called only from task DSIWTOMT, not from a subtask.

DSIEX10 does not receive a PDB. Refer to “DSIPDB: Parse Descriptor Block” on page 143 for information about parsing the messages in DSIEX10.

DSIEX10 is not called for commands that an operator enters using an MVS console OST. DSIEX03 is called instead.

Return Codes: DSIEX10 can pass the following return codes:

USERASIS (0) 
Process the command from the system console.

USERDROP (4) 
Do not process the command from the system console.

USERSWAP (8) 
Process the command whose address was returned by the exit in register zero (0).

DSIEX11: Unsolicited VTAM Messages

Description: The NetView program calls DSIEX11 when an unsolicited VTAM message is received through the POI. In addition, when the VTAM PPOLOG=YES modify or start option is used, copies are presented to DSIEX11. This installation exit is called before the resource name is analyzed and before the message is logged.

Example of Use: DSIEX11 can issue macro DSIMQS to send a copy of the message buffer prior to processing by NetView. If you want to send unsolicited messages to all operators, DSIEX11 can transform the messages into MSG ALL commands.

Coding Considerations: If DSIEX11 calls a command or a command procedure, the command restrictions for the PPT apply.

NetView automation is invoked after this installation exit routine has been called; therefore, any changes made for messages in this installation exit can affect NetView automation. NetView automation is not invoked for a message that has been deleted by this installation exit routine.

Return Codes: DSIEX11 can pass the following return codes:

USERASIS (0) 
Process the message from VTAM.

USERDROP (4) 
Do not process the message from VTAM.
Process the message whose address was returned by the exit in register zero (0).

**DSIEX12: Logon Validation**

**Description:** The NetView program calls DSIEX12 at the completion of the logon process after accepting the logon.

**Example of Use:** You can use DSIEX12 to check authorization and environmental customization. DSIEX12 can also send messages to other operators.

**Non-Takeover and Takeover Operator Task Processing:** When DSIEX12 gets control for a non-takeover or a takeover operator task, the input to DSIEX12 will include the USERTVB (address of the session TVB) and the USETOTVB (address of the TVB being taken over) fields. If the task is a non-takeover operator task, the USETOTVB field will be zero (0). If the task is a takeover operator task, the USETOTVB field will be set to the actual address of the TVB that is being taken over.

For a takeover operator task, DSIEX12 is invoked mainly to validate the operator ID and password. Therefore, consider using the USETOTVB field in DSIEX12 to determine if this exit was invoked for a takeover operator task.

**Coding Considerations:** If you require output to the screen, use only the following DSIPSS types:
- SCRSIZE
- WINDOW
- ASYPANEL
- CANCEL
- PSSWAIT
- TESTWAIT

**Return Codes:** If the installation exit routine issues a return code of 0, the logon proceeds. If specified, your hardcopy log starts and the initial command runs.

If the issued return code is nonzero, the operator is logged off.

This installation exit is called under all OSTs and NNTs, including unattended-operator and MVS console-operator tasks.

**DSIEX13: OST/NNT/PPT Message Receiver**

**Description:** The NetView program calls DSIEX13 upon receipt of certain subtask-subtask communication buffers received on a subtask as a result of the use of the DSIMQS macro on another task. The NetView program calls DSIEX13 when:
- A user-defined IFRCODUS is received by way of macro DSIMQS for an OST, an NNT, or the PPT.
- A message buffer (with HDRMTYPE=HDRTYPEM) is received by way of macro DSIMQS on an OST or NNT.

**Note:** The DSI039I message is an example of a HDRTYPEM message received by an OST or NNT. The DSI039I message, when sent to the PPT, has a HDRMTYPE=HDRTYPEP. In addition, the PPT does not invoke DSIEX13 for any other HDRTYPEM buffer.
HDRTYPEM messages and IFRCODUS buffers are treated differently:

- NetView does not process IFRCODUS buffers after DSIEX13 is called. It frees the buffer using DSIFRE, which is consistent with other buffers received by way of DSIMQS. For information about freeing NetView buffers, see “Free NetView Buffers Service Routine (DSIFREBF)” on page 277.

- HDRTYPEM buffers are displayed unless the return code USERDROP is used. Displayed messages are processed like typical NetView messages, including calling DSIEX02A and DSIEX16 exits, invoking NetView automation table processing, and logging the messages.

**Example of Use:** You can use DSIEX13 in conjunction with IFRCODUS to initiate a user function with a buffer. Code DSIEX13 to perform the user function specified by IFRCODUS. You can send the IFRCODUS message buffer to another task or you can queue it to the task in which your program (such as a command processor) is running using the DSIMQS macro. You can also use DSIEX13 to monitor DSI039I messages received on an OST or NNT.

**Coding Considerations:** DSIEX13 does not free (using DSIFRE) the IFRCODUS or HDRTYPEM buffers. For information about freeing NetView buffers, see “Free NetView Buffers Service Routine (DSIFREBF)” on page 277.

**Return Codes:** DSIEX13 can pass the following return codes:

- **USERASIS (0)**
  - Process the message.

- **USERDROP (4)**
  - Do not process the message.

- **USERSWAP (8)**
  - Process the message whose address was returned by the exit in register zero (0).

**Note:** An incorrect return code causes a user abend code 262 (X'106').

**DSIEX14: Before Logoff**

**Description:** The NetView program calls DSIEX14 when an OST or NNT is preparing to end for any of these reasons:

- The operator enters LOGOFF.
- The subtask LOSTERM exit is driven (VTAM).
- The subtask is posted to end.

The subtask cannot communicate with the operator’s terminal at this point. However, you can issue macro DSIWCS to write to the system console and macro DSIWLS to write entries to the log.

**Example of Use:** You can use DSIEX14 to save accounting information or update tables.

**Non-Takeover and Takeover Operator Task Processing:** For a non-takeover operator task, the NetView program calls DSIEX12 after a logon is accepted. When the task ends (for example, the operator logs off), the NetView program will call DSIEX14.

For a takeover operator task, the NetView program calls DSIEX12 as mentioned above. For this type of task, DSIEX12 is called mainly to validate the operator ID and password. When the takeover processing completes, the task for the session
that is being taken over will be the task used for the session, and the takeover operator task will be cleaned up. For the takeover task processing, the NetView program will not call DSIEX14.

**Coding Considerations:** Because no buffer is associated with logoff processing, DSIEX14 receives neither an input buffer nor a PDB.

**Return Codes:** NetView ignores any return code received from this installation exit routine.

**DSIEX16: Post-NetView Automation Table Exit**

**Description:** When a message is considered for automation under the display services (DSIPSS macro), NetView immediately calls DSIEX16. DSIEX16 runs under the OST, NNT, PPT, or CNMCSSIR task.

NetView automation table processing occurs before DSIEX16 is called (DSIEX16 processing is separate and not dependent on NetView automation table processing), and the resulting actions are scheduled immediately after this exit. DSIEX16 is called just before logging, display, routing, or command actions are processed. DSIEX16 receives a description of the total processing to be performed on the message.

DSIEX16 is called for everything DSIEX02A is called for, except messages that are deleted by DSIEX02A through USERDROP or zeroing IFRAUTBA. This provides for the monitoring of message deletion.

It is possible to have more than one distinct message chained together. The message that was processed by DSIEX02A can become a chain of messages representing the actions that the NetView automation table produced. Each element of the chain is a complete buffer structure similar to the one given to DSIEX02A, and they are chained together using the HDRNEXTM field in the buffer header (BUFHDR) of each buffer on the chain. Message buffers are followed by action buffers. The last buffer has a zero in its HDRNEXTM field.

If you establish cross-domain sessions to the same domain and you want these messages automated on the receiving or OST side, you can set the IFRAUMTB bit off in this exit on the sending or NNT side. The IFRAUMTB bit is normally set on by the NNT because all messages sent from an NNT have had an opportunity to invoke the NetView automation table on the sending or NNT side. NetView changes this bit to zero when received in a new domain for all cases except when the originating domain name is equal to the receiving domain.

**Example of Use:** DSIEX16 provides an opportunity to modify the effects of &WAIT (or WAIT) and the NetView automation table after they are determined but before the automation actions take effect. If an &WAIT or WAIT suppresses a message, it does not undergo automation table processing. It is sent to DSIEX16.

Use this exit to modify the processing options for a message and specify or substitute new logging, display, command, or routing actions independently of one another.

DSIEX16 can reformat messages by removing buffers, changing buffers, and adding entirely new buffers to the original message. The exit can prevent OVERRIDE command options from taking effect for messages.
DSIEX16 provides for different editing of the text for each group of destinations specified by the results of NetView automation table or default message processing options. This exit can also help monitor the effectiveness of message suppression and automation.

**Coding Considerations:** TVBINXIT is on when called for DSIPSS TYPE=IMMED. For more information, see "Limitations When TVBINXIT Is Active" on page 7.

Do not use DSIPSS in this exit routine. You can issue new messages by chaining them to the original message structure.

DSIEX16 does not receive a parse descriptor block (PDB). DSIEX16 uses the IFRCODAI internal function request like DSIEX02A does. However, DSIEX16 is written to run in 31-bit addressing mode exclusively.

DSIEX16 does not use DSIMQS on any part of the IFRCODAI structures when it is in the form given to DSIEX16. The routing list buffers whose addresses are in IFRAURTL are only used during the DSIEX16 interface. To issue DSIMQS for an IFRCODAI structure, ensure that HDRNEXTM is zero (there is only one IFRCODAI structure) and that IFRAURTL is zero. Use care in manipulating the HDRNEXTM fields and IFRAURTL fields to prevent accidental loss of buffers.

**Return Codes:** DSIEX16 differs from the other installation exits in that it ignores the return code, but it expects a zero. In the other NetView exits, these return codes indicate that messages are to be deleted or a new message is to be swapped for the original one.

In DSIEX16, you can swap or delete messages by direct manipulation of the buffers that are located by the USERMSG field of the DSIUSE parameter list. NetView uses whatever structure the USERMSG field refers to on return and performs the actions that are indicated.

Use the DSIGET macro to get new buffers and the DSIFRE macro to release buffers you have removed. Use the Q=NO option and specify subpool 0 on these DSIGET and DSIFRE requests. If you release all the buffers, you set the USERMSG field to zero to indicate that there are no buffers when you return.

**Additional DSIEX16 Features Not Found In DSIEX02A:** DSIEX16 specifies multiple IFRCODAI buffer structures chained by means of the HDRNEXTM buffer chaining field. The first such buffer is the representation of the original message, including display and actions before DEFAULTS and OVERRIDE options are considered. See Figure 10 on page 43.
Each EXEC action, specified on the NetView automation table statements that match the message, creates an additional IFR-type IFRCODAI chained using the HDRNEXTM field. An exception is for multiple EXEC actions that have only a ROUTE keyword. EXEC actions with only ROUTE keywords combine with existing IFRCODAI route lists, if the same type (ONE or ALL) of ROUTE-only buffer exists. Each of the IFR-type IFRCODAIs contains its own set of message buffers.

If a command keyword is specified, the corresponding IFR-type IFRCODAI has the IFRAUCMD bit on and contains the address of the command buffer in IFRAUCMB.

If the command buffer originated in the local domain, IFRAUCMB points to a IFRCODCR internal function request buffer (DSIFR). If DSIPSS sends the command from an NNT task, the command buffer is an HDRTYPEEX, not an HDRTYPEI (DSIIFR).

Figure 10. DSIEX02A, DSIEX16, DSIEX16B, and DSIEX17 Interface. IFRAURCL(n) is a notation to indicate that there are as many IFRAURCL entries as the value of IFRAURCC.
Writing Installation Exit Routines

If a route keyword is specified, the corresponding IFR-type IFRCODAI has the IFRAURTL field set to the address of a routing list buffer, as mapped by the IFRAURTB map within DSIIFR. The IFRAURTL field contains the address of a standard NetView buffer that must contain a standard BUFHDR and extension HDRMCEXT. You need to set buffer fields HDRBLENG, HDRMLENG, and HDRTDISP. NetView does not use other header fields for this buffer. HDRTDISP is the offset into the buffer at which the field IFRAURTB is located.

For all buffers processed within the NetView program and prefixed with a BUFHDR, HDRTDISP is set to a value greater than the last buffer header field. The value can change from one release of NetView to another. For references to specific data in buffers, use HDRTDISP as the starting index instead of assuming the data has a specific value. The value is based on the currently defined buffer headers, and is typically a 24, 36, or 46 decimal. Other values occur when commands are called (for example, when the IFRCODCR is skipped over calling a command processor).

On entry to DSIEX16, the DISPLAY, BEEP, HOLD, and HCYLOG actions are the same in each IFR-type IFRCODAI on the chain pointed to by the original AIFR’s HDRNEXTM field. However, the original AIFR can be different from those on the chain. SYSLOG and NETLOG actions are indicated in the original (first) buffer, and the additional EXEC action buffers indicate that logging is suppressed with overrides not available. This ensures that a message is logged only once, unless the DSIEX16 exit directs otherwise.

NetView uses the USERMSG field on return as the chain of IFR type IFRCODAIs to be processed. If USERMSG is set to zero by DSIEX16, the installation exit frees all buffers passed. These buffers are nonqueued, subpool zero storage. These buffers are 31-bit addressable buffers. When USERMSG is nonzero, it points to the chain of IFRCODAI buffers. The installation exit can return multiple IFR-type IFRCODAIs without routing lists (with or without commands), and multiple IFR-type IFRCODAIs with routing lists or commands or both.

Buffers with no routing list or commands are processed as display or logging buffers or both, depending on the settings of action flags in the IFR-type IFRCODAI. For example, independent structures can be returned with only a display action in one structure, and logging actions in the other structure to enable independent processing of logged and displayed buffers. In this case, use the flags specifying that overrides are to be ignored to prevent accidental display of log-only buffers. NetView sets logging actions only in the original IFR-type IFRCODAI to ensure that the message is logged only once. Display of the buffers is inline as if the DSIPSS macro is issued originally for all the buffers.

Buffers that have no routing list but contain a command are queued to the task that is processing the message using DSIMQS. This causes the command to be run when the task processes the message queue.

Buffers with routing lists are queued to the specified tasks using DSIMQS. The routing lists are generated exactly as specified by the AUTOMSG definitions and can include both active and inactive tasks. Any tasks that are inactive do not receive the buffers.

DSIEX16 can determine the current defaults by checking the bits in the MVTAIDFVT field in the DSIMVT.
Note: When IFRCODAI buffers have commands, the DISPLAY, logging, BEEP, and
HOLD options are not referenced during the command execution.

DSIEX16 has access to user bit and character fields in the IFR-type IFRCODAI that enable signaling between DSIEX02A, DSIEX03, DSIEX16, DSIEX17, user-written tasks, or user-written commands. Actions (including HOLD and BEEP) specified in the IFR-type IFRCODAI supersede the DSIMVT defaults. DSIEX16 cannot determine the overrides but can mark buffers that are not affected by the overrides. NetView does not process the default options BEEP=DISABLE or HOLD=DISABLE when specified by DSIEX02A, DSIEX16, or the NetView automation table, unless the IFRCODAI buffer has the corresponding force flag on in IFRAUTA4 and the action flags in IFRAUTA1 or IFRAUTA2 contain a nondefault value.

When a force flag is set on in IFRAUTA4 before NetView automation table processing (such as in DSIEX02A or before), NetView automation processing does not consider the corresponding action, even if the NetView automation table specifies a value for it.

Because the force action bits specify display and logging actions, their presence implies that no command action processing occurs when the NetView automation table is searched. Conversely, buffers that contain commands (IFRAUCMD=ON) cause NetView to ignore the display and logging action indicators, including the action force flags.

Because logging actions are serviced under the task under which the NetView automation table is invoked, buffers marked as forced nondisplay are not routed to other tasks by a matching NetView automation table entry.

You can pass information from your NetView automation table to DSIEX16. For example, you can direct what DSIEX16 does for a class of messages or MSUs that the automation table has identified. Use the CMD() part of the IF statement to send data to the DSIEX16, marked by using a dummy command name. To do this, first define a dummy command with a name (for example, CMDEX16) in the CMDMDL definitions in DSICMD. You can then code this in an automation table, even though you do not intend to execute it. For example:
EXEC(CMD('CMDEX16 ' PARM1 PARM2))

DSIEX16 then looks at the command buffer chained from IFRAUCMB for the dummy command name, and uses the PARM1 and PARM2 set by the automation table. If the dummy command name is present, DSIEX16 does what PARM1 and PARM2 indicate. DSIEX16 dechains and frees the IFRAUCMB buffer with the dummy command name command buffer before returning to NetView. This prevents the program from actually running the command.

To edit or create log buffers, set the appropriate logging options in a separate copy of the original IFRCODAI with its own data buffer chain. If you chain the new IFRCODAI with the appropriate logging flags or OVERRIDE suppressing flags, or both, to the original IFRCODAI buffer chain, the edited data is logged to the specified logs. You can set the IFRAUDIS flag off, and set the DISPLAY override suppressed flags on, to prevent accidental display of the log buffers if the task specifies override DISPLAY=YES.

DSIEX16 also provides for message processing accounting. The IFR-type IFRCODAI provides indicators that show a record of processing of a particular message, such as whether the message is solicited, came from the PPT, was an
Writing Installation Exit Routines

authorized-receiver routed message, satisfied an &WAIT or a WAIT, or was suppressed or processed by the NetView automation table.

Note: DSIEX02A has only one IFR-type IFRCODAI and its HDRNEXTM is zero. The command and routing buffers created by the NetView automation table first appear in DSIEX16. See Figure 10 on page 43 for an illustration of the DSIEX16 interface.

DSIEX16B: Post-Automation Table Exit for MSUs

Description: This installation exit is called for MSUs subject to automation table processing. This installation exit is called after automation actions, such as commands to be scheduled, and hardware monitor filter settings have been determined and are accessible in the AIFR, but before the actual actions are taken. DSIEX16B is loaded during NetView initialization.

Example of Use: DSIEX16B provides an opportunity to modify the effects of the hardware monitor SRFILTER command and the automation table, after they are determined but before automation actions take effect.

Use this exit to modify the filter settings, color, and other presentation attributes, and to change or substitute the command buffers to be executed. You can also use this exit to help monitor the effectiveness of filtering and automation.

Coding Considerations: Installation exit DSIEX16B is called when the AIFR driving the exit contains MSU data instead of standard message data. The AIFR is passed to the exit through the standard installation exit parameter list (DSIUSE) whose address is passed in register 1. You can customize the exit programs based on the type of AIFR data to be processed, or you can code a single program to be used for both message AIFRs and MSU AIFRs.

A buffer containing the MSU automated data is present. Other buffers containing the following can also be present:
- HIER information
- EXEC CMD actions

Return Codes: DSIEX16B handles return codes the same way as DSIEX16. Like DSIEX16, it ignores the return code, but it expects a zero. Only register 15 can be used to pass the return code.

DSIEX17: MVS Message and DOM Receive

Description: The NetView program calls DSIEX17 upon receipt of MVS messages and delete operator messages (DOMs) on MVS systems.

NetView keeps track of action messages in order to process DOMs as they arrive from MVS. NetView considers a message to be an action message if any of the following are true:
- IFRAUWWR (or CPOMLR) is on.
- Descriptor codes 1, 2, 3, or 11 in (IFRAUWDS or CPOCDESC) are set.

On return from DSIEX17, NetView determines whether these conditions are true. If any are true, NetView activates IFRAUACN to indicate that the message is an action message.
Notes:

1. Typically, MVS does not generate DOMs for messages that are not considered action messages. If DSIEX17 changes a nonaction message to an action message, a DOM may never be generated by MVS. Because storage is needed to keep track of action messages, storage growth can occur if NetView processes an action message for which no DOM is forthcoming. DSIEX17 is also invoked for user messages that originated with the DSIIMMDB or CNMMPMDB services on any operating system. DSIEX17 runs under the OST, PPT, NNT, and CNMCSSIR tasks. For messages, this exit is called before automation or ASSIGN processing.

2. The interface used is similar to DSIEX02A, but DSIEX17 is called with TVBINXIT off and must be written in assembler.

3. When running with Extended Consoles, the CNMCSSIR task gets a console with attribute AUTO(YES). This causes the CNMCSSIR console to receive all messages marked AUTO(YES) in MPF. The IFRAUSIR flag is set when the CNMCSSIR console receives a copy of a message that was solicited to another NetView console. If this message is an action message, not deleted by DSIEX17 and IFRAUDMN is not on, it will be used to facilitate DOM routing. If IFRAUSIR is on, this (duplicate) message will not normally be processed after DSIEX17 returns to CNMCSSIR, except for the creation of a queue element kept until a matching DOM arrives.

DSIEX17 should turn on IFRAUDMN if the message is known to be one for which no DOM is ever issued. This will signal NetView to avoid queuing data to remember this message, saving you storage. In addition, if a DOM were actually issued, the NetView program would not route it, because no record of the message was kept by NetView.

In summary, if IFRAUSIR is on, DSIEX17 is being asked to decide whether the message should be remembered by the CNMCSSIR task. DSIEX17 must turn on IFRAUDMN to indicate the message should not be retained.

4. DSIEX17 can set IFRAUDMA on to indicate that the deletion events for this message should also be automated. This is the same as using DOMACTION(A) in the automation table. DSIEX17 should set IFRAUACT on if IFRAUDMA is being set on.

When IFRAUDMA is on, the message being deleted will drive the automation table a second time. No user exits are driven by the delete action. The automation table then has the opportunity to issue a REXX (or other) procedure to react to the message deletion. The automation table is driven once per copy of the message being deleted, on each task that has a held copy of the message.

5. DSIEX17 can set IFRAUDMN on to indicate that the deletion events for this message should be ignored. This is the same as using DOMACTION(N) in the automation table.

This option prevents the CNMCSSIR task from queuing data to remember messages requiring deletion. It does not prevent messages from being held on operator screens or from being retained by autotasks.

6. DSIEX17 can test IFRAUDTO to determine whether the MVS DOM was DOM-by-TOKEN.

7. DSIEX17 will not see the IFRAUDLO bit set, and should not alter the setting.

Example of Use: You can use DSIEX17 to process DOMs for action messages that you convert into other action messages (with a new DOM identification value). This processing is helpful when you reformat messages such as tape mount requests.
To avoid storage growth when NetView processes an action message for which no DOM is forthcoming, set the IFRAUDMN bit on to indicate that a DOM for this message will not be received. For example, suppose DSIEX17 creates a table of message IDs. The table designates action messages for which no DOM will be generated. Assume that NetView receives a message that matches one in the table, the exit might contain the following code:

```
01 IFRAUI33,IFRAUDMN INDICATE NO DOM IS EXPECTED
```

This code tells the main line code that no DOM will be issued for this message, which prevents unnecessary storage growth.

Users of DSIEX17 may also find the IFRAUSIR flag useful. The CNMCSSIR task can process duplicate messages. For example, if a message is issued directly to a NetView owned console, the CNMCSSIR task can receive a copy of the message because of its AUTO(YES) console attribute. The IFRAUSIR flag will be turned on for the message received by CNMCSSIR task. This duplicate message is used by the CNMCSSIR task to determine whether the message is an action message. Then, the message is discarded to prevent duplicate automation. If DSIEX17 encounters a message under the CNMCSSIR task that has the IFRAUSIR flag turned on, it is recommended that this message not be dropped. A change to this duplicate message could affect how action messages and their respective DOMs are processed. You should, however, use the IFRAUDMN bit to indicate if such a duplicate message is also one for which no DOM will ever be issued.

**Coding Considerations:** DSIEX17 is supported only in 31-bit addressing mode.

An AIFR is the buffer structure given to DSIEX17.

**Return Codes:** DSIEX17 can pass the following return codes:

**USERASIS (0)**
NetView continues processing with the AIFR found in the USERMSG field. You can modify the AIFR directly as in DSIEX02A. If you return a zero in the USERMSG field, you are responsible for freeing the AIFR and all associated buffers.

**USERDROP (4)**
NetView frees the AIFR in the USERMSG field.

**USERSWAP (8)**
NetView continues processing on the AIFR found in register zero (0). This is similar to USERASIS, except that you return the AIFR in register zero (0) instead of USERMSG.

**Note:** The AIFR that is originally passed to the exit in the USERMSG field becomes the responsibility of the exit. If the same AIFR is not passed back to NetView in register zero (0), the exit is responsible for freeing the original AIFR.

**DSIEX18: Network LOG BROWSE Exit**

**Description:** The NetView LOG BROWSE function passes all network log entry records which would normally be displayed on an operator’s screen to installation exit DSIEX18. DSIEX18 decides whether or not to display the records by setting return codes.

When DSIEX18 is given control, register 1 points to the USE control block DSIUSE. DSIUSE contains USERMSG, which points to a message buffer. This message buffer
contains the following data:

| CHAR(8) | Operator ID that issued the request |
| CHAR(248) | (for future use) |
| CHAR(6) | Message sequence number |
| CHAR(1) | Blank |
| CHAR(8) | Operator ID |
| CHAR(1) | Blank |
| CHAR(5) | Domain ID |
| CHAR(1) | Blank |
| CHAR(2) | Message code |
| CHAR(1) | Blank |
| CHAR(8) | Time of day (HH:MM:SS) |
| CHAR(1) | Blank |
| CHAR(1) | Important message indicator |
| CHAR(1) | Blank |
| CHAR(220) | Message text |

**Example of Use:** You can use DSIEX18 to suppress certain log records from being displayed on an operator’s terminal.

**Coding Considerations:** DSIEX18 is a global exit routine. It is written in assembler language, not in HLL (PL/I or C). DSIEX18 is also written to run in a 31-bit addressing mode exclusively. It receives one record at a time from LOG BROWSE.

When a user issues the BLOG command, filtering occurs based first on the BLOG specification. Records that are not filtered by BLOG are then passed to the DSIEX18 exit for processing.

**Return Codes:** DSIEX18 can pass the following return codes:

- **USERASIS (0)**
  - LOG BROWSE displays the record.
- **USERDROP (4)**
  - LOG BROWSE suppresses the record.

**DSIEX19: RUNCMD Exit**

**Description:** The NetView RUNCMD exit DSIEX19 is called after normal enterprise security checking has authorized the use of the RUNCMD command. The exit is passed the text following the RUNCMD command verb.

**Example of Use:** You can use DSIEX19 to provide security checking at the service point command level. You can do this security checking by calling DSICES or DSIKVS or writing your own command and keyword checking code. To use DSICES or DSIKVS for security checking, you can define a model CMDMDL statement specifying DSISPCMD. You can then define command, keyword, and value authorization checking based on the model CMDMDL statement.

**Coding Considerations:** The following input is provided upon invocation of DSIEX19 in the DSIUSE control block:

- DSIEX19 is passed a read-only copy of the command buffer in the USERMSG field. See "[DSIXRCMD: RUNCMD Installation Exit Buffer" on page 161] for a description of the command buffer pointed to by USERMSG.
- Other DSIUSE fields passed are: USERLU, USEROPID, USERSWB and USERTVB. USERPDB is set to zero (0).
This user exit must be linked as AMODE=31, RMODE=ANY.

NetView provides an example of this exit as CNMS4307. This sample shows an example of authority checking various commands that are imbedded within a RUNCMD command. The sample uses the DSICES and DSIKVS macros for authority checking. The command and keyword names are translated by the sample to alternate names, so that the names are acceptable to the DSICES and DSIKVS macros.

**Return Codes:** DSIEX19 can pass the following return codes:

- **USERASIS (0)**
  Continue RUNCMD processing to send the command to the service point.

- **Any return code other than USERASIS**
  Discontinue RUNCMD processing. This causes message BNH192E to be issued indicating that processing of the RUNCMD has stopped.

**DSIEX20: SAW Exit**

**Description:** The NetView session awareness exit DSIEX20, or SAW exit, is called each time the session monitor receives a SAW record from VTAM. The SAW record is converted into a NetView-specific representation called an internal SAW, or ISAW, record. The DSIEX20 exit is called after the SAW record is converted into an ISAW record and before the NetView SAW filtering processing begins.

When DSIEX20 processing starts, register 1 points to control block DSIUSE. The USERMSG field of DSIUSE points to the message buffer containing a standard NetView buffer header, as defined by BUFHDR, followed by the ISAW. The AAUTISAW control block can be used to parse the ISAW, which begins at the offset indicated by BUFHDR HDRTDISP field.

The message buffer includes workspace after the ISAW. The workspace can be passed from one DSIEX20 invocation to the next. This workspace is at least 1000 bytes long. The exact amount of workspace can be calculated by the following formula:

\[
\text{workspace} = \text{HDRBLENG} - \text{HDRMLENG} - \text{HDRTDISP}
\]

See [“BUFHDR: Buffer Header” on page 109](#) and [“DSIUSE: Installation Exit Parameter List” on page 156](#) for more information.

**Example of Use:** The ISAW records cannot be changed, but they can be kept or discarded. The DSIEX20 exit provides the functions for more granular filtering of SAW records than that provided by VTAM or NetView.

**Note:** For more information about record filtering, refer to the KCLASS and MAPSESS statements in the [Tivoli NetView for z/OS Administration Reference](#).

You can use DSIEX20 to access the primary and secondary session endpoint names, PCID, session type (for example, LU-LU and SSCP-PU) and notification type (for example, session start and bind pending) to determine whether to keep or discard the SAW record.

**Coding Considerations:** DSIEX20 has the following coding considerations:

- To minimize performance impacts, the path length of DSIEX20 should be as short as possible.
- DSIEX20 must be run in 31-bit addressing mode.
AAUTISAW defines the format of the ISAW record. The DSECT comments describe the ISAW record fields.

All AAUTISAW name fields are coded in EBCDIC. If VTAM does not provide information for a name field, the field will be blank. The name fields are:

- ISAWCROSS
- ISAWCOSA
- ISAWLOGM
- ISAWFQCP
- ISAWPNAM
- ISAWSNAM
- ISAWPNET
- ISAWSNET
- ISAWPALI
- ISAWSALI

Because there is no PDB, the USERPDB pointer is used to address an 8-byte field that is initialized to zeros. If DSIEX20 changes this 8-byte field, the field is treated as the KCLASS name by session monitor, overriding MAPSESS statements.

Return Codes: DSIEX20 can pass the following return codes:

USERASIS (0)
Exit has no affect. System-defined SAW filtering will determine the disposition of the SAW record.

USERDROP (4)
The SAW record is discarded.

USERKEEP (8)
System defined SAW filtering is bypassed and the SAW record is kept.

Other return code
The exit has no effect. System-defined SAW filtering will determine the disposition of the SAW record. Message DWO050E is logged.

Note: The return codes will only affect the session types and SAW types supported by SAW filtering. Examples of supported types are session-start and init-pending for SSCP-LU and LU-LU sessions. Session-end is treated the same as its associated session-start.

DSIEX21: DSITCPRF Encryption
Refer to the Tivoli NetView for z/OS Security Reference for more information about DSIEX21 encryption code and messages.

XITBN: BSAM Empty File

Description: The DST calls XITBN if the DST encounters a BSAM open failure because of an empty data set or file.

Example of Use: You can use XITBN to place a record in the empty data set. Code this installation exit only if you want to write a BSAM subtask using the DST as a base.

Coding Considerations: XITBN can use only the service facilities available to the DST. The exit cannot issue the macros DSIZVSMS and DSIZCSMS.

Return Codes: XITBN can pass the following return codes:
Writing Installation Exit Routines

**USERASIS (0)**
- Use the NetView sequential log initialization buffer.

**USERSWAP (8)**
- Use the sequential log initialization buffer whose address was returned by the exit in register zero (0).

A return code other than USERSWAP or USERASIS is treated as if the return code was USERASIS and message CNM474I is issued.

**XITBO: BSAM Output**

**Description:** The DST calls XITBO immediately before the record is written to the BSAM database.

**Example of Use:** You can use XITBO to modify the record before it is sent to the BSAM data set or file.

**Coding Considerations:** XITBO can use only the service facilities available to the DST. The exit cannot issue the macros DSIZVSM and DSIZCSMS.

Avoid coding installation exits for frequently executed functions, such as BSAM input or BSAM output, because they can significantly degrade performance.

**Return Codes:** XITBO can pass the following return codes:

- **USERASIS (0)** Log the record.
- **USERDROP (4)** Do not log the record.
- **USERSWAP (8)** Log the record in the buffer whose address was returned by the exit in register zero (0).

**XITCI: CNM Data Input**

**Description:** The DST calls XITCI after communication network management (CNM) data is received through the communication network management interface (CNMI) or the MS transport.

**Example of Use:** You can use XITCI to modify CNM input data for the hardware monitor.

**Coding Considerations:** XITCI can use only the service facilities available to the DST. The exit cannot issue the macros DSIZVSM and DSIZCSMS.

If you specify USERSWAP (8), the substitute buffer must contain a valid network services request unit (RU) of the same type as the input RU. Refer to the SNA library for a description of RU formats.

DSICRTR is the subtask responsible for routing RECMS, RECFMS, ROUTE-INOP, CNM, NMVT, and cross-domain alerts. XITCI invoked under the DSICRTR subtask provides access to unsolicited CNM data prior to NetView routing.

CP-MSUs and MDS-MUs are not routed through DSICRTR and are only accessible under the BNJDSERV subtask.
XITCI invoked under a DST other than DSICRTR can access CNM data routed to that particular subtask.

Do not use DST installation exits under the network product support (NPS) task named DSIGNDS.

Network services request units are routed as shown in Table 7.

<table>
<thead>
<tr>
<th>Request</th>
<th>Header Value</th>
<th>Receiving Subtask</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECMS</td>
<td>X'010381'</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>RECFMS</td>
<td>X'410384'</td>
<td>AAUTSKLP, BNJDSEVR</td>
</tr>
<tr>
<td>ROUTE-INOP</td>
<td>X'410289'</td>
<td>AAUTSKLP</td>
</tr>
<tr>
<td>CNM</td>
<td>X'810814'</td>
<td>AAUTSKLP</td>
</tr>
<tr>
<td>NMVT</td>
<td>X'41038D'</td>
<td>AAUTSKLP, BNJDSEVR, DSIGNDS</td>
</tr>
<tr>
<td>CP-MSU</td>
<td>X'1212'</td>
<td>BNJDSEVR, DSIGNDS</td>
</tr>
<tr>
<td>MDS-MU</td>
<td>X'1310'</td>
<td>BNJDSEVR, DSIGNDS</td>
</tr>
<tr>
<td>Cross-Domain Alert</td>
<td>X'1040'</td>
<td>BNJDSEVR</td>
</tr>
</tbody>
</table>

Table 7. Routing of Network Services Request Units

For the various receiving subtasks, Table 8 shows the major vector keys that can be found in the specific RU.

<table>
<thead>
<tr>
<th>Major Vector Key</th>
<th>Description</th>
<th>Receiving Subtask</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0000'</td>
<td>Alert</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'0001'</td>
<td>Link Event</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'0002'</td>
<td>Resolution</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'000F'</td>
<td>CMIP statistics</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'0010'</td>
<td>Trace</td>
<td>AAUTSKLP</td>
</tr>
<tr>
<td>X'0025'</td>
<td>PD Statistics</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'006F'</td>
<td>Send Message to Operator</td>
<td>DSIGNDS</td>
</tr>
<tr>
<td>X'0080'</td>
<td>RTM</td>
<td>AAUTSKLP</td>
</tr>
<tr>
<td>X'132E'</td>
<td>RECFMS envelope</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'1332'</td>
<td>Link configuration data</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'13FF'</td>
<td>Reserved</td>
<td>BNJDSEVR</td>
</tr>
<tr>
<td>X'154D'</td>
<td>Routing and targeting instruction</td>
<td>BNJDSEVR</td>
</tr>
</tbody>
</table>

Table 8. Routing of RUs by Major Vector Key

The focal point transfer request unit (RU) header is part of the CNM router support. All cross-domain unsolicited alert data is routed to the CNM router, and the focal point transfer RU header carries management services information between distributed host and the focal point.
Writing Installation Exit Routines

The fields in the focal point transfer RU header are listed in Table 9.

Table 9. Focal Point Transfer RU Header

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HDR LEN</td>
<td>2 bytes, binary</td>
<td>Length of the total RU (includes RU header and NMVT)</td>
</tr>
<tr>
<td>2</td>
<td>HDR ID</td>
<td>2 bytes</td>
<td>Always X’1040’</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>11 bytes</td>
<td>For the NetView program use only</td>
</tr>
<tr>
<td>15</td>
<td>DOMID LEN</td>
<td>1 byte, binary</td>
<td>Originator’s domain ID length</td>
</tr>
<tr>
<td>16</td>
<td>DOMAIN ID</td>
<td>8 bytes, char</td>
<td>Originator’s domain ID, padded with blanks</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
<td>20 bytes</td>
<td>For the NetView program use only</td>
</tr>
<tr>
<td>44</td>
<td>Name</td>
<td>Variable</td>
<td>NMVT data</td>
</tr>
</tbody>
</table>

If the data is an alert forwarded using the NV-UNIQ/LUC alert forwarding protocol, the first 44 bytes of the data are mapped by the focal point transfer RU and the remainder of the data is the actual NMVT. The first 2 bytes of the focal point transfer RU contain the length of the entire buffer (FPT RU + NMVT). The next 2 bytes contain the header ID, which is X’1040’. The 16th byte contains the length of the originating domain ID, and the 17th–24th bytes contain the actual originating domain ID. When returning a substitute buffer, do not modify the focal point transfer RU (the first 44 bytes); replace only the NMVT portion of the buffer with a valid NMVT.

Note: For more information about the format for a specific RU and how to access data within the RU, refer to the SNA library, NCP and EP Reference Summary and Data Areas, and “Flows and Control Blocks” in the Tivoli NetView for z/OS Diagnosis Guide.

Return Codes: XITCI can pass the following return codes:

USERASIS (0)
Pass the buffer with the CNM data.

USERDROP (4)
Do not pass the buffer with the CNM data.

USERSWAP (8)
Substitute the CNM data buffer whose address was returned by the exit in register zero (0).

USEREXLG (252)
Stands for external log only. The hardware monitor, executing under task BNJDSERV, records the message to the system management facilities (SMF) only and then discards it. No data is logged to the database. This processing is the same for all alerts including forwarded alerts. This occurs when you designate NPDA REPORTS ON. Refer to the NetView online help for an explanation of the REPORTS command.

USEREVNT (253)
The hardware monitor, executing under task BNJDSERV, records the message as an event or statistic on its database, but not as an MSU. The hardware monitor recording filters are not applied to the message as they would be normally. Instead, the ESREC filter is set to PASS and all other recording filters are set to BLOCK.
For NV-UNIQ/LUC alert forwarding protocol forwarded alerts and SNA-MDS forwarded alerts from nonNetView entry points, only event data is recorded to the database. For SNA-MDS forwarded alerts from an entry point NetView, no data is recorded to the database.

Refer to the description for the SRFILTER command in the NetView online help for an explanation of the recording filters.

Note: Some alerts displayed by the hardware monitor do not drive the XITCI installation exit and are therefore still logged as alerts. One example is alerts generated by the 4700 Support Facility.

If return code USEREXLG (252) or USEREVNT (253) is returned for an input record, the input record is not processed as an alert. The hardware monitor alert recording filter is not passed so the input record is not forwarded to the alert focal point.

Messages that are blocked as a result of a filter from the rate function might not be automated. You can use the AUTORATE statement to control this.

Note: Refer to the RATE and AUTORATE statements in the Tivoli NetView for z/OS Administration Reference for more information about these statements.

XITCO: CNM Interface Output

Description: The DST calls XITCO prior to a request for CNM interface output.

Example of Use: You can use XITCO to modify the request for CNM data (forward RU).

Coding Considerations: XITCO can use only the service facilities available to the DST. The exit cannot issue the macros DSIZVMS and DSIZCWS.

Do not use DST installation exits under the network product support (NPS) task named DSIGDS.

Note: If a substitute buffer is returned in register zero (0), the data is a valid SNA RU. Refer to the SNA library for a description of RU formats.

Return Codes: XITCO can pass the following return codes:

USERASIS (0)  
Send the original CNM data.

USERDROP (4)  
Do not send the original CNM data.

USERSWAP (8)  
Send the CNM data in the buffer whose address was returned by the exit in register zero (0).

XITDI: Data Services Task (DST) Initialization

Description: The DST calls XITDI for each statement that the DST reads during initialization. When the end of the file is reached, this installation exit is entered and two DSIUSE fields, USERMSG and USERPDB, are set to 0, indicating that there is no more data. You can code up to ten module names per DST for each user-written exit routine.
Note: Refer to the Tivoli NetView for z/OS Administration Reference for more information about XITDI during DST initialization. Also, see “Data Services Task (DST)” on page 90.

Example of Use: You can add XITDI to the DST initialization deck to provide user initialization values for DST initialization.

Coding Considerations: XITDI can use only the service facilities available to the DST. The exit cannot issue the macros DSIZVSMS and DSIZCSMS. XITDI cannot refer to DSRB fields.

Note: If all initialization data is to be processed by XITDI, specify the DST initialization statement that identifies XITDI as the first statement in the DST initialization member.

Return Codes: XITDI can control the processing of an initialization statement by the DST by returning the following values:

USERASIS (0)  
Process the original DST initialization statement.

USERDROP (4)  
Do not process the original DST initialization statement.

USERSWAP (8)  
Process the DST initialization statement in the buffer whose address was returned by the exit in register zero (0).

When called for an end-of-file situation, a nonzero return code in register 15 indicates that the DST can be stopped.

XITVI: VSAM Input

Description: The DST calls XITVI after a DSIZVSMS macro is issued. The record is read from the VSAM database, but it is not yet passed to the requesting data services command processor.

Example of Use: You can use XITVI to modify the record after it is retrieved from a VSAM data set or file.

Coding Considerations: XITVI can use only the service facilities available to the DST. The exit cannot issue the macros, DSIZVSMS and DSIZCSMS.

Avoid coding installation exits for functions that could cause a wait, such as VSAM input or VSAM output, because the wait can significantly degrade performance.

Return Codes: XITVI can pass the following return codes:

USERASIS (0)  
Process the original data read from the VSAM file.

USERDROP (4)  
Do not process the original data read from the VSAM file.

USERSWAP (8)  
Process the data in the buffer whose address was returned by the exit in register zero (0).
XITVN: VSAM Empty File

Description: The DST calls XITVN if the DST encounters a VSAM open failure because of an empty data set or file.

Example of Use: You can use XITVN to place a record in the empty data set. NetView provides its own XITVN for VSAM logs generated under the DST. Code this installation exit only if you wish to write your own VSAM subtask using the DST as a base.

Coding Considerations: XITVN can use only the service facilities available to the DST, excluding macros DSIZVSMS and DSIZCSMS. Only VSAM key-sequenced data sets (KSDS) are supported. Do not replace the NetView-provided XITVN exits for the DSILOG and DSITRACE subtasks.

Return Codes: To initialize the VSAM data set or file, return the USERSWAP return code and have register zero (0) point to a buffer that contains the record to be used. A return code other than USERSWAP causes the DST to end.

XITVO: VSAM Output

Description: The DST calls XITVO immediately before the record is written to the VSAM database.

Example of Use: You can use XITVO to modify the record before it is sent to the VSAM data set or file.

Coding Considerations: XITVO can use only the service facilities available to the DST. The exit cannot issue the macros DSIZVSMS and DSIZCSMS.

Avoid coding installation exits for frequently executed functions, such as VSAM input or VSAM output, because the functions can significantly degrade performance.

Return Codes: XITVO can pass the following return codes:

USERASIS (0)
Write the original data in the VSAM file.

USERDROP (4)
Do not write the original data in the VSAM file.

USERSWAP (8)
Write (in the VSAM file) the data in the buffer whose address was returned by the exit in register zero (0).

XITXL: External Logging

Description: The DST calls XITXL whenever data can be sent to an external log using DSIWLS with EXTLOG parameter. For example, the session monitor performs external logging of response time and configuration data.

Coding Considerations: XITXL can use only the service facilities available to the DST. The buffer passed to the installation exit contains the standard header, with HDRTDISP pointing to control block DSIELB. The data that is to be logged follows DSIELB.

Return Codes: XITXL can pass the following return codes:
Writing Installation Exit Routines

USERASIS (0)
Write the original SMF record.

USERDROP (4)
Do not write the original SMF record.

USERSWAP (8)
Write the SMF record in the buffer whose address was returned by the exit in register zero (0).

Unused Installation Exits
NetView attempts to load the global exits (DSIEXnn) and the DST exits (XITnn) specified with DSTINIT statements. If a load attempt fails, NetView issues the message:

DSI090I LOAD FAILED FOR NCCF MODULE exitname

This message is a not a cause for concern for DSIEXnn exits unless you expect a particular exit to load that did not load.

Installing an Installation Exit Routine
Link-edit the installation exit routine load module into the NetView load library. For global installation exits, use the appropriate DSIEXnn name. For DST installation exits, use a name that you select. Use only one load module for each routine. Conditional selection upon exiting is not valid.

Global installation exit routines are automatically loaded when NetView starts. DST installation exit routines are loaded when the DST starts, provided you have specified them on the DSTINIT statement.

See "Testing Your Program" on page 3 for information about testing your exit routine before use.

Template for an Installation Exit Routine
The basic structure of an installation exit routine, including standard entry and exit linkage, is available in a template as part of the NetView sample library. This template runs as written for any of the NetView installation exits. However, no functions are performed until you add your code at the designated place in the template. The template can be found as member CNMS4282 of the CNMSAMP data set.
Chapter 4. Writing Command Processors

This chapter describes designing, coding, and installing the assembler language command processors for NetView. Command processors perform a particular service or function, such as extracting relevant data from a control block and presenting the data to an operator.

Overview of Command Processors

Command processors run in any of several execution environments, determined by the command processor’s type, defined by the CMDMDL statement in DSICMD. The CMDMDL statement identifies the command processor as one of the following types:

- **TYPE=R**  Regular commands
- **TYPE=H**  High-priority commands
- **TYPE=I**  Immediate commands
- **TYPE=D**  Data services commands
- **TYPE=B**  Regular and immediate commands combined
- **TYPE=RD** Regular and data services commands combined

In addition, long-running commands are composed of regular (or type RD or type B) commands. Parts of long-running commands are coupled by their internal processing.

Regular Command Processors

A regular command runs under the operator station task (OST), NetView-NetView task (NNT), or primary program operator interface task (PPT). A regular command receives control with the TVBINXIT bit set off. This means that the NetView program and VTAM exit routines, and immediate command processors, can interrupt the processing of a regular command.

For specific coding instructions, see "Writing a Full-Screen Command Processor" on page 67 and "Calling the Command Processor" on page 18.

High-Priority Command Processors

A high-priority command processor is the same as a regular command processor except that type H commands are queued at high priority regardless of the setting of CMD priority on the DEFAULTS and OVERRIDE commands.

Immediate Command Processors

An immediate command runs under the OST and NNT environments. Unlike regular commands, immediate commands receive control with the TVBINXIT bit set on. This means that they interrupt mainline processing and cannot be interrupted by another command.

An immediate command starts processing as soon as an operator enters the command, regardless of any regular command currently running. The requested function is performed at once, even if the task is in the middle of a large queue of work.
Data Services Command Processors

A data services command processor (DSCP) runs under the data services task (DST) environment. DSCPs perform communication network management (CNM) data services using the DSIZCSMS macro, VSAM data services using the DSIZVSMS macro, or both services. DSCPs are also appropriate for centralized or serialized user-defined functions that do not use CNM or VSAM services. See "Data Services Command Processor" on page 92 for details.

Combination Command Processors

One type of combination command processor runs as a regular or immediate command, depending on its environment. This command processor checks the TVBINXIT bit and processes the command as an immediate command if the bit is on or as a regular command if the bit is off.

Another type of combination command processor runs as a regular or data services command, depending on the task type indicated in the task vector block (TVB). If the task type is DST, the command runs as a data services command. Otherwise, the command runs as a regular command.

Long-Running Command Processors

A long-running command (LRC) processor enables other processing to continue after a command begins processing. The DSIPUSH macro provides for continuation by the same or another processor under varying conditions. The caller of the original command can run after that command returns. Other processing, such as messages, can occur between the calls to the various parts of the LRC processor.

LRC processors run under an OST, NNT, primary POI tasks (PPT), or DST (logoff routines only). Your operator can invoke an LRC processor using a command procedure, or an LRC processor can be called by another LRC processor. The LRC processors return control to the NetView program after scheduling work but before processing is complete. The NetView program then processes other work that is pending. Only long-running commands can act as a NetView component, suspending for unrelated operator commands, including ROLL, and resuming, in turn.

LRC processors are often used to retrieve data from another task or from another domain without enabling the calling function or calling command procedure to proceed in the midst of this retrieval. During this retrieval, the processor’s task can continue to receive messages and accept commands.

For specific coding instructions, see "Calling the Command Processor" on page 18.

Unattended and MVS Console Operator Task Command Considerations

Command processors using DSIPSS TYPE=ASYPANEL or other full-screen functions must test the TVBAUTOO bit. If this bit is 1, full-screen mode is not permitted. TVBAUTOO indicates an unattended or MVS console operator task.
Designing and Coding a Command Processor

Command processors adhere to the guidelines for user-written programming described in "General Coding Guidelines" on page 3. In addition, command processors conform to the special requirements described in this section. After coding your command processor, follow the instructions in "Installing a Command Processor" on page 78.

To accommodate error recovery, test the TVBRESET flag set by the RESET command. You can code your command processor to examine this flag regularly and to end prematurely if the flag is on.

Keep in mind that an OST, NNT, PPT, or autotask can be receiving work (for example, DOMs) of which you can be unaware. Failure to process that work can result in a filling of the task’s message queues, and the task can receive message DSI374A, indicating that the buffers threshold for the message queue has been reached.

Therefore, if you are coding a command processor for an OST, NNT, PPT, or autotask and it is going to run for an extended period of time, the command processor should be an LRC processor. An LRC processor returns control to the NetView program after scheduling work, but before processing is complete. The NetView program then processes other work that might be pending.

Input to the Command Processor

When the command processor gains control, the registers contain the following information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Undefined when command is first invoked; storage address for resume routines for long-running commands.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the command work block (CWB). The CWB contains the following:</td>
</tr>
<tr>
<td></td>
<td>• A user save area (CWBSAVEA) that is the command processor’s 72-byte save area.</td>
</tr>
<tr>
<td></td>
<td>• The address of the command buffer (CWBBUF) for a command call. This field is 0 for a RESUME, abend reinstate, or LOGOFF call.</td>
</tr>
<tr>
<td></td>
<td>• The address of a service work block (SWB) for calling service facilities (CWBSWB).</td>
</tr>
<tr>
<td></td>
<td>• The address of a parse descriptor block (CWBPDB) if the following are true:</td>
</tr>
<tr>
<td></td>
<td>– CWBBUF does not equal 0</td>
</tr>
<tr>
<td></td>
<td>– PARSE=Y was specified (or defaulted to) in the CMDMDL statement of the command processor</td>
</tr>
<tr>
<td></td>
<td>If PARSE=N is specified, the CWBPDB equals 0 (zero).</td>
</tr>
<tr>
<td></td>
<td>• A work area (CWBADATD) that is the command processor’s 256-byte temporary storage for keeping variables while remaining reentrant.</td>
</tr>
<tr>
<td>2-12</td>
<td>Unspecified.</td>
</tr>
<tr>
<td>13</td>
<td>The address of a standard 72-byte save area used to store the caller’s registers.</td>
</tr>
<tr>
<td>14</td>
<td>The return address.</td>
</tr>
<tr>
<td>15</td>
<td>The entry address of the command processor.</td>
</tr>
</tbody>
</table>
When a command results from the NetView automation table, the TVBAIIFR field contains the address of the message buffer structure that is automated. If TVBAIIFR=0, the command did not result from an automated message or MSU. See Figure 11 for an example of an automation internal function request buffer structure.

![Diagram of Automation Internal Function Request]

Figure 11. Automation Internal Function Request. An example of a buffer structure when a command processor is driven from an automation statement in the NetView automation table. If TVBAIIFR=0, this command is not driven by an automation statement.

### Output from the Command Processor

When NetView regains control, the registers and their contents are as follows:

- **15** A return code
- **0-14** Restored to caller’s contents

If a regular command processor is called by a command procedure (in NetView command list language, REXX, or, high-level language), the return code is made available to the caller (&RETCODE, RC, HLBRCT, respectively). NetView makes no other use of the return code.

For an immediate command, NetView ignores the return code.
For an LRC processor, the completion code is specified on the DSPOP macro invocation. See “DSIPOP: Remove Long-Running Command” on page 212 for more information about the DSPOP macro. The register 15 return codes returned upon command resumption indicate processing options. See “Message STIFLE” on page 74.

Control Blocks

Command processors usually access a command buffer and seven control blocks: CWB, PDB, SWB, TVB, TIB, MVT, and SVL. In addition, type RD command processors, running under a DST and type D command processors, require the DSRB. Further, a command driven by means of NetView automation can require the automation internal function request (AIFR). The command buffers CWB, PDB, SWB, DSRB, and AIFR are specific to the command being executed. The TVB and TIB are global to the task, and the MVT and SVL are global to the NetView program.

Figure 12 on page 66 illustrates an example of the required control blocks and their relevant pointers. For detailed descriptions of these control blocks, see “Chapter 7 Control Blocks” on page 103. Figure 11 on page 64 illustrates an example of a buffer structure when a command processor is driven from an automation statement in the NetView automation table.
Writing Command Processors

Figure 12. Example of Control Blocks Used by Command Processors. The NetView control block header appears at
the beginning of the CWB, DSRB, MVT, PDB, SVL, TIB, and TVB.

The • character is shown only as a place holder in the figure, because the character used to represent the first, third, and fifth PDBTYPE delimiters does not reproduce properly on all output devices. The delimiter is a blank, normally represented as a
lower-case b with a slash (diagonal line) running through the letter from lower left to upper right (b).

Figure 13 shows sample code to access a command buffer.
Writing a Full-Screen Command Processor

A full-screen command processor (FSCP) is a regular command processor that presents a full screen of data to an operator’s terminal and runs only under an OST. For line-by-line presentation to an operator’s terminal, see “Title-Line Output” on page 15.

An FSCP uses DSIPSS TYPE=ASYPANEL to present data. In conjunction with long-running command support, you can code an FSCP to enable additional OST work requests to be processed without ending the full-screen presentation. Issue DSIPSS TYPE=ASYPANEL with both output data and input parameters specified.

```
L R3,CWPDB  GET ADDRESS OF PDB
USING DSIPDB,R3  R3 IS BASE FOR PDB
LA R4,PDBENTRY  GET ADDRESS OF PDB TABLE
USING PDBENTRY,R4  R4 IS BASE FOR A PDB ENTRY

* First PDB entry is for command name
  CLC PDBNOENT,=H'1' ANY COMMAND PARAMETERS ENTERED?
  BNH ............... NO, GO HANDLE THIS SITUATION

* Process 1st parameter after command name
  LA R0,PDBENTRY-PDBENTRY GET LENGTH OF PDB ENTRY
  AR R4,R0  BUMP PAST COMMAND NAME ENTRY
  CLI PDBLENG,0 WAS ONLY A DELIMITER SPECIFIED?
  BE ............... YES, GO HANDLE THIS SITUATION
  L R2,CWBBUF  GET ADDRESS OF COMMAND BUFFER
  AH R2,PDBDISP  POINT TO PARM IN BUFFER
  SLR R1,R1  CLEAR LENGTH REGISTER
  IC R1,PDBLENG  GET LENGTH OF PARM

  ...... application code to process parm ............................

  CLC PDBNOENT,=H'2' MORE COMMAND PARAMETERS ENTERED?
  BNH ............... NO, GO HANDLE THIS SITUATION

* Process 2nd parameter after command name
  LA R0,PDBENTRY-PDBENTRY GET LENGTH OF PDB ENTRY
  AR R4,R0  BUMP TO NEXT ENTRY
  CLI PDBLENG,0 WAS ONLY A DELIMITER SPECIFIED?
  BE ............... YES, GO HANDLE THIS SITUATION
  L R2,CWBBUF  GET ADDRESS OF COMMAND BUFFER
  AH R2,PDBDISP  POINT TO THIS PARM IN BUFFER
  SLR R1,R1  CLEAR LENGTH REGISTER
  IC R1,PDBLENG  GET LENGTH OF PARM

  ...... application code to process parm ............................

* Process nth parameter after command name

Figure 13. Sample Code to Access a Command Buffer

Writing a Full-Screen Command Processor
```
Writing Command Processors

in the PANEL parameter list. If you use separate DSIPSS TYPE=ASYPANEL macros for output and input, operator-input data might be discarded before the input DSIPSS can be processed.

An FSCP can issue the DSIPSS macro to request input and then perform other work before issuing DSIPSS TYPE=PSSWAIT to receive the input. In addition, an FSCP has direct access to operator input and can use the DSIWAT macro to synchronize an operator scenario.

The issuer of the DSIPSS TYPE=ASYPANEL request can use DSIPSS TYPE=PSSWAIT to wait on important NetView event control blocks (ECBs) such as the termination ECB, the solicited POI ECB, the cross-domain ECB, message ECBs, the reset ECB, and the user ASYPANEL ECB. When control is returned from the PSSWAIT, a return code of 56 indicates a NetView ECB has been posted. If you expect the action of your input to be short (for example, QUIT), first check your ASYPANEL ECB. The value of the post indicates the status of the DSIPSS TYPE=ASYPANEL request. See the post codes on page 228.

Screen Formatting for the 3270 Data Stream

Because the FSCP is responsible for the 3270 data stream, the processor issues the DSIPSS macro with TYPE=SCRSIZE to find the presentation space dimensions. If the result is larger than 24 by 80 characters, the processor can use the 3270 Erase/Write Alternate command. Otherwise, it must use the Erase/Write command.

When the processor issues DSIPSS with TYPE=SCRSIZE for a terminal that uses 14-bit or 16-bit addressing and query support (indicated in the logmode definition), the returned and actual screen size can be larger than the alternate screen size. If so, when you use the Erase/Write Alternate command to address the parts of the screen outside the alternate screen size, a terminal program checks the results. To avoid this problem, do either of the following:

- Use a 24 x 80 character screen image data stream and use the Erase/Write command instead of the Erase/Write Alternate command.
- Use a Write Structured Field command to create a partition-structured field that controls the buffering in the terminal.

Note: For more information about the 3270 data stream and buffering, refer to the IBM 3270 library.

Requesting Input

You do not need to send READ instructions to the terminal. A READ MODIFIED command is set up and executed whenever your operator uses an AID key. To receive the data, specify an input buffer and an input ECB on a DSIPSS TYPE=ASYPANEL request. When asking for input, check that the operator’s keyboard is unlocked (set bit 6 of WCC to ’1’B). Do this with the same DSIPSS invocation that requests the input or with an earlier one. You can also choose to reset the modified data tags. After requesting input, do not request input on a further DSIPSS TYPE=ASYPANEL request until your ECB is posted.

Posting an ECB

Do not free the storage where your input buffer and ECB reside until the ECB is posted. When necessary, you can force the ECB to be posted early by issuing DSIPSS TYPE=CANCEL. Be aware that after you issue DSIPSS TYPE=CANCEL, the operator cannot use the terminal until another input request is made or until a DSIPSS TYPE=OUTPUT request restores the command facility panel.
When an FSCP sends a full screen of data to the display terminal, the system reads the 3270 data stream into the buffer area. The FSCP can then write more data to the screen while the operator is viewing or entering data. However, avoid writing over or erasing the operator’s input areas.

When the data is read, the NetView program posts an ECB. The command processor processes the input and, optionally, presents more full-screen panels. While the FSCP has a read outstanding, input to the terminal is treated as input to the command processor, not to the NetView program.

When the command processor is called, it reads and writes to the terminal using DSIPSS TYPE=ASYPANEL. The PANEL parameter of DSIPSS points to a 20-byte parameter list. See "DSIPSS: Presentation Services" on page 222 for details about this process.

Waiting for Lists of Events
The DSIPSS macro with TYPE=PSSWAIT enables the FSCP to wait for both its own list of events and another list of events, such as important messages. The FSCP can interrupt its own processing to receive these lists of events. After waiting, the command processor tests the return code to determine if its ECB was posted or if a NetView ECB was posted. If the return code shows that a NetView event is completed, the command can return to NetView to enable the processing of the event. If the panel ECB is posted, the FSCP processes the input in the buffer. In this way, the command processor has complete control of the screen format. The command processor returns to NetView after saving the screen status to enable future processing. The FSCP can specify a RESUME routine (using DSIPUSH) to enable the full-screen presentation to resume.

The reshow option permits an operator to retain previously saved screen data. You can suspend and resume full-screen processing by using macros DSIFIND, DSIPOP, and DSIPUSH and by specifying a RESUME routine.

Testing for Posted Events
The DSIPSS macro with TYPE=TESTWAIT helps the command processor to test whether or not a NetView event is already posted. You can use this option before issuing DSIPSS TYPE=ASYPANEL to avoid performing input or output processing when a NetView event is already posted. This option enables early detection of interruptions. It also enables you to return to the NetView program with a minimum of screen interruptions. For more information about DSIPSS, see the full description of the macro and its attributes in "DSIPSS: Presentation Services" on page 222.

Logging Full-Screen Input and Output
The NetView program does not automatically log full-screen input and output. Use the DSIWLS macro to log pertinent data.

Accommodating Responses and Interruptions
When you write a panel to the terminal, provide operator response by specifying an ECB address and READ buffer with at least one of your output requests. This provides your program with all of the input from the terminal.

Note: The NetView program treats power-off and power-on procedures and the attention signal as error conditions. For powering off and powering on, your ECB is posted for a permanent error and your OST is placed in termination status. The ECB is not posted until the NetView program is notified that the terminal is powered off. The signal that results from the Attention key causes the NetView program to set TVBRESET. Your PSSWAIT also ends.
Writing Command Processors

The program command processor responds to PF3 by terminating your FSCP. If you want your FSCP to make a temporary exit under other conditions, you must have previously prepared for resumption using DSIPUSH.

You do not need to use TYPE=PSSWAIT when you do not want interruptions, such as messages and their resulting automation, or to process cross-domain commands. The command processor can wait on its own list of ECBs. Even if you choose not to wait on other NetView events, include the OST termination ECB in the list. This ECB is located in the TVBTECB field of control block TVB. TVBTECB enables the command processor to be aware of any major condition requiring the command processor to clean up and exit. Use only the ECBLIST parameter with TYPE=PSSWAIT in DSIPSS.

Changing FSCP Characteristics

The DSIPSS macro with TYPE=CANCEL makes it possible for you to change characteristics of the FSCP. These characteristics include the input area length and the ECB address. You can issue TYPE=CANCEL when a DSIPSS TYPE=ASYPANEL is active or inactive. You can also issue TYPE=CANCEL if input from TYPE=ASYPANEL is posted as complete or is not yet complete.

However, you should issue DSIPSS TYPE=CANCEL only when a program is going to free up or change the storage areas used for output, input, ECB or parmlist for a previously issued DSIPSS TYPE(ASYPANEL). Usually, this does not occur until the program is going to end. If this FSCP is long running, “ending” includes issuing a DSIPOP and returning to NetView. Issuing further CANCELS may result in operator input being discarded between the CANCEL and the next TYPE(ASYPANEL) or TYPE(OUTPUT).

Note: If a program must run on NetView V3 or earlier, the program should run MVTVER to determine which version is used. For versions of NetView prior to V3, DSIPSS TYPE(CANCEL) is required anytime FSCP returned to NetView.

You issue TYPE=CANCEL because there is no way to guarantee that the operator will enter data on any given panel. If an active ASYPANEL input request is canceled, the system posts the ECB with a special post code. See page 228 for ECB post codes. The storage where the ASYPANEL ECB is located must not be freed until a DSIPSS TYPE=CANCEL is issued or the NetView program has posted ASYPANEL ECB for successful or unsuccessful input.

Writing a Long-Running Command Processor

Long-running command (LRC) processors use macros DSIPUSH, DSIFIND, and DSIPOP to synchronize the order in which functions are processed so that asynchronous events run in sequence or in parallel. An LRC processor synchronizes functions so that the programs it initiates complete before it ends and its callers do not resume until after it ends.

DSIPUSH identifies three routines that provide for command resumption, and recovery and termination. These routines are the RESUME routine, the abend reinstate routine, and the LOGOFF routine. DSIFIND locates the storage you associated with the DSIPUSH input name. DSIPOP indicates that a long-running command has completed.

When one of these routines receives control, CWBBUF is set to 0 and register 0 contains the storage pointer associated with the long-running command (0 if no storage). Additionally, one of the following occurs:
For a RESUME routine, the TVBRESUM bit is set on.
For an abend reinstate routine, the TVBABEND bit is set on.
For a LOGOFF routine, the TVBLOGOF bit is set on.

Notes:
1. The flags TVBRESUM, TVBABEND, and TVBLOGOF are meaningful only when your input CWBBUF address is 0.
2. HLL command processors cannot be pushed (with DSIPUSH) as ABEND, LOGOFF, or RESUME routines.

The other registers contain the information described in “Input to the Command Processor” on page 63.

When a RESUME, abend reinstate, or LOGOFF routine returns control to the NetView program, all registers must be restored. You need to set register 15 to tell NetView what action to take, as described in the following sections.

RESUME Routines
Before invoking any subordinate command processors or command lists, and before issuing DSIPSS TYPE=FSSWAIT (or TESTWAIT), the LRC processor schedules a RESUME routine (using DSIPUSH). The RESUME routine suspends any other active LRC processors and enables the LRC processor to regain control.

The first request at the top of the long-running command chain defines the controlling RESUME routine. If you use DSIPUSH while another RESUME routine is in control, the new RESUME routine becomes the controlling routine. All other RESUME routines are temporarily suspended.

The suspension period depends on the environment from which the long-running command received control. If the long-running command was called because of an asynchronous event, such as an operator command or the automation of a message, then the NetView ROLL command, if issued, can move (rotate) the long-running command to the bottom of the long-running command chain. This gives control to the next long-running command. If the long-running command received control by direct call from another long-running command, the two commands are regarded as being related by that direct call. This includes direct commands from NetView command list language, REXX, and high-level language command procedures. The ROLL command, if issued, acts against both (or all) such related commands as a group, moving them altogether and preserving their order. In either case, DSIPOP can remove the topmost RESUME routine, giving control to the next long-running command.

Note: Neither the ROLL command nor DSIPOP causes an asynchronous interrupt. A command gives up control only by returning to its caller, except for the action of immediate commands.

You can also use DSIPOP to remove (by name) a RESUME routine that is not at the top of the stack of the long-running command chain. This action is regarded as a cancellation of that long-running command unless your DSIPOP invocation specifies COMPCDE. If the long-running command that was removed was part of a larger group, the calling long-running command is given control, as soon as the current process permits, and is given a cancel indication, as follows:
- NetView command list language command lists are stopped
- REXX command lists receive a HALT
- All others receive a minus 5 return code
Note: Refer to Tivoli NetView for z/OS Customization: Using PL/I and C for more information about cancellable and noncancellable high-level language command procedures.

Long-Running Commands
All command procedures are long-running commands. A command procedure’s RESUME routine blocks RESUME routines pushed earlier, exactly like other LRC processors. A pause or wait state for a command procedure is no exception.

DSIPUSH for a RESUME routine is either major or minor. A major DSIPUSH places the new RESUME routine at the top of the LRC processor stack, suspending previously issued long-running commands. A major DSIPUSH is used with NetView command list language processors. A minor DSIPUSH places the new RESUME routine on the stack after any leading command procedures (in the same group), and enables the leading command procedures to complete before the new RESUME routine gains control. Command procedures already suspended by other LRC processors are not affected.

You can use a major DSIPUSH to suspend a command procedure until your LRC processor executes a DSIPOP. You can use a minor DSIPUSH to enable a calling command procedure to complete, after which your RESUME routine gains control. The following list describes completion codes and return codes:

- **Completion codes**
  A long-running command can return a completion code to the long-running command that invoked it (in the same group) by specifying a value for the COMPCDE keyword in the DSIPOP macro. If a long-running command was invoked asynchronously, the value specified for COMPCDE is ignored. The completion code is passed to the calling long-running command in CWBRCODE upon resumption.

- **Return codes**
  A RESUME routine can return control to its caller many times before it completes, to provide messages, queued commands, called long-running commands, and other asynchronous work to process. Upon the initial return (when commanded, CWBBUF=0), the value in register 15 is ignored. After each resumption, the value in register 15 conveys the long-running command requirements for STIFLE. For an explanation of STIFLE, see “Message STIFLE” on page 74. Register 15 = -8 requests stifle; register 15 = +8 requests no stifle. Meanings for other return codes are reserved. For compatibility with prior releases, a zero (0) return code is valid after DSIPOP is issued to remove the long-running command from the stack.

An important part of any RESUME routine function is screen control. Because the state of the operator’s terminal is not known (see “Screen Identifier” on page 76) on entry, the RESUME routine must ensure that the operator is not locked out by a panel left over from a previous LRC processor. Ensuring this situation can mean issuing a message (DSIPSS TYPE=FLASH) to guarantee that the command facility panel and command line are available to the operator. Ensuring this situation can also mean displaying a full-screen panel (DSIPSS TYPE=ASYPANEL). The screen control requirement means that you cannot use the NetView-supplied routine DSILRCR8 as a RESUME routine with the NetView program, as was sometimes appropriate with NCCF. You can use DSILRCR8 as an abend reinstate or LOGOFF routine if no cleanup besides DSIPOP is needed.
Issuing Messages Upon Resumption

Note: When you use messages for screen control, issuing a message on every resumption is excessive and can cause looping if the message is automated or routed. Use the screen serial number (TIBSCRSN) to determine when to issue messages upon resumption. Be especially cautious when issuing VTAM messages upon resumptions under the PPT, because all messages are routed and can generate new activity under the PPT. This new activity will disrupt VTAM message reception that the NetView program might be performing.

To assist RESUME routines that display panels, the NetView program provides status information through the TIBSCRID field (see page 74).

A completed RESUME routine (one that has issued DSIPOP against itself) need not be concerned with screen control because the following RESUME routine assumes responsibility. Ending messages (associated with NCCF) that are issued when an LRC processor finishes are not appropriate in the NetView program.

NetView enables operators to recover from operator errors and certain program errors through the use of the attention signal or the RESET (NORMAL) command. When attention is signaled or the RESET command executes a flag, TVBRESET is set, and an ECB, TVBRESET, is posted. All commands, and especially LRC processors, can test TVBRESET regularly. Whenever it is set, the command ends its processing (using DSIPOP, if appropriate) and returns to the NetView program.

The following steps illustrate the use of macros DSIPUSH and DSIPOP:

1. A command list invokes a command and, to complete the request, the command processor requests data from a DST.
2. The command processor issues DSIPUSH specifying a RESUME routine. At this point, the command processor becomes a long-running command.
3. The command processor uses the DSIMQS macro to queue a buffer with IFRCODE set to IFRCODCR containing its request for data to the DST.
4. The command processor returns to its caller. The terminal is left in the state it was in when the command list was running. In this case, the command facility panel is displayed.
5. This operator’s task is idle (the command list is suspended by DSIPUSH); therefore, the RESUME routine defined earlier by DSIPUSH is immediately called.
6. The command processor finds that CWBBUF=0 (no command is being passed) and TVBRESUM is set, but TIBLRCNP is not set. The command processor returns control to its caller.
7. A message is received and displayed. The command processor is called again as a RESUME routine.
8. The operator issues a full-screen command, which issues its own DSIPUSH and waits for input.
9. The operator exits (or rolls away from) this latter command processor.
10. The panel of the second command processor is left in place, and the original command processor RESUME routine is called. This time TIBLRCNP is set. The command processor issues a FLASH message: STILL WAITING FOR DATA. The command facility panel is restored by DSIPSS.
11. An IFRCODCR buffer containing the DST reply is received. After issuing the DSIFIND macro, the IFRCODCR command processor places data in the LRC
Writing Command Processors

The command processor's long-running command storage (as defined in the original DSIPUSH). The IFRCODCR command processor only saves the data because another long-running command can be in control at this point.

12. The command processor is resumed again. Finding its data request satisfied, it completes its function and issues DSIPOP against itself, using the COMPCDE keyword on DSIPOP to indicate the nature of the completion. The command processor returns a return code of 8 in register 15.

Note: For compatibility with prior releases of the NetView program, a zero (0) return code is acceptable after DSIPOP is issued to remove the long-running command from the stack.

13. The NetView program calls the next RESUME routine, the command list invoking the original command, which then continues.

14. The command list receives the return code (RC in REXX or &RETCODE in NetView command list language) that was specified for COMPCDE on DSIPOP.

You can set a RESUME routine to return control to the NetView program without giving up control of the operator's display. For example, the screen can be dynamically updated based on information sent by another task in the form of an IFRCODCR message. (See "DSIIFR: Internal Function Request" on page 125.) To assist with such a function, the NetView program provides two tools: message STIFLE and a screen identifier.

Message STIFLE

A RESUME routine can request a message STIFLE when it returns control to the NetView program, indicating that ordinary line mode messages are not displayed and the operator's screen is not disturbed by the processing of the messages.

Some messages are displayed by the NetView program whether or not STIFLE is in effect. These messages are said to break the stifle mode. The following messages can break the stifle mode:

• Action messages with ISTnnnA or DSInnnA identifiers.
• Messages that request a reply (HDRTYPEY, except the ASSIGN=COPY of HDRTYPEY that does not break STIFLE).
• Any message issued with DSIPSS TYPE=FLASH. The intended use of DSIPSS TYPE=FLASH is for command echoes and screen control messages.

If STIFLE is broken, it remains off until re-invoked by a RESUME routine.

A stifle request can be honored only while the network log or the hardcopy log remains active. The NetView program counts messages that are stifled and displays message DSI593I to remind the operator that it is necessary to consult the network log or hardcopy log to see these messages. A stifle request is not honored while the command facility panel is in place. The request is not honored because the NetView program assumes that the LRC processor gains control of the panel through the use of DSIPSS TYPE=ASYPANEL before requesting STIFLE.

STIFLE affects only line-mode messages. A full-screen display is not affected. If, for example, the result of a START DOMAIN command (the logon panel) is delayed long enough for an LRC processor to gain control, the returning logon panel is displayed without regard to the STIFLE.
ROLL Function
A roll group is the NetView program’s way of enabling the operator to switch from one component, such as the hardware monitor, to another component, such as the session monitor, and return to the place at which the operator last left the component. This procedure is similar to window processing in other applications.

A roll group is a set of related DSIPUSH macro requests. DSIPUSH begins a new roll group when it is invoked from an asynchronous command environment. Operator commands, commands generated by automation, and commands scheduled using DSIMQS are asynchronous. A command called directly from another long-running command is synchronous. The synchronous long-running command is added to the roll group started by the asynchronous long-running command and blocks it until a DSIPOP request is issued against the synchronous long-running command. The current roll group is defined as the roll group that is first on the long-running command chain.

The ROLL command treats each specified roll group as a unit when manipulating the chain. The ROLL command moves the topmost roll group to the bottom of the stack. All elements within the roll group maintain their position within the group.

Note: For additional information about using the ROLL command, refer to the NetView online help.

Roll Group Usage
The simplest roll group is a command that invokes DSIPUSH with a RESUME routine. The RESUME routine enables the command processor to respond to a ROLL request. If this command accepts command input from the operator (with DSIPSS TYPE=ASYPANEL), a ROLL command causes the long-running command to move (rotate) to the bottom of the long-running command chain. When the ROLL command rotates another long-running command similarly to the bottom of the long-running command stack (or the long-running command ends with DSIPOP), the RESUME routine regains control.

You can use multiple DSIPUSH requests with different RESUME routines or different storage pointers for such procedures as implementing a hierarchical panel structure. By scrolling forward and then backward, the operator returns to the same panel that was previously displayed. When a panel ends (that is, the operator uses PF3, and the program invokes DSIPOP), the operator returns to the panel within the hierarchy, that is, above the current panel.

The following steps describe how a full-screen function makes use of the ROLL capability:
1. Issue DSIPUSH for a RESUME routine.
2. Provide a line on your panels for NetView command input, using 3270 data stream orders.
3. When the operator enters data on the command line, the input data stream contains orders that identify the area of the panel in which the operator typed.
4. Use the DSICES macro to verify the command. When command entry is detected, verify the command, build a standard NetView buffer with HDRMTYPE=HDRTYPEOP, and issue the DSIMQS macro to send the buffer to the operator’s own task, using TVBOPID as the destination of the DSIMQS.

Note: You must translate the command input to uppercase if the language the operator is using has uppercase and lowercase characters.
5. Return to NetView to enable the command to be processed.
6. The NetView program re-invokes your RESUME routine when the command has completed processing and when your RESUME routine is the first routine on the stack or the current roll group.

   If the command entered was ROLL, the ROLL command processor automatically switches your roll group to last.

   If the command entered establishes a new roll group, your roll group is pushed down on the stack and the new group becomes current. When the operator exits from the new roll group, your roll group is invoked by NetView calling the topmost RESUME routine.

7. You can also detect PF6 and PF18 as ROLL. You build the command buffer, but you need to look up the command name for the DSIROLL load module using the DSICES macro and then issue DSIMQS to queue the buffer.

8. To enable the operator to switch directly to your function from a different roll group without ROLL, you can define a command as a reshow request. It can be your function’s command name with no operands provided.

   When your command processor is entered (with no operands) with reshow requested, issue DSIPUSH with PROMOTE=YES to move your roll group to the top of the stack. Proceed by refreshing the screen from the last panel the operator viewed.

   Note: Issuing DSIPUSH with PROMOTE=YES exchanges the storage address in the DSIPUSH parameter list with the one already associated with the named request, and returns the old value in register 0. Therefore, you can issue DSIFIND to determine the current address and specify it on the PROMOTE=YES request to make sure the address stays the same. If you do not specify the address, the 0 value in the parameter list replaces the current value.

Screen Identifier

Requesting STIFLE does not guarantee that an LRC processor’s panel is not modified, but the NetView program provides a way to determine whether modifications occurred. After writing the panel to the screen, an LRC processor saves the value of TIBSCRID. Upon regaining control, a RESUME routine can compare the present and saved values of TIBSCRID to determine whether—and to some extent, what type of—screen modifications occurred since it last had control.

TIBSCRID, a 4-byte field, consists of two subfields:

   • TIBSCRSN
     Specifies the low-order 3 bytes that form a serial number for the screen’s contents. This number is incremented whenever anything is sent to the screen that changes what the operator sees.

   • TIBSCRM
     Specifies the high-order byte that is a state change indicator. A change in TIBSCRM usually means a DSIPSS TYPE=CANCEL request was issued. A change can also mean that a Lock Keyboard or other non-data 3270 command was sent to the terminal.

When an LRC processor regains control and TIBSCRID is unchanged, the LRC processor can resume processing as if it had never lost control.

When TIBSCRM (and not TIBSCRSN) is changed, the LRC processor reestablishes its read by issuing DSIPSS TYPE=ASYPANEL to send a Write and Unlock
command (XF182') to the terminal and re-specify the ECB, if any, by which the long-running command waits for input. This procedure makes refreshing the screen unnecessary.

When TIBSCRSN is changed, some visible modification to the LRC processor’s panel is made. You must rewrite the entire panel.

**Abend Reinstate Routines**

An abend reinstate routine performs a required recovery action, such as freeing control blocks, after the NetView program recovers from a task’s abnormal end (abend). You cannot use abend routines while running under the DST because DSTs are not reinstated after abends. While running under the DST, use a LOGOFF routine instead.

The abend routine assesses the damage caused by an abend and either keeps or cancels the long-running command. Each command in the stack must be associated with an abend reinstate routine. With its return codes in register 15, the abend routine notifies the task whether the command is to be kept or removed from the queue and freed, as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Keep the long-running command request queued.</td>
</tr>
<tr>
<td>8</td>
<td>Remove the currently queued long-running command request from the queue.</td>
</tr>
</tbody>
</table>

If the routine keeps a long-running command, the RESUME routine runs the first time the task has no other work to perform. All stacked long-running command routines are maintained in their current order. Abend reinstate routines cannot issue macros DSIPOP or DSIPUSH.

When a task recovers from an abend, all abend reinstate routines are called, starting with the top one in the stack, which is the most recent. When the abend reinstate routine returns to its task, it specifies whether the associated command is to be left on the stack or removed.

**LOGOFF Routines**

A LOGOFF routine gives the command processor control before the task ends. The task is not reinstated, and the command processor can perform any final cleanup processing, such as closing a data set or freeing storage. Each command in the stack must contain a LOGOFF routine.

A LOGOFF routine is called sequentially for each request on the queue. LOGOFF routines cannot issue macros DSIPOP or DSIPUSH.

When the command processor returns to the task, requests are taken off the queue and freed.

NetView ignores all return codes for LOGOFF routines.

When a task ends, all the LOGOFF routines are called starting with the top, or most recent, request on the stack. Each request is removed from the queue and freed.
Automation Task Command Processors

When you write commands to run under OSTs, consider the effects of running under automation tasks and MVS console tasks. These OSTs have the TVBAUTOO bit set to 1, indicating that immediate commands and full-screen mode commands are not supported in this task.

Installing a Command Processor

**Note:** To install a command processor, define the command verbs with CMDMDL statements as described in *Tivoli NetView for z/OS Installation: Configuring Additional Components*. The command processor can also be added dynamically using the ADDCMD command. For more information on ADDCMD, refer to *Tivoli NetView for z/OS Command Reference*.

If your command processor supports line-mode output (DSIPSS TYPE=OUTPUT), you can specify ECHO=Y. If your command processor supports full-screen mode output (DSIPSS TYPE=ASYPANEL), you can specify ECHO=N. Then assemble and link-edit the command processor into a load module in the NetView load library. The NetView program loads and calls the command processor according to its linkage editor attributes.

See “Testing Your Program” on page 3 for information about testing your command processor before using it.

Template for a Command Processor

*Figure 14 on page 79* illustrates entry and exit processing required by all command processors. This template, a part of the NetView sample library, is member CNMS4202 of the CNMSAMP data set.
Figure 14. Example Template for a Command Processor (Part 1 of 3)
**Writing Command Processors**

```
***********************************************************************
* SAVE REGISTERS AND ESTABLISH BASE REGISTER *
***********************************************************************
USING *,R15
B PROLOG
DC C'ATMPCMDP &SYSDATE. AT &SYSTIME.'
PROLOG DS 0H
STM R14,R12,12(R13) SAVE REGISTERS
DROP R15
LR R12,R15 SET BASE REGISTER
USING ATMPCMDP,R12
***********************************************************************
* ESTABLISH ADDRESSABILITY TO THE COMMAND WORK BLOCK (CWB) AND SET *
* UP THE SAVE AREA USING CWBSAVEA *
***********************************************************************
LR R11,R1 LOAD CWB ADDR
USING DSICWB,R11 R11 BASE FOR COMMAND WORK BLOCK
LA R1,CWBSAVEA USE CWBSAVEA FOR SAVEAREA
ST R1,8(R13) STORE MY SA INTO CALLERS SA
ST R13,4(R1) STORE CALLERS SA IN MINE
LR R13,R1 R13 HAS MY SAVEAREA ADDRESS
***********************************************************************
* ESTABLISH ADDRESSABILITY TO THE TASK INFORMATION BLOCK (TIB), THE *
* TASK VECTOR BLOCK (TVB), AND THE MAIN VECTOR BLOCK (MVT). *
***********************************************************************
L R10,CWBTIB GET DSITIB ADDRESS
USING DSITIB,R10 ESTABLISH ADDRESSABILITY
L R9,TIBTVB GET DSITVB ADDRESS
USING DSITVB,R9 ESTABLISH ADDRESSABILITY
L R8,TVBMVT GET DSIMVT ADDRESS
USING DSIMVT,R8 ESTABLISH ADDRESSABILITY
* XC CWBADATD,CWBADATD ZERO AUTODATA AREA
SLR R15,R15 ZERO RETURN CODE REGISTER
***********************************************************************
* MAIN PROCESSING GOES HERE *
***********************************************************************
```

Figure 14. Example Template for a Command Processor (Part 2 of 3)
Figure 14. Example Template for a Command Processor (Part 3 of 3)
Chapter 5. Writing User Subtasks

This chapter describes user subtasks and illustrates how to write, process, and install them.

Types of User Subtasks

NetView provides two methods for writing user subtasks. The first and recommended method uses the data services task (DST) interface as a coding base. The DST base provides interfaces for the following functions:

- An initialization installation exit
- A subtask processing module (DSIZDST)
- CNMI service
- VSAM service
- Data services command processor (DSCP)

The DST provides an ideal structure for user-written tasks because the DST can be defined with VSAM services, CNM services, or neither service. You can implement user-defined functions within the data services command processors. The DST provides all the low-level user subtask functions that you would otherwise need to code if you wrote a complete optional subtask (OPT). Write an optional subtask only if access to the subtask event control block (ECB) processing loop is required.

The other method requires that you code an OPT. With this method, NetView supplies an intertask communication (message queue) ECB and a termination ECB. You must provide an appropriate ECB processing loop and any additional function. This requires more coding than the first method, but it offers more flexibility in the kinds of functions that you can implement.

Optional Subtask Processing Overview

A user subtask requires the following processes:

Installation

Use CNMSTYLE to define an optional subtask to NetView. CNMSTYLE is processed during NetView initialization.

Initialization

The initialization process of the subtask performs any required initialization functions. Examples include acquiring NetView control blocks using the DSILCS macro and acquiring dynamic storage using the DSIGET macro.

Processing

The processing part of a subtask should begin by invoking the DSIWAT macro to wait on an ECB list. The ECB list must include the subtask termination ECB (TVBTECB) and generally includes the message queue ECB (TVBMECB), which is used for intertask communication (using the DSIMQS macro). You can also include user-defined ECBs in the ECB list.

Termination

As shown in Figure 15 on page 84, the termination process frees all acquired resources (for example, storage and NetView control blocks) and returns to NetView.
Writing User Subtasks

Installation of User Subtasks

Code a TASK statement for the subtask in the CNMSTYLE member of the DSIPARM data set. The TASK statement defines the task to NetView and provides information through the following keywords:

**MOD** Specifies the name of the module to be run as a subtask. You can link-edit the module into the proper NetView library.

**MEM** Specifies the user-defined initialization member found in DSIPARM to be used by this task. You are responsible for the format and contents of the specified member. The member can be read and processed during the task initialization.

**PRI** Specifies the relative task priority (1–9). 1 is the highest task priority that you can assign, and 9 is the lowest.

Figure 15. Subtask Organization
INIT  Specifies whether the task is to be started during NetView initialization (INIT=Y) or using the START command only (INIT=N).

An example of a TASK statement is:

```
TASK.USERTASK.MOD=USERMOD
TASK.USERTASK.MEM=USERMEM
TASK.USERTASK.PRI=7
TASK.USERTASK.INIT=Yes
```

In the preceding example, the subtask identification is USERTASK. USERMEM is the name of the DSIPARM member. The priority of the subtask is 7 and is started during NetView initialization.

For more information about the TASK statement, refer to the Tivoli NetView for z/OS Administration Reference.

Initialization of the OPT

Issuing the START TASK command or specifying INIT=Y on the TASK definition statement attaches the optional subtask (OPT) normally. The TVBTERM bit in the TVB is set off (0).

When an OPT is attached, the registers contain the following information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The address of the task vector block (TVB)</td>
</tr>
<tr>
<td>13</td>
<td>The address of a standard 72-byte save area used to store the caller’s registers</td>
</tr>
<tr>
<td>14</td>
<td>The return address</td>
</tr>
<tr>
<td>15</td>
<td>The entry address of the subtask</td>
</tr>
<tr>
<td>0, 2–12</td>
<td>Unspecified</td>
</tr>
</tbody>
</table>

The control blocks used in attaching an OPT are:
- Task vector block (TVB)
- Task information block (TIB)
- Main vector table (MVT)
- Service routing vector list (SVL)

The TVB contains the address of the TIB and the MVT, and the task control block of the operating system. The MVT contains the address of the SVL. See Chapter 7 Control Blocks for detailed descriptions of these control blocks.

After the subtask is initialized, but before it starts processing, you can indicate that the subtask is ready to begin processing by doing the following:
1. Ensure that the subtask is not already on the TVB chain.
2. Enqueue the TVB chain.
3. Set the TVBOPID field of the TVB to a unique subtask identifier.
4. Set the TVBACTV bit on.
5. Dequeue the TVB chain.
6. Specify an end to task initialization by issuing a message. Use DSIPSS TYPE=OUTPUT for this message. The message will be returned to the appropriate operator and process (for example a pipeline) that issued a START command for your task, otherwise to the authorized receiver. This message signals an end to correlation for the START command.
Writing User Subtasks

One method of setting the TVBOPID field is to copy the contents of TVBLUNAM into TVBOPID. (TVBLUNAM is the value of the TSKID operand in the TASK definition statement.) Another method is to use a predefined value.

If TVB374CT is turned on during subtask initialization, NetView maintains a counter of the number of buffers on the public message queues. If this counter exceeds the threshold value defined in the NetView constants module, DSICTMOD (CNMS0055), message DSI374A is issued. TVB374CT should be set just before TVBACTV is turned on if you desire public message queue thresholding. TVB374CT turns off in the main task during subtask termination.

Note: Refer to the Tivoli NetView for z/OS Installation: Getting Started for more information about the NetView constants module (DSICTMOD).

TVBMEMNM Field

The value of the MEM keyword of the TASK definition statement is used as the name (1 to 8 characters) of an initialization member or file. You find the value of the MEM keyword in the TVBMEMNM field of the TVB if initialization input is required. If the subtask is coded to process this member or file, the following macros are invoked to read from the member or file:

- DSIDKS TYPE=CONN,NAME=DSIPARM to connect the subtask to disk services for the DSIPARM data set.
- DSIDKS TYPE=FIND,NAME=TVBMEMNM to find the member or file and read the first record.
- DSIDKS TYPE=READ to read a record. The READ is repeated until a user-defined END statement is read or until an end-of-file return code is returned.
- DSIDKS TYPE=DISC to disconnect from disk services.

If the subtask does not use the initialization member or filename, you can use TVBMEMNM for other purposes, depending on the manner in which you specify the MEM keyword of the TASK statement. For example, you can decide to use this field as the data definition (DD) name to be opened by the subtask, or you can specify a default operator to receive messages.

Processing

The following sections discuss processing subtasks using an ECB loop, intertask communication, and operator communications.

ECB Loop

The processing section of a subtask usually begins by issuing the DSIWAT macro to wait on an ECB list. The ECB list contains the subtask termination ECB (TVBTECB), and generally contains the message queue ECB (TVBMECB), used for intertask communication using the DSIMQS (message queuing service) macro. In addition to these two NetView-provided ECBS, the ECB list can contain user-defined ECBS for additional functions.

Intertask Communication

If your subtask receives messages or commands from other tasks (or exits), include the normal message ECB (TVBMECB) in your subtask’s wait list. Optionally, you can include the high-priority and low-priority message ECBS (TVBMECBH and TVBMECBL). If the subtask services the high and low queues, set the bit TVBMM to B’1’ to indicate this. (When TVBMM is B’0’, the message queuing service puts all messages on the normal queue, regardless of how they are sent.) Retest the
high-priority ECB after each item of work on a lower queue, to enable higher-priority work to preempt the queue of lower-priority work.

Each message queue has three parts: a private queue, a public queue, and an ECB. (Each has two queues for reentrant code.) The elements and their respective priorities are shown in Table 10.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Private Queue</th>
<th>Public Queue</th>
<th>ECB</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>TVBMPRQH</td>
<td>TVBMPUBH</td>
<td>TVBMECBH</td>
</tr>
<tr>
<td>Normal</td>
<td>TVBMPRIQ</td>
<td>TVBMPUBQ</td>
<td>TVBMECB</td>
</tr>
<tr>
<td>Low</td>
<td>TVBMPRQL</td>
<td>TVBMPUBL</td>
<td>TVBMECBL</td>
</tr>
</tbody>
</table>

**Message Queue Processing:** When your subtask receives a message buffer, the TVBMECB is posted and the message buffer is inserted at the head of the public message queue pointed to by TVBMPUBQ, a last in, first out (LIFO) queue. When your subtask detects that the TVBMECB ECB has been posted, the public message queue can be moved to the private message queue (TVBMPRIQ) for processing. Because the DSIMQS service handles situations such as mainline interruption (for example, an exit running asynchronously queues a message buffer to your subtask), simultaneous processing in multiple subtasks, and parallel processing in multiprocessor environments, you must use the assembler compare and swap (CS) instruction to acquire message buffers from the public message queue.

To process the public message queue, do the following:

- Set TVBMECB to 0.
- Use the assembler CS instruction to obtain the queue of buffers from TVBMPUBQ and store 0 in TVBMPUBQ. If public message queue thresholding is being done (TVB374CT is on), you can add code here to set TVBQCNT to 0 using the assembler CS instruction.
- Reverse the order of the queue, that is, make it first in, first out (FIFO) so that the message buffers can be processed in the order they were actually received.

The following segment of assembler code demonstrates how to move the public message queue to the private message queue. Addressability to the DSITVB control block is assumed.

```
MESSAGEQ EQU *          BRANCH HERE WHEN TVBMECB POSTED
XC    TVBMECB,TVBMECB    CLEAR MESSAGE ECB
CHEQ  EQU *              CLEAR SWAP REGISTER
  SLR  R0,R0
  L     R3,TVBMPUBQ      LOAD COMPARAND REGISTER
  CS    R3,R0,TVBMPUBQ   CS ZERO ON THE PUBLIC QUEUE
  BNE   CHEQ        RETRY IF TVBMPUBQ CHANGED
  LTR   R3,R3          IS QUEUE EMPTY
   BZ    WAIT          BR IF EMPTY
   USING BUFHDR,R3     MAP BUFHDR ONTO QUEUE HEAD
 REQ   EQU *              MAKE LIFO QUEUE A FIFO QUEUE
   L     R1,HDRNEXTM    R1 = POINTER TO NEXT BUFFER
   ST    R0,HDRNEXTM    SET NEXT TO PREVIOUS
   LR    R0,R3          MAKE CURRENT PREVIOUS
   LTR   R3,R1          END OF QUEUE?
   BNZ   REQ           CONTINUE UNTIL END REACHED
   ST    R0,TVBMPRIQ    ANCHOR THE PRIVATE QUEUE
 PROCESS EQU *            BEGIN BUFFER PROCESSING
```
Writing User Subtasks

The message buffers can be dequeued from the private message queue and processed. After each buffer is processed, it is be freed. Message buffers are obtained with DSIGET Q=NO and SUBPOOL 0, so they are be freed with DSIFRE Q=NO and SUBPOOL 0. These are the default values.

**Message Buffer Contents:** Message buffers are described in detail in [Chapter 2: Designing Assembler Modules](#). They can be actual messages to be displayed (HDRMTYPE = HDRTYPEU) or internal function requests (HDRMTYPE = HDRTYPEI). For internal function requests queued to your optional subtask, you can define your own function type by setting the IFRCODE field to IFRCODUS (user function) and then taking appropriate user-defined action when your optional subtask receives the buffer.

**Using IFRCODUS to Invoke a User-Defined Command Processor:** Use the following technique to call a user-defined command processor under your optional subtask:

- Issue DSIMQS (from any subtask environment) to send an internal function request (HDRMTYPE=HDRTYPEI and IFRCODE=IFRCODUS) to your optional subtask. The command name and command parameters follow the DSIIFR portion of the buffer passed to DSIMQS.
- When processing the IFRCODUS message buffer under your subtask, add 2 to HDRTDISP to adjust the displacement to the start of the command and subtract 2 from HDRMLENG to keep the length consistent.
- Follow the steps listed under "Calling a Command Directly" on page 17 to call the command processor.

Define the command processor called as TYPE=D or TYPE=RD on its respective CMDMDL statement in DSICMD. It is a user-written command processor. Do not call any NetView-provided command procedures from your optional subtask.

**Note:** The DSIZCSMS and DSIZVSMS macros are not valid under an optional task.

**Sending Message Buffers:** Use the DSIMQS macro to send message buffers to other subtasks. These message buffers can contain actual operator messages to be displayed, or they can contain internal function requests (IFRs) to be executed by the receiving subtasks. See [Chapter 8: Macros](#) for details of the DSIMQS macro.

**Operator Communications**

**Sending Messages to Operators:** You can use DSIPSS TYPE=OUTPUT or TYPE=IMMED to send messages. Messages sent this way go to the owner who started the subtask. If the subtask was started during NetView initialization or the owner has logged off, the message is sent to the PPT for routing and automation.

You can also use the DSIMQS macro to send messages to the authorized receiver of messages or to the operator that started the subtask. The TIBMSGNM field of the DSITIB control block contains zeros if the subtask was started during NetView initialization (that is, if you specified INIT=Y on the TASK definition statement).

**Note:** Commands from operators are buffers with HDRMTYPE equal to HDRTYPEU, HDRTYPEB, or HDRTYPEQC. All other commands are HDRTYPEI.

**Logging Messages:** You can use macro DSIWLS to write messages from the subtask to the network log, the MVS system log, an external log, or a NetView
sequential log. See “DSIWLS: Write Log Services” on page 249 for more information. You cannot start hardcopy logging for user-written subtasks.

**Scheduling Commands:** You can use macro DSIMQS to schedule commands to run under other tasks. See “Scheduling Commands Using DSIMQS” on page 18 for more information.

**Calling Command Processors:** See “Calling a Command Directly” on page 17.

**Notes:**
1. You cannot call command lists, immediate commands, or regular commands from an optional task. Only command processors defined as TYPE=D or TYPE=RD can be called from an optional task.
2. You cannot invoke DST service macros DSIZVSMS and DSIZCSMS under an optional task.

**User-defined Functions:** When a user-defined ECB is posted for work, a user-defined command processor can be called as explained in the previous section, or a subroutine can be called to perform the requested function. The method depends on how you want to handle your implemented function.

**Termination**

**Terminating the OPT**
When an optional subtask (OPT) terminates normally, the TVBTERM bit is set on, indicating that the subtask resources can be released. When a subtask abends, the TVBTERM bit is set on and the subtask is reattached. This is called a cleanup attach. When the subtask regains control, it frees the resources it had obtained and exits normally.

Include the TVBTECB field of TVB (termination ECB) in the subtask ECB list for each OPT you write. When a CLOSE NORMAL command is issued and all operators have logged off, the main task posts the TVBTECB of the subtask. This posting indicates that subtask termination is requested.

When the subtask finds the TVBTECB posted, the subtask performs the following:
1. Releases all resources.
2. Sets the TVBOPID field to blanks.
3. Enqueues the TVB chain.
4. Sets the TVBACTV bit off.
5. Sets the TVBTERM bit on.
6. Dequeues the TVB chain.
7. Reloads the registers originally passed to the subtask and return to the operating system.

After the subtask releases all resources, NetView macros cannot be issued.

**Releasing Queued Storage**
The DSIGET Q=YES option enables storage to be freed for both normal and abnormal subtask termination. To release all queued storage, issue DSIFRE AQ=YES. Be certain that any VTAM access method control blocks (ACBs) owned by this task have been closed before issuing DSIFRE AQ=YES. NetView frees all queued storage with one invocation of DSIFRE AQ=YES (both mainline and exit storage). Some macros may require queued storage; therefore, the subtask may not issue any NetView macros after releasing the queued storage.
Additional Considerations

Special Requirements for IRB Exits
User-written interruption request block (IRB) exits that invoke NetView macro services require the following special processing:

- On entry, if the TVBINXIT is not on, set it on. If the TVBINXIT bit is already set on, increment TIBMUXIT by 1.
- On exit, if TIBMUXIT is zero (0), clear the TVBINXIT bit. However, if TIBMUXIT is greater than 0, decrement it by 1.

Displaying Status
The LIST command displays the status of asubtask on an operator’s terminal. For OPTs, in addition to status, a header line and the contents of TVBOPID and TVBLUNAM are also displayed. Status is determined by the following TVB bit fields in the following order:

1. TVBLGOFF – Stopping
2. TVBACTV – Active
3. TVBLGON – Starting
4. None of the above – Inactive

The subtask can also create a status display.

VTAM Outage Processing
User-written code (exit routines, command processors, and subtasks) that requires VTAM to operate receives error codes when VTAM is inactive. User-written subtasks that require VTAM must enable VTAM to end without terminating or abending NetView. Do this as follows:

- If your subtask opens a VTAM ACB, code a TPEND exit for VTAM that is called when VTAM ends. Your TPEND exit can post TVBTECB to signal subtask termination to begin.
- If your subtask requires VTAM to be active and does not open an ACB, you can still be notified. Set the TVBAUTVE bit in the TVB for your subtask. When the main subtask TPEND is called (for HALT NET, QUICK, for HALT NET, CANCEL, and for VTAM ABEND), the NetView main subtask posts TVBTECB for every subtask that has TVBAUTVE set to 1.
- NetView also provides another bit, TVBAUTVS, which causes the main subtask of NetView to reattach your subtask when NetView detects that VTAM has successfully opened the main subtask’s ACB. Set TVBAUTVS to 1 for this function.

Data Services Task (DST)
A DST is a set of NetView interfaces built on top of the optional task base. NetView provides a subtask processing module (DSIZDST) along with the following:

- An initialization exit interface
- A data services command processor (DSCP) interface that provides the following services:
  - A CNM data services macro interface (DSIZCSMS) to request and send data across the communication network management interface (CNMI)
  - An interface to enable a command processor to receive unsolicited CNM data
  - A VSAM data services macro interface (DSIZVSMS) using PUT and GET to obtain records from a predefined VSAM data set.
- Various installation exit interfaces
Installation of a DST

You can dynamically start data services tasks (DST) without defining them in CNMSTYLE. If you do not want to define data services tasks in CNMSTYLE, you can use the START command with the TASK keyword.

If you choose to define the TASK statement in CNMSTYLE, the statement follows the same format as the optional-task TASK statement, with the following exceptions:

- The MOD keyword must specify DSIZDST as the subtask processing module. DSIZDST, provided by NetView, provides the necessary initialization, processing, and termination routines to use the DSCP interfaces.
- The initialization data set member (specified by the MEM keyword) must contain DSTINIT statements to provide initialization parameters required by DSIZDST. The statements are described in the following sections under their respective interfaces.

The other TASK statement keywords have the same meanings as those coded for an optional task, as follows:

- PRI Specifies the relative task priority (1–9). One is the highest task priority that can be assigned, and 9 is the lowest.
- INIT Specifies whether the task is to be started during NetView initialization (INIT=Y) or using the START command only (INIT=N).

An example of a TASK statement is as follows:

```
TASK.SQLOGTSK.MOD=DSIZDST
TASK.SQLOGTSK.MEM=SQLOGMEM
TASK.SQLOGTSK.PRI=2
TASK.SQLOGTSK.INIT=N
```

In the preceding example, the subtask identification is SQLOGTSK. SQLOGMEM is the name of the DSIPARM member. The priority of the subtask is 2 and is not started during NetView initialization. You can use the START command to start the subtask.

Initialization of a DST

The following keywords apply to DST initialization (DSTINIT) processing.

- FUNCT -Specifies the DST services that are required. In all cases, the ability to call DSCPs is provided. The function choices are:
  - OTHER The DST does not require the CNMI or VSAM interfaces.
  - BOTH Both of the VSAM and CNMI interfaces are required.
  - CNMI Only the CNM interface is required.
  - VSAM Only the VSAM interface is required.

- XITDI -Specifies the name of the user-provided initialization exit. The exit is called with the standard NetView installation exit interface as described in "Chapter 3. Writing Installation Exit Routines" on page 23, and is called once for every statement in the specified initialization member (MEM keyword of TASK statement). When the end of file has been reached, USERPDB and USERMSG are both 0. For each statement (except end-of-file condition), the standard installation exit return codes cause the following actions:

  - USERASIS (0) The statement is processed by the NetView DST module (DSIZDST). If it is not a valid DSTINIT statement, DSIZDST rejects it with an error message and continues processing.
**Writing User Subtasks**

**USERDROP (4)**
The statement is not processed by DSIZDST. Use this return code if your installation exit is going to process the statement (you can define your own initialization statements).

**USERSWAP (8)**
The swapped buffer is not processed by DSIZDST. If the swapped buffer does not contain a valid DSTINIT statement, it is rejected by DSIZDST and processing continues.

When returning from the last call (for end of file), any nonzero return code terminates the DST. Termination occurs only if the initialization process has failed.

The initialization exit invokes the DSIPUSH service to define a LOGOFF routine. The LOGOFF routine is invoked during normal or abnormal end-of-task processing. No termination exit is provided. The LOGOFF routine frees any resources that the user has acquired. Storage that has been acquired with the Q=YES option is automatically freed by the DSIZDST module.

**Note:** For additional details on DSTINIT statements, refer to the Tivoli NetView for z/OS Administration Reference.

**Data Services Command Processor**
A data services command processor (DSCP) generally performs CNM data services using macro DSIZCSMS, or VSAM data services using macro DSIZVSMS, or both services.

When a DST calls a DSCP, the input to the DSCP includes the address of a data services request block (DSRB) in the CWBDSRB field. The function code (DSRBFNCD) indicates the purpose for which the command was called. DSRBFNCD is described in "DSIDSRB: Data Services Request Block" on page 122.

Observe these two restrictions when writing a DSCP:
- Only commands defined as TYPE=D or TYPE=RD can be called under a DST or queued to a DST. Call only user-defined commands directly. Commands called from your DSCP cannot use NetView macros DSIZVSMS or DSIZCSMS.
- Use only DSIPSS TYPE=OUTPUT or TYPE=IMMED. Messages sent this way go to the operator who started the subtask (that is, to the owner). If the subtask was started during NetView initialization, or if the owner has logged off, the message is sent to the primary POI task (PPT) for routing and automation.

Data services requests are generally sent with HDRMTYPE=HDRTYPEI. Operators can queue commands using the EXCMD command. These commands can be identified because they are HDRTYPET, HDRTYPEB, or HDRTYPQC. You can check the HDRMTYPE field and reject direct operator requests.

**CNM Data Services**
The DST provides access to both solicited and unsolicited CNM data. A DSCP can issue DSIZCSMS to solicit CNM data from the network. You can define a DSCP to receive unsolicited data from VTAM.

An access method control block (ACB) with AUTH=CNM must be defined to VTAM with the ACB name matching the task ID of the DST.
Unsolicited CNM Data Interface
VTAM provides a default table (ISTMGC01) that controls the routing of unsolicited CNM request units (RUs). You can write a supplemental table (ISTMGC00) to override the default routing information that VTAM provides. The routing information consists of a particular RU type and the name of an application that is to receive the particular type of data. When a DST is defined with CNMI services, an ACB is opened with an ACB name (the application name) that is the same as the task name defined by the TSKID operand of the DST TASK definition statement. The exception is the hardware monitor, whose CNMI DST task name is BNJDSERV, but the application name is BNJHWMON. If the DST task name is entered as the application name in the VTAM routing table, the unsolicited data RU is passed to the unsolicited data services command processor for that DST.

DSTINIT Keywords: Keywords for the DSTINIT statement are:

UNSOL
Specifies the command verb name of the module that is to serve as the unsolicited DSCP for this DST. The unsolicited DSCP cannot issue the DSIZCSMS macro, but can issue the DSIZVSMS macro.

DSRBU
Specifies the number of unsolicited DSRBs that are to be allocated to this DST. If this DST does not process unsolicited CNM data, set this value to 0. If the unsolicited DSCP issues the DSIZVSMS macro, set this value to the number of concurrent DSIZVSMS requests that are valid. If the unsolicited DSCP does not issue the DSIZVSMS macro, set this value to 1.

Note: To issue DSIZVSMS, you must also specify FUNCT=BOTH.

DSCP Interface: When the unsolicited DSCP receives control, the DSRBFNCD field contains the DSRBFUNS (unsolicited) function code, DSRBUBUF is 0, and DSRBCUSB contains the address of a NetView buffer containing the unsolicited data. The RU starts at the offset specified in HDRTDISP, and the RU length is in HDRMLENG. If a deliver header is present, it is considered part of the data (for example, HDRTDISP points to the start of the deliver header). Refer to the VTAM library for more information. Table 11 describes the return codes on entry to the unsolicited DSCP.

Table 11. Return Codes on Entry to the Unsolicited DSCP

<table>
<thead>
<tr>
<th>DSRBRCMA</th>
<th>DSRBRCMI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>00</td>
<td>16</td>
<td>Installation exit rejected the deliver RU. HDRMLENG is set to 0.</td>
</tr>
<tr>
<td>00</td>
<td>20</td>
<td>Data has been truncated. The length of the deliver RU is greater than the length of the buffer. HDRMLENG is set to the truncated length.</td>
</tr>
<tr>
<td>00</td>
<td>24</td>
<td>Data is truncated after the installation exit returned with a return code of USERSWAP. HDRMLENG is set to the truncated length.</td>
</tr>
</tbody>
</table>

Solicited CNM Data Interface
A DSCP can invoke the DSIZCSMS macro to acquire CNM data from the network.

DSTINIT Keyword: The DSTINIT statement keyword is:
DSRBO

Specifies the number of solicited DSRBs that are required by this task and limits the number of concurrent DSIZCSMS and DSIZVSMS requests. This value must be at least 1 (a DSCP cannot be called unless a solicited DSRB is available) and no greater than 862.

DSCP Interface: Acquiring CNM data is a two-part process. When the DSCP is first driven (generally by a command buffer queued by an OST using a DSIMQS macro), the DSRBFNCD field contains a value of DSRBFNRM. The CWBDSRB field points to a DSRB that must be passed on the DSIZCSMS macro. You must issue the DSIZCSMS macro with this DSRB. After you issue the macro, register 15 contains the major return code and register 0 contains the minor return code (additional completion information). If register 15 is not 0, the macro has failed. If register 15 is 0, the request has successfully been sent to VTAM. At this time, the DSCP should exit because the data is returned on a subsequent invocation of the same DSCP. (This is called a redrive operation.)

When the DSCP is redriven (the second part of the process), the DSRBFNCD code is DSRBFSSOL. Check the DSRB major return code (DSRBRCMA) and the DSRB minor return code (DSRBRCMI) to determine whether the request completed successfully.

Table 12 describes major and minor return codes and their meanings.

Table 12. DSIZCSMS Return Codes

<table>
<thead>
<tr>
<th>DSRBRCMA</th>
<th>DSRBRCMI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>00</td>
<td>04</td>
<td>Negative response was received. DSRBINPT contains the address of the negative response.</td>
</tr>
<tr>
<td>00</td>
<td>08</td>
<td>Insufficient storage to process the request.</td>
</tr>
<tr>
<td>00</td>
<td>16</td>
<td>Installation exit rejected the deliver RU. HDRMLENG is set to 0.</td>
</tr>
<tr>
<td>00</td>
<td>20</td>
<td>Data has been truncated. The length of the deliver RU is greater than the length of the buffer. HDRMLENG is set to the truncated length.</td>
</tr>
<tr>
<td>00</td>
<td>24</td>
<td>Data is truncated after the installation exit returns with a return code of USERSWAP. HDRMLENG is set to the truncated length.</td>
</tr>
<tr>
<td>00</td>
<td>28</td>
<td>VTAM rejected the request.</td>
</tr>
<tr>
<td>00</td>
<td>32</td>
<td>CNM interface closed because of an unrecoverable error.</td>
</tr>
<tr>
<td>00</td>
<td>36</td>
<td>Positive response was received.</td>
</tr>
<tr>
<td>00</td>
<td>44</td>
<td>Cancellation because of timer completion. This code is returned only when running with VTAM V3R1.1 or later.</td>
</tr>
</tbody>
</table>

If the request completes successfully, the input buffer supplied on the initial DSIZCSMS invocation (INPUT parameter) contains the received data. The buffer contains a standard NetView buffer header with HDRTDISP containing the offset to the start of the data. If the data is preceded by a deliver RU, HDRTDISP contains the offset to the start of the deliver RU.

After the initial invocation of DSIZCSMS and until the DSCP is redriven, the DSRB is considered in use and is not available to other DSCPs. Other DSCPs can run
during this time if the DSRBO value is greater than 1 and there is a DSRB that is not in use by another DSCP. When the DSCP is redriven, the DSRB is the only control block that does not change from the initial invocation of the DSCP. You can use the DSRBUSER field to contain or point to any additional environment information that you want to maintain. See the description of "DSRBUSER" on page 124 for more information.

VSAM Service Interface

A DSCP can invoke the DSIZVSMS macro to perform input/output to a specified VSAM data set. You should use the access method services DEFINE CLUSTER statement to define a key-sequenced read/write data set as needed by NetView.

Note: For information about the DEFINE CLUSTER statement, refer to the MVS/DFP™ library.

DSTINIT Keywords

The following are the keywords for the DSTINIT statement. The primary and secondary data sets are user-defined and can be switched with each other.

PDDNM
Specifies the DD name of the primary data set to be used by VSAM services. Allocate this data set before starting the DST.

PPASS
Specifies the VSAM password to be used when the primary data set ACB is opened.

SDDNM
Specifies the DD name of the secondary data set to be used by VSAM services. Allocate this data set before starting the DST. The NetView SWITCH command controls which data set is currently the active data set.

SPASS
Specifies the VSAM password to be used when the secondary data set ACB is opened.

MACRF
Specifies local resource sharing.

XITVN
Specifies an installation exit to receive control when an empty VSAM data set has been opened for processing. This exit enables you to put an initialization record into the data set.

XITVI
Specifies an installation exit to receive control upon input from the VSAM data set before the input record is passed to the requesting DSCP.

XITVO
Specifies an installation exit to receive control before output of a record to the VSAM data set.

Note: If DSRBO is greater than 1, NetView does not guarantee that the DSIZVSMS requests for VSAM PUTS are processed in the order they were submitted. The requests are completed asynchronously.

DSCP Interface

Like the CNMI service (DSIZCSMS), using DSIZVSMS is a two-part process.
When the DSCP is first driven, generally by a command buffer sent by an OST using a DSIMQS macro, the DSRBFNCD field contains a value of DSRBFNRM. The CWBDSRB field points to a DSRB that must be passed on the DSIZVSMS macro.

Issue the DSIZVSMS macro with the supplied DSRB. After you issue the macro, register 15 contains the major return code and register 0 contains the minor return code, which is additional completion information. If register 15 is not 0, the macro has failed. If register 15 is 0, the request has successfully been sent to VSAM. At this time, the DSCP should exit because the success or failure of the VSAM input/output requested is returned on a subsequent invocation of the same DSCP. This is called a redrive operation. When the DSCP is redriven, the DSRBFNCD code is DSRBFVSM. Check the DSRBRCMA (DSRB major return code) and the DSRBRCMI (DSRB minor return code) to see if the request completed successfully.

Table 13 describes the major and minor return codes for the DSIZVSMS macro.

<table>
<thead>
<tr>
<th>DSRBRCMA</th>
<th>DSRBRCMI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>00</td>
<td>16</td>
<td>Installation exit processing of VSAM input has rejected the input. HDRM LENG is set to 0.</td>
</tr>
<tr>
<td>00</td>
<td>24</td>
<td>Data has been truncated. Installation exit returned data longer than NetView buffer on RC = USERSWAP. HDRM LENG is set to the truncated length.</td>
</tr>
<tr>
<td>00</td>
<td>28</td>
<td>Return code from installation exit is not valid.</td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>VSAM RPL feedback VSAM logical error, indicated in DSRBRCMI. Refer to the OS/VS VSAM library and the MVS/ESA VSAM library.</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>VSAM RPL feedback VSAM physical error, indicated in DSRBRCMI. Refer to the OS/VS VSAM library and the MVS/ESA VSAM library.</td>
</tr>
</tbody>
</table>

Relevant DSRB fields are as follows:

**DSRBVRPL**

The address of the VSAM RPL that was used for the input/output.

**DSRBVACB**

The address of the VSAM ACB for the DST.

**DSRBVDAD**

The address of the VSAM input/output buffer, with a standard BUFHDR. For GET requests, the BUFHDR HDRM LENG field indicates the length of the data read. HDRTDISP contains the offset to the data.

**DSRBVKLEY**

The address of the key in the DSRBVDAD buffer.

**DSRBVKLN**

The key length.

**DSRBVRTP**

Indicates the type of request just completed:

1 - DSRVGET (VSAM GET)
2 - DSRVPUT (VSAM PUT)
3 - DSRVPNT (VSAM POINT)
4 - DSRVERS (VSAM ERASE)
After the initial invocation of DSIZVSMS and until the DSCP is redriven, the DSRB is considered in use and is not available to other DSCPs. Other DSCPs can run during this time if the DSRBO value is greater than 1 and there is a DSRB that is not in use by another DSCP. When the DSCP is redriven, the DSRB is the only control block that does not change from the initial invocation of the DSCP. You can use the DSRBUSER field to contain or point to any additional environment information that you wish to maintain. See the description of “DSRBUSER” on page 127 for more information.

Example of DSCP Design

You can use a DSCP to solicit CNM data from a resource in the network and record the results on a VSAM data set. To accomplish this, DSCP processing can follow the steps that are listed below and referenced in Figure 16 on page 98.

1. A DST that receives an IFRCODCR buffer calls the DSCP with a function code of initial call. The DSCP issues DSIZCSMS to solicit the CNM data. The DSCP returns to the caller.

2. The DSCP is redriven with a function code of solicited CNM data. The DSCP issues DSIZVSMS to write the data to the VSAM data set. The DSCP returns to the caller.

3. The DSCP is redriven with a function code of VSAM I/O completed. If more than one CNM buffer is needed, the DSCP issues another DSIZCSMS to retrieve the buffer. The DSCP returns to the caller.

4. Steps 2 and 3 repeat alternately until the DSCP has retrieved all of the data.

5. When the DSCP is redriven for completion of the last VSAM PUT, it sends a completion message and returns control to the DST. Because neither DSIZCSMS nor DSIZVSMS was issued, the DSRB is not redriven. Processing for the DSCP ends.

---

1. When a DSCP is redriven, its input control blocks and fields, except for the DSRB, can be completely different from those used in its previous invocation.
Figure 17 on page 99 shows one way you can structure data services requests. This example starts with an initial operator command. This command invokes the DSCP using DSIMQS with an IFRCODCR buffer. When the DSCP has obtained VSAM or CNM data to be presented, it can do either of the following:

- Send the message data to the terminal (for standard or title-line output).
- Invoke a presentation services command processor (PSCP) to present the data (for full-screen output).

Figure 16. Structure of a Data Services Command Processor
The command processor uses DSIGET to obtain storage. The address of this storage can be saved in DSRBUSER (provided that the DSRB is considered “in use”) or DSIPUSH can save it as a named storage pointer. Saving this information establishes and maintains data from one DSCP call to the next.

If the DSCP does not use the parse buffer PDB, you can improve the performance by specifying PARSE=N on the CMDMDL statement defining the DSCP. In this case, the command buffer is not parsed and no PDB is provided to the command processor.
Writing User Subtasks

An operator can have one or more pending DST requests. You can use the LIST DST command to list active DST requests.

User-Defined Services

You can invoke command processors defined as TYPE=D or TYPE=RD under the DST to perform user functions. The processors are invoked with a standard NetView command processor interface (register 1 points to a DSICWB control block). If parsing of the command buffer is not required, specify PARSE=N on the respective CMDMDL statement for the DSCP. This option improves performance.

See “Appendix A. Assembler Samples” on page 283 for an example of a user-written optional subtask.
Chapter 6. Writing User Function Directories

This chapter explains how to add functions, expand, or replace those REXX functions that exist in the NetView REXX environment.

The MVS environment uses the DSIRXEBS macro described in Chapter 8. Macros to obtain storage for an evaluation block.

Overview of User-Written Functions

You can write external functions that enable you to extend the capabilities of the REXX language. You can write functions that supplement either the built-in functions or the functions that are provided. You can also write a function that can be used in place of a function already provided. For example, if you want a new substring function that performs differently from the SUBSTR built-in function, you can write your own substring function and name it STRING. Users at your installation can then use the STRING function in their REXX command lists.

You can write an external function or subroutine in assembler and store it in a load library. This provides faster access to the function or subroutine than if the function or subroutine were written in REXX. For even faster access to a function or subroutine, and therefore better performance, you can group frequently used external functions and subroutines in function packages (grouped or packaged together). To include an external function or subroutine in a function package, it must be linked into a load module together with a function package directory. The name of the resulting load module must be added to the function package table in NetView module DSIRXPRM. Refer to sample CNMSJMJ11 for the default NetView DSIRXPRM module, which includes the function package table, and modify it.

NetView supports the following types of function package directories:

- DSIRXUFP user packages, which are function packages that an individual user can write to replace or supplement certain system-provided functions. When the function packages are searched, the user packages are searched before the local and system packages.
- DSIRXLFP local packages, which are function packages that a system support group or application group can write. Local packages can contain functions that are available to a specific group of users or to the entire installation. Local packages are searched after the user packages and before the system packages.
- DSIRXFPG system packages, which are function packages written for NetView. System packages are searched after any user and local packages.

Subroutines can also be included in the function package directories.

To provide new functions or change existing functions, several steps are required. The steps are described and explained in more detail in the following sections.
When your code gets control, the function gets a control block called the evaluation block (EVALBLOK). The function places the result in the evaluation block, which is returned to the language processor. The result in the evaluation block is used in the interpretation of the REXX instruction that contained the function.

For instructions on obtaining an evaluation block, see the NetView DSIRXEBS macro description in Chapter 8. Macros.

Entry Specifications
When the code for the function gets control, the contents of the registers are as follows:

Register 0
Environment block for MVS (TIB address is in ENVBLOCK_USERFIELD)

Register 1
Address of the external function parameter list (EFPL)

Registers 2-12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

External Function Parameter List
When the function gets control, register 1 points to the external function parameter list, as described in Table 14. TSO/E provides the mapping macro IRXEFPL for the external function parameter list.

Table 14. External Function Parameter List

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>The address of the parsed argument list. Table 15 on page 103 shows the format of the argument list.</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>The address of a fullword that contains the address of an evaluation block (EVALBLOK). The evaluation block is used to pass back the result of the function. Table 16 on page 103 describes the evaluation block.</td>
</tr>
</tbody>
</table>

Argument List
Table 15 on page 103 shows the format of the parsed argument list that the function receives at offset +16 (decimal). TSO/E provides the mapping macro IRXARGTB
for the argument list.

Table 15. Format of the Argument List

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>ARGSTRING_PTR</td>
<td>Address of argument 1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>ARGSTRING_LENGTH</td>
<td>Length of argument 1</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>ARGSTRING_PTR</td>
<td>Address of argument 2</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>ARGSTRING_LENGTH</td>
<td>Length of argument 2</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>ARGSTRING_PTR</td>
<td>Address of argument 3</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>ARGSTRING_LENGTH</td>
<td>Length of argument 3</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>---</td>
<td>X'FFFFFFFFFFFFFFFF'</td>
</tr>
</tbody>
</table>

In the argument list, each argument consists of the address of the argument and its length. The argument list is terminated by X'FFFFFFFFFFFFFFFF'.

**Evaluation Block**

Before the function returns control to the language processor, the address of a fullword that contains the address of the evaluation block (EVALBLOK) is placed at offset +20 of the external function parameter list. The function computes the result and returns the result in the evaluation block.

The evaluation block consists of a header and data, in which you place the result from your function. Table 16 shows the format of the evaluation block.

TSO/E provides the mapping macro IRXEVALB for the evaluation block.

Table 16. Format of the Evaluation Block

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Number of Bytes</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>EVPAD1</td>
<td>A fullword that contains X'00'. This field is reserved and is not used.</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>EVSIZE</td>
<td>Specifies the total size of the evaluation block, in doublewords.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>EVLEN</td>
<td>On entry, this field is set to X'80000000', which indicates no result is currently stored in the evaluation block. On return, specify the length of the result, in bytes, that your code is returning. The result is returned in the EVDATA field at offset +16.</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>EVPAD2</td>
<td>A fullword that contains X'00'. This field is reserved and is not used.</td>
</tr>
<tr>
<td>16</td>
<td>n</td>
<td>EVDATA</td>
<td>The field in which you place the result from the function or subroutine. The length of the field depends on the total size specified for the control block in the EVSIZE field. The total size of the EVDATA field is EVSIZE 8–16.</td>
</tr>
</tbody>
</table>

The function computes the result, moves the result into the EVDATA field, and updates the EVLEN field. If the initial evaluation block is too small to hold the complete result, you can use the DSIRXEBS macro to obtain a larger evaluation
Writing User Function Directories

block. DSIRXEBS creates the new evaluation block and returns the address of the new block. Your code can then place the result in the new evaluation block. You must also change the parameter for the address of the EVALBLOK in the parameter list to point to the new evaluation block. If you used DSIRXEBS to get the initial evaluation block, DSIRXEBS releases the evaluation block if its address is provided when obtaining the larger evaluation block.

Functions must return a result. Subroutines are not required to return a result.

Directory for Function Packages

After writing the code for the function, create an entry in one of the directories. You need a directory entry for each individual function package you want defined.

Note: Functions not contained in the function package can be link-edited into the load library as stand-alone functions.

A load module contains a function package directory that you can tailor to individual users or local groups. The name of the entry point at the beginning of the directory is the function package directory name. The name of the directory is specified only on the CSECT. In addition to the name of the entry point, the function package directories define each entry point for the individual functions that are part of the function package. The directories consist of two parts:

- A header
- Individual entries for each function included in the function package

Table 17 on page 104 shows the format of the directory header. Table 18 on page 105 illustrates the rows of entries in the function package directory.

Table 17. Format of the Function Package Directory Header

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>An 8-byte character field that defines the directory. This is the name of the directory. For example, you can specify DSIRXUFP, which is one of the dummy function package names that is provided. The name must be in uppercase and left-justified.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Specifies the length, in bytes, of the header. This is the offset from the beginning of the header to the first entry in the directory. This must be a fullword binary number equivalent to decimal 24.</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Specifies the number of functions defined in the function package (the number of rows in the directory). The format is a fullword binary number.</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Reserved.</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>Specifies the length, in bytes, of an entry in the directory (length of a row). This must be a fullword binary number equivalent to decimal 32.</td>
</tr>
</tbody>
</table>

At offset +0 in the header, specify the name of the function package directory. Two dummy function package directory names are provided:

- DSIRXUFP for a user function package
- DSIRXLFP for a local function package
Format of the Entry in the Directory

Table 18 shows an entry in a function package directory. The entry starts immediately after the directory header and defines a function or subroutine in the function package. The individual fields are described following the table.

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Number of Bytes</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>FUNC-NAME</td>
<td>The name of the first function or subroutine (entry) in the directory.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>ADDRESS</td>
<td>The address of the entry point of the function or subroutine (for the first entry).</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>---</td>
<td>Reserved.</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>SYS-NAME</td>
<td>The name of the entry point in a load module that corresponds to the function or subroutine (for the first entry).</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>SYS-DD</td>
<td>The ddname from which the function or subroutine is loaded.</td>
</tr>
</tbody>
</table>

The following describes each entry (row) in the directory:

ADDRESS

Specifies a 4-byte field that contains the address, in storage, of the entry point of the function or subroutine. This address is used only if the code has already been loaded.

If the address is 0, the sys-name and, optionally, the sys-dd fields are used.

If you specify the address, the sys-name and sys-dd fields for the entry are ignored.

FUNC-NAME

Is the 8-character name of the external function or subroutine. This is the name that is used in the REXX command list. The name must be in uppercase and left-justified.

If this field is blank, the entry is ignored.

Reserved

Is a 4-byte field that is reserved.

SYS-DD

Specifies an 8-character name of the DD from which the function or subroutine is loaded. The name must be in uppercase and left-justified. If the address is 0 and this field is blank, the module is loaded from the linklist.

SYS-NAME

Supplies an 8-character name of the entry point in a load module that corresponds to the function to be called for the func-name. The name must be in uppercase and left-justified.

If you specify the address, this field can be blank. If you specify an address of 0 and this field is blank, the entry is ignored.

If you specify sys-dd, the LOAD is issued from DD sys-dd. If you do not specify sys-dd, the module is loaded from the linklist.
Writing User Function Directories

Example of a User Function Directory

Figure 18 shows an example of a user function directory. The example is explained following the figure:

```
DSIRXUFP CSECT
   DC CL8'DSIRXUFP' String identifying directory
   DC FL4'24' Length of header
   DC FL4'4' Number of rows in directory
   DC FL4'0' Word of zeros
   DC FL4'32' Length of directory entry
   * Start of definition of first entry
   DC CL8'MYF1 ' Name used in command list
   DC FL4'0' Address of preloaded code
   DC FL4'0' Reserved field
   DC CL8'ABCFUN1 ' Name of entry point
   DC CL8'FUNCTDD1' DD from which to load entry point
   * Start of definition of second entry
   DC CL8'MYF2 ' Name used in command list
   DC FL4'0' Address of preloaded code
   DC FL4'0' Reserved field
   DC CL8'ABCFUN2 ' Name of entry point
   DC CL8'FUNCTDD1' DD from which to load entry point
   * Start of definition of third entry
   DC CL8'MYS3 ' Name used in command list
   DC AL4(ABCSUB3) Address of preloaded code
   DC FL4'0' Reserved field
   DC CL8'ABCFUN3 ' Name of entry point
   DC CL8'FUNCTDD1' DD from which to load entry point
   * Start of definition of fourth entry
   DC CL8'MYF4 ' Name used in command list
   DC VL4(ABCFUNC4) Address of preloaded code
   DC FL4'0' Reserved field
   DC CL8' ' Name of entry point
   DC CL8'FUNCTDD1' DD from which to load entry point
   SPACE 2
END DSIRXUFP
```

Figure 18. Example of a Function Package Directory

In Figure 18, the name of the function package directory is DSIRXUFP, which is one of the dummy function package directory names provided with NetView. Four entries are defined in this function package:

- MYF1, an external function
- MYF2, an external function
- MYS3, a subroutine
- MYF4, an external function
<table>
<thead>
<tr>
<th>Service Routine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIVARS: Variable Services</td>
<td>243</td>
</tr>
<tr>
<td>Return Codes in Register 15</td>
<td>246</td>
</tr>
<tr>
<td>DSIWAT: ECB Wait Services</td>
<td>247</td>
</tr>
<tr>
<td>DSIWCS: Write Console Services</td>
<td>248</td>
</tr>
<tr>
<td>Return Code in Register 15</td>
<td>248</td>
</tr>
<tr>
<td>DSIWLS: Write Log Services</td>
<td>249</td>
</tr>
<tr>
<td>Return Codes in Register 15</td>
<td>250</td>
</tr>
<tr>
<td>DSIZCIMS: CNM Data Services</td>
<td>251</td>
</tr>
<tr>
<td>Return Codes in Register 15</td>
<td>254</td>
</tr>
<tr>
<td>Minor Return Codes in Register 0</td>
<td>254</td>
</tr>
<tr>
<td>CNM Data Services Module Examples</td>
<td>255</td>
</tr>
<tr>
<td>DSIZVSM: VSAM Data Services</td>
<td>256</td>
</tr>
<tr>
<td>Return Codes in Register 15</td>
<td>257</td>
</tr>
<tr>
<td>Minor Return Codes in Register 0</td>
<td>257</td>
</tr>
<tr>
<td>DSIZ6REGS: LU 6.2 Registration</td>
<td>258</td>
</tr>
<tr>
<td>Return Codes in Register 15</td>
<td>263</td>
</tr>
<tr>
<td>DSIZ6SNDS: Send a Message Unit</td>
<td>264</td>
</tr>
<tr>
<td>Return Codes in Register 15</td>
<td>270</td>
</tr>
<tr>
<td>DULIFSMTE: Customize the DisplayStatus Table for</td>
<td>272</td>
</tr>
<tr>
<td>Exception Views</td>
<td></td>
</tr>
<tr>
<td>Default Values for DisplayStatus</td>
<td>273</td>
</tr>
<tr>
<td>Specifying Resource Names for DisplayStatus</td>
<td></td>
</tr>
<tr>
<td>Mapping</td>
<td>273</td>
</tr>
<tr>
<td>Examples of Customizing DisplayStatus Mapping</td>
<td></td>
</tr>
<tr>
<td>Mapping</td>
<td>274</td>
</tr>
<tr>
<td>Assembling and Link-Editing the DisplayStatus Mapping Table</td>
<td>275</td>
</tr>
<tr>
<td>Chapter 9. Called Service Routines</td>
<td>277</td>
</tr>
<tr>
<td>Overview of Called Service Routines</td>
<td>277</td>
</tr>
<tr>
<td>Free NetView Buffers Service Routine (DSIFREBF)</td>
<td>277</td>
</tr>
<tr>
<td>Return Code.</td>
<td>278</td>
</tr>
<tr>
<td>Abend Code.</td>
<td>278</td>
</tr>
<tr>
<td>Process Message Data Block Routine (DSIMMDB)</td>
<td>278</td>
</tr>
<tr>
<td>Return Codes</td>
<td>279</td>
</tr>
</tbody>
</table>
Chapter 7. Control Blocks

This chapter describes control block fields related to customization interfaces. The other fields that can appear in control block mapping macros are those containing the internal use control information and are not to be used as a programming interface.

NetView control blocks and buffers passed to installation exits and command processors are intended for read-only use. Do not alter them, with the exception of fields specifically designed as user fields, such as TVBUFLD, CWBSAVEA, and CWBADATD.

MVTUFLD, TVBUFLD, and TIBUFLD are user fields. They are 4-byte fields that NetView does not modify or refer to. User fields are for non-NetView code, such as customer code or other applications running on NetView.

One DSIMVT and one MVTUFLD field exist for each NetView program. One TVBUFLD and one TIBUFLD exist for each NetView subtask and for each optional task.

Any product that uses NetView’s task user fields can remove valuable NetView resources. Some NetView functions can keep you from overriding task user fields. See “DSIPUSH: Establish Long-Running Command” on page 223 for information about the DSIPUSH function.

Because NetView only provides a single set of user fields, use a method to communicate the application usage of a particular field so that subsequently developed applications do not interfere with the existing usage of the field. Use one of these methods or one of your own:

- Any application or product using any field should document its use. If a task user field (such as TVBUFLD or TIBUFLD) is used, the name of the task (such as TVBOPID or TVBLUNAM) should be documented and communicated to anyone who uses that application or product.
  
  For example, if a component of a product is coded to run as an optional task, and the product uses the TIB or TVB user fields of that optional task, that information should be included in the product documentation.

- The TVBUFLD and TIBUFLD fields of a non-NetView task should be under control of the code that “owns” the task. In the previous example, if a customer queries the value in the TVBUFLD of every TVB on the TVB chain, do not query the TVBUFLD in the product’s optional task. The product may be using the TVBUFLD of its task for a purpose that conflicts with the customer’s intended use.

**BUFHDR: Buffer Header**

All message and command buffers have an initialized buffer header (BUFHDR) preceding the buffer text. BUFHDR describes the buffer’s size and use, and the origin of the message or command. The items marked with an asterisk (*) in Figure 19 on page 110 are the BUFHDR fields you must initialize.

BUFHDR is a separate DSECT contained in the task information block (DSITIB). To obtain the BUFHDR DSECT, use macro DSICBS to include the TIB control block.
The size of the BUFHDR was increased in a previous release of NetView. The BUFHDR presentation-extension fields shown in Figure 19 were added.

![Figure 19. Buffer Header (BUFHDR)](image-url)
An initialized BUFHDR has the fields set as shown in Table 19.

**Table 19. Control Block Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFHDRND</td>
<td>0</td>
<td>Label at the end of BUFHDR for use in computing the length, in bytes, of BUFHDR.</td>
</tr>
<tr>
<td>HDRBLENG</td>
<td>2</td>
<td>The length, in bytes, of the entire buffer: header, text, and unused space. This length is used if the buffer is to be released with macro DSIFRE. It is a number between 0 and 32767.</td>
</tr>
<tr>
<td>HDRDOMID</td>
<td>8</td>
<td>The identifier of the domain that originated the message. This field is displayed and logged. The domain identifier under which a particular program is running is shown in the MVTCURAN field of the main vector table (DSIMVT). The value of HDRDOMID equals the value of MVTCURAN. MVTCURAN is an 8-byte field that contains a field DOMAINID of 5 bytes and is padded on the right with blanks. The three bytes farthest to the right are reserved.</td>
</tr>
</tbody>
</table>
| HDRIND         | 1      | This field is normally set to 0, except when HDRMTYPE=HDRTYPEJ, HDRTYPEK, or HDRTYPEL. In these cases:  
|                |        | HDRLNCTL TYPE = CONTROL LINE  
|                |        | HDRLNLBL TYPE = LABEL LINE  
|                |        | HDRLNDAT TYPE = DATA LINE  
|                |        | HDRLNEND TYPE = DATA END LINE.  
|                |        | Note: A multiline message within the NetView program can end with a buffer that contains no data, but it is simply marked as the end of the multiline message (HDRIND = HDRLNEND). In this case, the HDRMLENG field of this last buffer is 0.  
|                |        | When HDRMTYPE=HDRTYP10:  
|                |        | HDRLNMSU TYPE=MSU DATA BUFFER  
|                |        | HDRLNHIR TYPE=MSU HIERARCHY BUFFER |
| HDRMCEXT       | 12     | An extension that is appended to the BUFHDR when a buffer is transferred from one subtask to another. Except for a buffer with HDRMTYPE of HDRTYPEI (an IFR), which must already have an extension, macro DSIMQS BFRFLG=NO builds this extension when it creates a buffer copy for the destination task. If you want to pass the actual buffer with DSIMQS BFRFLG=YES, you must build the extension and initialize HDRSENDR. |
| HDRMLENG       | 2      | The length, in bytes, of the text in the buffer. |
| HDRMSG         | 0      | Label to indicate the place for text to begin if HDRMCEXT is present. |
| HDRMSGGLN      | 0      | Label at the end of HDRMCEXT for use in computing the length, in bytes, of BUFHDR+HDRMCEXT. |
| HDRMTYPE       | 1      | Indicates the current use of the buffer or the origin of the command. If the buffer is written using macro DSIPSS, this character is displayed and logged. For a list of values for HDRMTYPE, see Table 21 on page 112. |
| HDRNEXTM       | 4      | Multiline write-to-operator (MLWTO) buffer chain pointer; points to the next buffer in the chain. |
### Values for HDRIND Fields

The following table shows the values for the HDRIND fields:

<table>
<thead>
<tr>
<th>Field Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| HDRLNCTL    | The control line of the BUFHDR.  
  Note: If you use HDRTCNT, set HDRLNTYP to the value of HDRLNCTL, and set HDRMTYPE to HDRTYPEJ, HDRTYPEK, or HDRTYPEL. |
| HDRLNDAT    | The data line of the BUFHDR.  
  Note: If you use HDRTDATT, set HDRLNTYP to the value of HDRLNDAT, and set HDRMTYPE to HDRTYPEJ, HDRTYPEK, or HDRTYPEL. |
| HDRLNEND    | The end line of the BUFHDR.  
  Note: If you use HDRTENDT, set HDRLNTYP to the value of HDRLNEND, and set HDRMTYPE to HDRTYPEJ, HDRTYPEK, or HDRTYPEL. |
| HDRLNLBL    | The label line of the BUFHDR.  
  Note: If you use HDRTLABT, set HDRLNTYP to the value of HDRLNLBL, and set HDRMTYPE to HDRTYPEJ, HDRTYPEK, or HDRTYPEL. |
| HDRLNMSU    | The length, in bytes, of the MSU data buffer. |

### Values for HDRMTYPE Fields

The following table shows the values for the HDRMTYPE fields:

<table>
<thead>
<tr>
<th>Field Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRTYPDT (D)</td>
<td>Indicates nonmessage data type.</td>
</tr>
</tbody>
</table>
Table 21. Values for HDRMTYPE Fields (continued)

<table>
<thead>
<tr>
<th>Field Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRTYPEB (?)</td>
<td>Indicates a command or command list buffer that suppresses display and logging. It is not displayed on the operator’s screen. This field type is used to suppress display and logging of commands entered with a suppression character as defined in initialization member CNMSTYLE. It is also used to suppress display and logging of command list statements that are preceded by this same suppression character.</td>
</tr>
<tr>
<td>HDRTYPEC (C)</td>
<td>Indicates a command or message from a command list. This field type changes to HDRTYPEB for suppressed command list statements. It changes to HDRTYPQC for quiet commands.</td>
</tr>
<tr>
<td>HDRTYPED (!)</td>
<td>Indicates a message from an immediate command processor. It is usually sent to the screen using DSIPSS TYPE=IMMED. When displayed in the immediate message area on the screen, the HDRMTYPE and DOMAIN name are not displayed. When received cross-domain, this type of message is in the normal output area, along with its domain name and type prefix. DSIPSS TYPE=IMMED does not enforce or set HDRTYPED.</td>
</tr>
<tr>
<td>HDRTYPEE (E)</td>
<td>Indicates a message from the operating system interface. This type is not used for title-line mode (MLWTO), system action, or WTOR messages. Also see HDRTYPEK and HDRTYPEY for other forms of operating system interface messages.</td>
</tr>
<tr>
<td>HDRTYPEF (F)</td>
<td>Indicates a VSAM record. This is not displayed on the operator’s screen. This type is used within the data services task (DST).</td>
</tr>
<tr>
<td>HDRTYPEG (G)</td>
<td>Indicates a CNMI record. This is not displayed on the operator’s screen. This type is used within the data services task (DST).</td>
</tr>
<tr>
<td>HDRTYPEI (I)</td>
<td>Indicates an internal function request (DSIIFR). This buffer is a formatted interface within and between tasks. The IFR contains a function number (IFRCODE) that determines the format and function of the buffer.</td>
</tr>
<tr>
<td>HDRTYPEJ (')</td>
<td>Indicates a title-line (MLWTO) message originating from the NetView program. These buffers must be in a sequence and include a description of control, label, data, and end designators. The NetView program treats these sequences of buffers as a single message for presentation and automation.</td>
</tr>
<tr>
<td>HDRTYPEK ('')</td>
<td>The same as HDRTYPEJ but for IBM non-NetView code.</td>
</tr>
<tr>
<td>HDRTYPEL (=)</td>
<td>The same as HDRTYPEJ but for non-IBM written code.</td>
</tr>
<tr>
<td>HDRTYPEM (M)</td>
<td>Indicates a message from the NetView message command processor.</td>
</tr>
<tr>
<td>HDRTYPEN (-)</td>
<td>Indicates a regular single buffer message from the NetView program.</td>
</tr>
<tr>
<td>HDRTYPOR (</td>
<td>)</td>
</tr>
<tr>
<td>HDRTYPEQ (Q)</td>
<td>Indicates a message from the VTAM POI that is a single-buffer unsolicited message. See also HDRTYPEV, HDRTYPEY, and HDRTYPEK for other VTAM POI messages. This message type is not set for messages from VTAM received on the operating system interface.</td>
</tr>
<tr>
<td>HDRTYPER (R)</td>
<td>Indicates that an operator entered the VTAM REPLY command in response to NetView WTOR number DSIs02A. This message type is logged but does not appear on NetView consoles.</td>
</tr>
<tr>
<td>HDRTYPES (S)</td>
<td>On some installation exit interfaces, HDRMTYPE is set to HDRTYPES to indicate a swapped buffer.</td>
</tr>
<tr>
<td>Field Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>HDRTYPET (*)</td>
<td>Indicates a command issued to the NetView program from a NetView terminal. This message type indicates that the buffer is a command rather than a message. HDRTYPET generally implies a command buffer as if an operator had typed the command. HDRTYPET buffers cannot contain non-printable text.</td>
</tr>
<tr>
<td>HDRTYPEU (U)</td>
<td>Reserved for non-IBM users. This type cannot be used for action messages (WTOR) or title-line (MLWTO) messages.</td>
</tr>
<tr>
<td>HDRTYPEV X'40'</td>
<td>Indicates a message from the VTAM POI that is a single-buffer solicited message. Also see HDRTYPEQ, HDRTYPEY, and HDRTYPEK for other VTAM POI messages. This message type is not set for messages from VTAM received on the operating system interface.</td>
</tr>
<tr>
<td>HDRTYPEW (+)</td>
<td>Indicates an IBM-written single-line message. This type is similar to HDRTYPEN and HDRTYPEU.</td>
</tr>
<tr>
<td>HDRTYPEX (X)</td>
<td>Indicates a cross-domain (NNT-OST) command. Code running in an NNT can issue DSIPSS TYPE=OUTPUT for a HDRTYPEX buffer, and the corresponding command is executed in the OST that started the session with that NNT. This message type is useful for sending nonformatted (hexadecimal) data from the NNT to an OST for full-screen or other formatting. This type is limited to 256 bytes. It is not displayed on the operator’s screen.</td>
</tr>
<tr>
<td>HDRTYPEY (&gt;)</td>
<td>Indicates single-buffer action or WTOR. This type can be a message from the operating system interface and from the VTAM POI. These messages remain on the screen until an action is taken or the reply is entered. The operator can delete these messages by overstriking the &gt; character and pressing ENTER. The message disappears the next time the screen wraps over the text. When the HDRTYPEY flag is set and the IFRAUWQE flag is not set, the NetView program looks for a 3-character reply ID immediately preceding the message number in the message text. If this reply ID exists, then the message is a VTAM WTOR. Otherwise, the message is treated as a held message. If IFRAUWQE is set to 1, the IFRAUWQD data is checked to see if the WQE data indicates a WTOR or action message. A WTOR is indicated if a 2-character reply ID immediately precedes the message ID. If the reply ID exists, it is delimited from the message ID by one space. Otherwise, the message is an action message.</td>
</tr>
<tr>
<td>HDRTYPEZ (Z)</td>
<td>Similar to HDRTYPEN but specifically indicates a message from a data services task (DST).</td>
</tr>
<tr>
<td>HDRTYPES ($)</td>
<td>Indicates a nondisplayable data message. This type is used for data transfer between high-level language command procedures.</td>
</tr>
<tr>
<td>HDRTYPELT (L)</td>
<td>Indicates a TRACE record. This message type is not displayed on the screen or in the NetView logs.</td>
</tr>
<tr>
<td>HDRTYPELX (x)</td>
<td>Indicates a message response from a UNIX® command.</td>
</tr>
<tr>
<td>HDRTYPEQC X'32'</td>
<td>Indicates a command with all synchronous messages suppressed (performance option).</td>
</tr>
<tr>
<td>HDRTYPES (t)</td>
<td>Indicates a message response from a TSO command.</td>
</tr>
<tr>
<td>HDRTYPEWB (B)</td>
<td>Indicates a command issued from the NetView Web browser. This message type indicates that the buffer is a command rather than a message.</td>
</tr>
</tbody>
</table>
Table 21. Values for HDRMTYPE Fields (continued)

<table>
<thead>
<tr>
<th>Field Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRTYPWT (W)</td>
<td>Indicates a message that matched an &amp;WAIT and was displayed. The W appears in the Match Fonts field on the panel and in the logs but is not in the HDRMTYPE field in the buffer. The HDRMTYPE field in the buffer contains the original message type.</td>
</tr>
<tr>
<td>HDRTYPE1 (V)</td>
<td>Indicates PPOLOG echo of console operator command.</td>
</tr>
<tr>
<td>HDRTYPE2 (Y)</td>
<td>Indicates PPOLOG copy of console message, suppressed or unsuppressed.</td>
</tr>
<tr>
<td>HDRTYP10 X’10’</td>
<td>Indicates management services unit (MSU) data buffer.</td>
</tr>
<tr>
<td>HDRTYPZZ X’37’</td>
<td>Indicates end-of-response message.</td>
</tr>
</tbody>
</table>

**BUFHDR Presentation Extension**

Presentation information is present in all text data buffers in automation internal function request (AIFR) structures if you have turned on the IFRAUMVI indicator. Table 22 shows the values for the HDRMCEXT fields.

**Note:** You must index HDRTDISP beyond HDRTMSGT.

Table 22. Description of HDRMCEXT Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRTLLEN</td>
<td>2</td>
<td>The length of the text object (10 decimal).</td>
</tr>
<tr>
<td>HDRTTYPE</td>
<td>2</td>
<td>The type of text object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRTOBJ</td>
<td>The type value for the message text object.</td>
</tr>
</tbody>
</table>

| HDRTLNTY | 2 | The line type flags. The first byte is the value of HDRLNT1; the second is the value of HDRLNT2. For information on using this field, see page 150 and the programming notes in macro DSITIB. |

| HDRTPCON | 1 | Alarm indication. (Not used by the NetView program.) To use this field, see the programming notes in macro DSITIB. |

| HDRTPCOL | 1 | The presentation color field. The following HDRTPCOL color values override any other setting for the foreground color for one line of a multiple line message or the entire message for single-buffer messages: |

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRDEFCO</td>
<td>Default color</td>
</tr>
<tr>
<td>HDRBLACK</td>
<td>Black</td>
</tr>
<tr>
<td>HDRBLUE</td>
<td>Blue</td>
</tr>
<tr>
<td>HRRRED</td>
<td>Red</td>
</tr>
<tr>
<td>HDRPINK</td>
<td>Pink</td>
</tr>
<tr>
<td>HDRGREEN</td>
<td>Green</td>
</tr>
<tr>
<td>HDRTURQ</td>
<td>Turquoise</td>
</tr>
<tr>
<td>HDRYELLOW</td>
<td>Yellow</td>
</tr>
<tr>
<td>HDRWHITE</td>
<td>White</td>
</tr>
</tbody>
</table>

To use this field, see the programming notes in macro DSITIB.
Table 22. Description of HDRMCEXT Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRTPHIL</td>
<td>1</td>
<td>The foreground highlighting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To use this field, set both HDRTFPAF and IFRAUMVI on.</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>HDRDEFAU</td>
<td></td>
<td>Default highlighting</td>
</tr>
<tr>
<td>HDRHNONE</td>
<td></td>
<td>No highlighting</td>
</tr>
<tr>
<td>HDRLINK</td>
<td></td>
<td>Blinking characters</td>
</tr>
<tr>
<td>HDVRVIDEO</td>
<td></td>
<td>Reverse video</td>
</tr>
<tr>
<td>HDRUNDER</td>
<td></td>
<td>Underscored characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To use this field, see the programming notes in macro DSITIB.</td>
</tr>
</tbody>
</table>

| HDRTPINT     | 1      | Foreground intensity.                                                       |
| Value        | Meaning|                                                                               |
| HDRINDEF     |        | Default intensity                                                           |
| HDRINORM     |        | Normal intensity                                                            |
| HDRIHIGH     |        | High (bright) intensity                                                     |
|              |        | To use this field, set both HDRTFPAF and IFRAUMVI on.                      |

Note: A value of HDRINDEF causes checking of fields in DSIAIFRO, and/or the screen format values from the DEFAULTS and OVERRIDE commands, or the NetView standard default table.

Note: Do not use HDRTMSGT to reference message text. HDRTDISP is the true offset to text start.

Example Scenario of BUFHDR Usage

Macro DSIDKS uses the buffer header when reading a disk data set. You specify the blocking factor to block the disk data set. The disk services module DSIDRS prefixes the physical read buffer with a BUFHDR.

When the first record is requested, macro DSIDKS reads the first block. HDRTDISP is adjusted to indicate the first logical record. DSIDKS also sets HDRMLENG to reflect the logical record length. When DSIDKS is issued for the next logical record, HDRTDISP is adjusted to indicate the next logical record until the block is exhausted. DSIDKS reads another physical record; the process starts again from the first logical record in the block.

DSIAIFRO: Automation Vector Extensions

The DSIAIFRO maps the extensions to the AIFR, which are in vectorized object format. These extensions are chained from the base AIFR. The first DSIAIFRO of the chain is contained in the base AIFR.
Modifications of the AIFR Format

Figure 20 shows a diagram of the AIFR with a chain of DSIAIFRO objects. Figure 21 on page 118 shows a diagram of the AIFR extension.

IFRAUVEC in the base AIFR indicates the existence of AIFR extensions. This bit is zero when there is no AIFR extension. If this bit is one, there is at least one extension. The first (or only) extension commences at the 240th byte of the AIFR.

Every AIFR extension (including the first one) is in the format of an SNA GDS variable that can contain other SNA GDS variables. The contained items are called objects and are of the form:

\[
\text{LLLLKKKD...D,}
\]

Where:

- **LLLL** Represents a 2-byte length field
- **KKKK** Represents a 2-byte type-identification field
- **D...D** Represents either data values, or one or more contained objects

An AIFR extension is an object that can contain objects. In general, the objects it contains are unordered. Its first contained object is necessarily the next extension locator, and the next extension locator must be present and must be first.

The format of AIFR extension offsets are as follows:

- **0 1** AIFR extension length, including this length field (n+1).
- **2 3** AIFR extension type code = X’0010’.
- **4 n** The AIFR extension. \(n\) is determined by the preceding length field.

*Figure 20. DSIAIFRO Chain*

IFRAUVEC in the base AIFR indicates the existence of AIFR extensions. This bit is zero when there is no AIFR extension. If this bit is one, there is at least one extension. The first (or only) extension commences at the 240th byte of the AIFR.

Every AIFR extension (including the first one) is in the format of an SNA GDS variable that can contain other SNA GDS variables. The contained items are called objects and are of the form:

\[
\text{LLLLKKKD...D,}
\]

Where:

- **LLLL** Represents a 2-byte length field
- **KKKK** Represents a 2-byte type-identification field
- **D...D** Represents either data values, or one or more contained objects

An AIFR extension is an object that can contain objects. In general, the objects it contains are unordered. Its first contained object is necessarily the next extension locator, and the next extension locator must be present and must be first.

The format of AIFR extension offsets are as follows:

- **0 1** AIFR extension length, including this length field (n+1).
- **2 3** AIFR extension type code = X’0010’.
- **4 n** The AIFR extension. \(n\) is determined by the preceding length field.
### DSIAIFRO

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td>Length of first part of the AIFR extension</td>
</tr>
<tr>
<td></td>
<td>((m+1-4 = m-3))</td>
</tr>
<tr>
<td>6 7</td>
<td>Code for first part of the AIFR extension</td>
</tr>
<tr>
<td>8 m</td>
<td>Value or content of first part of the AIFR extension</td>
</tr>
<tr>
<td>m+1 m+2</td>
<td>Length of second part of the AIFR extension ((k-m))</td>
</tr>
<tr>
<td>m+3 m+4</td>
<td>Code for second part of the AIFR extension</td>
</tr>
<tr>
<td>m+5 k</td>
<td>Value or content of second part of the AIFR extension</td>
</tr>
</tbody>
</table>

Third and later parts of the AIFR extension

The minimum value of \(n\) is 15; that is, the minimum length of the AIFR extension is 16. There must be at least one contained object, because the first part of the AIFR extension must be the next extension locator.

The format of this locator is:

| 4 5     | Length of next extension locator (12)     |
| 6 7     | Code for next extension locator = X'0011' |
| 8 11    | Address space identifier for the next AIFR extension |
| 12 15   | Address of the next AIFR extension         |

If the address (positions 12-15) is zero, there is no further AIFR extension.

**Note:** In the NetView message processor, the maximum length of a DSIAIFRO object is 4 kilobytes.

The address space identifier is in access list entry token (ALET) format. When this identifier is set to zero, the extension is in the same address space as the base. (The ALET is a S/370™ architecture item, and is not operating system specific.)

### Format of the Message Control Object

A message control object can contain data from an MDB and its source object. The message control object is mapped by DSIMCO, which is contained within DSIAIFRO. The message control object, together with its contained objects, is used to transport the MDB data from a message data block. The message control object is one of the objects that can be included in a DSIAIFRO object. It is in the format of an SNA generalized data stream (GDS) variable that can contain other GDS variables. It begins with a 2-byte field that defines the length of the object, followed by a 2-byte code identifying it as a message control object. All of its contained objects are in the same form.

The message control object can contain an MDB global object and a source object. In turn, the MDB global object can contain a general object and a control program object. An example of this “nesting” is shown in Figure 21 on page 119.
Table 23 details the self-identifying codes for the message control object and the objects it can contain.

Table 23. Message Control Object Type/Self-Identifying Code Matrix. All objects have 2-byte lengths and 2-byte codes; some objects might be absent.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Map Name in DSIAIFRO</th>
<th>Self-Identifying Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Control Object</td>
<td>DSIMCO</td>
<td>0006</td>
</tr>
<tr>
<td>MDB Global Object</td>
<td>DSIMGO</td>
<td>0007</td>
</tr>
<tr>
<td>MDB General Object</td>
<td>DSIGOJ</td>
<td>0001</td>
</tr>
<tr>
<td>MDB Control Program Object</td>
<td>DSICPO</td>
<td>0002</td>
</tr>
<tr>
<td>Source Object</td>
<td>DSIMSO (optional)</td>
<td>0005</td>
</tr>
<tr>
<td>MDB NLS</td>
<td>(none)</td>
<td>0003</td>
</tr>
<tr>
<td>MDB Message Text Object</td>
<td>DSITOJ (optional)</td>
<td>0004</td>
</tr>
</tbody>
</table>

Notes:
1. MDB NLS is not used.
2. MDB Message text objects are not contained in DSIMCO. The data from the MDB message text objects is stored in the IFRAUTBA buffer chain.

Format of the Source Object

The source object is mapped by DSIMSO, which is contained within DSIAIFRO. The source object is used to identify the origin of an MDB and is one of the objects that can be included in a message control object. It is in the format of an SNA generalized data stream (GDS) variable that can contain other GDS variables. It begins with a 2-byte field that defines the length of the object, followed by a 2-byte code identifying it as a source object. Following that initial 4-byte descriptor, there are as many sub-objects as needed up to a maximum of 1100 bytes.

The codes for a sub-object (a part of the source object containing a name) are the following:

D5 The nickname, in character format 1–8 bytes
01 The NETID from the MDS-MU header 1–8 bytes
02 The NAU name from the MDS-MU header 1–8 bytes

Note: Characters for these fields can be from the coded graphic character set 01134–00500. Trailing EBCDIC blanks are permitted but are insignificant. This set consists of capitals A–Z and numerals 0–9, without punctuation or special symbols.

<table>
<thead>
<tr>
<th>Source Object Header</th>
<th>Sub-object 1</th>
<th>Sub-object 2</th>
<th>...</th>
<th>Sub-object n</th>
</tr>
</thead>
</table>

Figure 22. Source Object Diagram. Source Object has 2-byte length and 2-byte code. Each Sub-object has 1-byte length, 1-byte code, and data.

Use of a Source Object
The source object name identifies the origin of the message in the MDB. The name of the source is selected, following the hierarchy shown below, from the sub-objects contained in the source object:

1. The first nickname, if any. If no nickname is found, go to step 2.
2. The first NETID concatenated to a NAU name, with a period (.) between them, if both exist in sequence. If that combination is not found, go to step 3.
3. The first NAU name, if it exists. If that is not found, go to step 4.
4. Either the character string N/A if none of the above listed names exist, or null if the message has no source object vector, or if the vector data has structural damage (unacceptable length fields, for example).

DSICBH: Control Block Header
The control block header (DSICBH) precedes all NetView control blocks, except the BUFHDR and the internal function request (DSIFFR) control block. CBH identifies the length and type of control block that follows it, and the type of subtask the control block represents.

The fields in the DSICBH are as follows:

Table 24. Control Block Header Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBHID</td>
<td>1</td>
<td>A 1-character identifier of the control block. CBHPDB is the identifier for a parse descriptor block (DSIPDB).</td>
</tr>
</tbody>
</table>
### Table 24. Control Block Header Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBHTYPE</td>
<td>1</td>
<td>The type of subtask that the control block represents. (The TIB and the TVB each contain this identifier.) The types are the primary POI task (PPT), the operator station task (OST), the NetView-NetView task (NNT), the hardcopy task (HCT), a data services task (DST), and the optional subtask (OPT). Macro DSILCS uses this field for managing control work blocks (DSICWBs) and service work blocks (DSISWBs). In all other cases, this byte is reserved.</td>
</tr>
<tr>
<td>CBHHCT</td>
<td></td>
<td>The identifier used for a TIB or a TVB belonging to an HCT</td>
</tr>
<tr>
<td>CBHMNT</td>
<td></td>
<td>The identifier used for a TIB or a TVB belonging to an MNT</td>
</tr>
<tr>
<td>CBHNNT</td>
<td></td>
<td>The identifier used for a TIB or a TVB belonging to an NNT</td>
</tr>
<tr>
<td>CBHOPTSK</td>
<td></td>
<td>The identifier used for a TIB or a TVB belonging to an OPT or a DST</td>
</tr>
<tr>
<td>CBHOST</td>
<td></td>
<td>The identifier used for a TIB or a TVB belonging to an OST</td>
</tr>
<tr>
<td>CBHPPT</td>
<td></td>
<td>The identifier used for a TIB or a TVB belonging to the PPT</td>
</tr>
<tr>
<td>CBHLENG</td>
<td>2</td>
<td>A halfword that contains the length of the control block. CBHLENG represents either the length that is pre-allocated or the length that is obtained by macro DSIGET. For example, a parse descriptor block (DSIPDB) has both a fixed-size portion and a variable number of entries. For a DSIPDB, CBHLENG contains the length of both parts. This field is to be treated as a 16-bit unsigned binary (maximum 65535).</td>
</tr>
</tbody>
</table>

#### DSICWB: Command Work Block

The command work block (DSICWB) contains the command processor parameters, a save area, and a work area. The following fields are in the DSICWB:

### Table 25. Command Work Block Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWBCBH</td>
<td>4</td>
<td>A standard control block header.</td>
</tr>
<tr>
<td>CWBADATD</td>
<td>256</td>
<td>A work area for the command processor. If more storage is required, the command processor obtains it with macro DSIGET and releases it with macro DSIFRE. The command processor frees any storage it obtains.</td>
</tr>
<tr>
<td>CWBBUF</td>
<td>4</td>
<td>A pointer to a buffer containing a BUFHDR and the command text.</td>
</tr>
<tr>
<td>CWBDSRB</td>
<td>4</td>
<td>Used only by data services command processors (DSCPs). The data services task (DST) initializes this field with the address of the data services request block (DSIDSRB). This field contains 0 for all other command processor types.</td>
</tr>
<tr>
<td>CWBPARMS</td>
<td>12</td>
<td>A command processor parameter area. Its subfields are CWBBUF, CWBPDB, and CWBSWB.</td>
</tr>
</tbody>
</table>
Table 25. Command Work Block Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWBPDB</td>
<td>4</td>
<td>A pointer to a parse descriptor block (DSIPDB), which is described under <a href="#">DSIPDB: Parse Descriptor Block</a> on page 147. The PDB contains parse information for the command pointed to by CWBBUF. If a special type of parse is required, the command processor can reuse the PDB.</td>
</tr>
<tr>
<td>CWBPECB</td>
<td>4</td>
<td>A pointer to a parameter list containing the address of the posted ECB and an environment pointer. The parameter list is mapped by the DSITECBR control block. DSITECBR contains the returned parameter list when an ECB is posted on a DST list.</td>
</tr>
<tr>
<td>CWBRCODE</td>
<td>4</td>
<td>On resumption, an LRC receives a return code from a called command.</td>
</tr>
<tr>
<td>CWBSAVEA</td>
<td>72</td>
<td>A save area that the command processor can use.</td>
</tr>
<tr>
<td>CWBSWB</td>
<td>4</td>
<td>A pointer to a service work block (DSISWB) that the command processor can use or pass as a parameter to service macros or modules. Service macros build parameter lists in the SWB for the service modules. The SWB also contains a task information block (DSITIB) pointer, a parameter list, and a save area for use by the service routine. You can reuse SWBs without re-initialization. Service routines or macros need only the CBH and the TIB address.</td>
</tr>
<tr>
<td>CWBTIB</td>
<td>4</td>
<td>The address of the TIB for the subtask. The TIB and the task vector block (DSITVB) to which the TIB field TIBTVB is pointed contains all the information that relates to the subtask under which the command is running. This information contains the operator ID, the task type, and the task status.</td>
</tr>
</tbody>
</table>

DSIDSB: Data Services Block

The data services block (DSIDSB) contains information used by the disk read service routines called when the DSIDKS macro is issued. The following fields are used by the issuer of DSIDKS to access the records read:

Table 26. Data Services Block Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSBBUFF</td>
<td>4</td>
<td>The address of the I/O buffer containing the record read from the data set member. The buffer contains a BUFHDR with HDRTDISP containing the offset of the requested logical record.</td>
</tr>
<tr>
<td>DSBREC</td>
<td>4</td>
<td>The address of the requested logical record read using the DSIDKS macro.</td>
</tr>
</tbody>
</table>

DSIDSRB: Data Services Request Block

The data services request block (DSIDSRB) contains information that a data services command processor (DSCP) needs to communicate with the data services task (DST). It also contains work space for the I/O routines. The relevant fields in the DSIDSRB are:
Table 27. Data Services Request Block Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSRBCBH</td>
<td>4</td>
<td>A standard control block header.</td>
</tr>
<tr>
<td>DSRBCUSB</td>
<td>4</td>
<td>The address of a buffer used by the CNMI for unsolicited data. This field is used only when the DSRB function code (DSRBFNCD) indicates that unsolicited data has been received. The buffer contains a BUFHDR, and the data length is in the HDRMLENG field of BUFHDR.</td>
</tr>
<tr>
<td>DSRBFLG</td>
<td>1</td>
<td>The flag settings described below. The bits can be examined but not changed.</td>
</tr>
<tr>
<td>DSRBACTV = 1</td>
<td></td>
<td>An active transaction is using this DSRB. A transaction is defined as a request from the time of its first arrival at the DSCP to the last exit of the DSCP. When a transaction ends, you can reassign the DSRB to another transaction.</td>
</tr>
<tr>
<td>DSRBINUS = 1</td>
<td></td>
<td>The VSAM or the CNMI interface service routine has an active request using this DSRB. DSRBINUS is not on when DSRBACTV is off.</td>
</tr>
<tr>
<td>DSRBTYPE = 1</td>
<td></td>
<td>The DSRB is used for unsolicited CNM data.</td>
</tr>
<tr>
<td>DSRBTYPE = 0</td>
<td></td>
<td>The DSRB is used for VSAM or CNM solicited data traffic.</td>
</tr>
<tr>
<td>DSRBFLGS</td>
<td>1</td>
<td>The flag settings described below. The bits can be examined but not changed.</td>
</tr>
<tr>
<td>DSRBCPMS = 1</td>
<td></td>
<td>The alert was generated from the distributed host.</td>
</tr>
<tr>
<td>DSRBCPMS = 0</td>
<td></td>
<td>The alert came from the local CNMI.</td>
</tr>
<tr>
<td>DSRBFNCD</td>
<td>1</td>
<td>Indicates the reason that the command processor was called. The constants for DSRBFNCD are:</td>
</tr>
<tr>
<td>DSRBFNRM (1)</td>
<td></td>
<td>The first calling of the command processor, as the result of an IFRCODCR queued from another subtask using DSIMQS.</td>
</tr>
<tr>
<td>DSRBFUNS (2)</td>
<td></td>
<td>The command processor was called to handle unsolicited CNM data.</td>
</tr>
<tr>
<td>DSRBFISOl (3)</td>
<td></td>
<td>Solicited data was received from the CNMI.</td>
</tr>
<tr>
<td>DSRBFVSM (4)</td>
<td></td>
<td>A VSAM I/O request has completed.</td>
</tr>
<tr>
<td>DSRBINPT</td>
<td>4</td>
<td>The address of the CNM interface input buffer.</td>
</tr>
<tr>
<td>DSRBOID</td>
<td>8</td>
<td>The ID of the operator that initiated the transaction.</td>
</tr>
<tr>
<td>DSRBPRID</td>
<td>2</td>
<td>A halfword field that contains a correlation identifier for use by the CNM interface.</td>
</tr>
</tbody>
</table>
### Table 27. Data Services Request Block Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSRBRCMA</td>
<td>4</td>
<td>The return code for a completed request. It is set after the request is completed but before the DSCP is called again for request completion. This return code value is further explained by the minor return code (DSRBRCMI). See <a href="#">DSCP Interface</a> on page 95 for a description of the return codes.</td>
</tr>
<tr>
<td>DSRBRCMI</td>
<td>4</td>
<td>The minor return code for a completed request. See DSRBRCMA.</td>
</tr>
<tr>
<td>DSRBTIB</td>
<td>4</td>
<td>The address of the DST information block (DSITIB).</td>
</tr>
<tr>
<td>DSRBUBUF</td>
<td>4</td>
<td>The address of the original command that was sent to the DST. This field is unchanged during the data services transaction. This buffer contains a BUFHDR and the HDRMCEXT extension. It also has an X'0003' IFRCODE and HDRTYPEI. See <a href="#">DSIIFR: Internal Function Request</a> on page 125.</td>
</tr>
<tr>
<td>DSRBUSER</td>
<td>4</td>
<td>A field available for user purposes. If this field is used for a storage address, the DST does not free the storage; it is freed by the user. DSIGET Q=YES allocates storage. You can free storage by using DSIFRE Q=YES. The field DSRBUSER is set to zero (0) when the control returns to DST unless the DSRB is considered “in use” (DSRB’s field DSRBFGLG is set to DSRBINUS). If you do not free storage, the storage remains allocated but not addressable and this storage will not be freed until the subtask terminates.</td>
</tr>
<tr>
<td>DSRBVACB</td>
<td>4</td>
<td>The address of the VSAM ACB for the DST.</td>
</tr>
<tr>
<td>DSRBVDAD</td>
<td>4</td>
<td>The address of the VSAM I/O buffer with a standard BUFHDR. For GET requests, the BUFHDR HDRMLENG field indicates the length of the data read. HDRTDISP contains the offset to the data.</td>
</tr>
<tr>
<td>DSRBVECB</td>
<td>4</td>
<td>An event control block (ECB) for use by DST when requesting VSAM I/O.</td>
</tr>
<tr>
<td>DSRBVKEY</td>
<td>4</td>
<td>The address of the key in the DSRBVDAD buffer.</td>
</tr>
<tr>
<td>DSRBVKLN</td>
<td>2</td>
<td>The key length.</td>
</tr>
<tr>
<td>DSRBVRPL</td>
<td>4</td>
<td>The address of the VSAM request parameter list (RPL) that was used for the I/O.</td>
</tr>
<tr>
<td>DSRBVRTP</td>
<td>1</td>
<td>An indicator of the type of request just completed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - DSRVGET (VSAM GET)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - DSRVPUT (VSAM PUT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - DSRVPNT (VSAM POINT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 - DSRVERS (VSAM ERASE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 - DSRVNRQ (VSAM ENDREQ)</td>
</tr>
</tbody>
</table>

---

**DSIDTR: Data Transport Request**

The data transport request (DSIDTR) control block is used to pass a request to the NetView program-to-program interface (PPI). For field names and descriptions, refer to the [Tivoli NetView for z/OS Application Programmer’s Guide](#).

**DSIELB: External Logging Block**

The external logging block (DSIELB) maps the header information about the buffer passed to the XITXL exit.
The following table describes the XITXL exit fields:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ELBLENG</td>
<td>2</td>
<td>Unsigned length of DSIELB</td>
</tr>
<tr>
<td>5</td>
<td>ELBLOG</td>
<td>3</td>
<td>EBCDIC log type</td>
</tr>
<tr>
<td>2</td>
<td>ELBRLENG</td>
<td>2</td>
<td>Unsigned length of record</td>
</tr>
<tr>
<td>4</td>
<td>ELBTYPE</td>
<td>4</td>
<td>Log type</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>4</td>
<td>Reserved by the NetView program</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Start of record</td>
</tr>
</tbody>
</table>

**DSIIFR: Internal Function Request**

The internal function request (DSIIFR) is a formatted buffer that is transmitted to a subtask’s message queue using macro DSIMQS. The following is one relevant field in the DSIIFR:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRCODE</td>
<td>2 bytes</td>
<td>Most commonly specified examples are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>IFRCODAI</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A NetView automation internal function request (AIFR). See <a href="#">AIFR—Automation Internal Function Request</a> for the subfields of the AIFR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>IFRCODCR</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The remainder of the buffer is a command to run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>IFRCODPN</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The command is entered from a full-screen panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>IFRCODUS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The buffer is user-defined and is passed to DSIE13, the message receiver exit routine (applies only to an OST or NNT).</td>
</tr>
</tbody>
</table>

For a complete list of IFRCODEs, see the DSIIFR macro.

For the IFR, the value of HDRTDISP is always decimal 36, which includes a standard 24-byte BUFHDR and a 12-byte HDRMCEXT. The IFRCODE must begin immediately after the BUFHDR extension (at displacement decimal 36 as indicated by the HDRTDISP value).

**AIFR—Automation Internal Function Request**

The fields described in the following are defined starting at the location with label IFRPARMS when IFRCODE=IFRCODAI.
**Note:** The AIFR is 256 bytes, including the headers.

### Table 30. Automation Internal Function Request Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUIND</td>
<td>1 byte</td>
<td>NA</td>
<td>Contains the primary AIFR control flags. These identify optional fields and describe routing and processing actions as follows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFRAUWQE</td>
<td>1...</td>
<td>IFRAUWQE</td>
<td>See IFRAUSSI. When this flag is on, fields in IFRAUWQD are set from the WQE data received from the system. This data is also forwarded to all NetView domains by NNTs and RMTCMD autotasks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Notes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If the IFRAUTBA buffer has HDRMTYPE=HDRTYPEY and the IFRAUWQE flag is on, the message must have a 2-to 4-digit EBCDIC reply ID in front of the message number in the text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. Whenever MDB fields are present and have corresponding WQE fields, the WQE fields are set from the MDB when the AIFR is created, and IFRAUWQE is set on. This preserves the meaning of WQE data accessed by command list variables and the automation table in existing functions.</td>
</tr>
<tr>
<td></td>
<td>.1...</td>
<td>IFRAUCMD</td>
<td>Set by NetView automation table to indicate that an action is in the buffer referenced by IFRAUCMB. Each command action for a particular message has a separate AIFR structure. <strong>Note:</strong> IFRAUCMD and IFRAUACT cannot both be on at once.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>..1.</td>
<td>IFRAUACT</td>
<td>Indicates that MSG/MSU actions exist in the bits in IFRAUTA1 and IFRAUTA2. See IFRAUTA1 and IFRAUTA2 for settings controlled by IFRAUACT. IFRAUCMD and IFRAUACT cannot both be on at once. If both IFRAUCMD and IFRAUACT are 0, the buffer is subject to defaults and overrides. This bit must be on to specify force flags in IFRAUTA4.</td>
</tr>
</tbody>
</table>
Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . . 1</td>
<td>IFRAUMTB</td>
<td></td>
<td>Used to control recursion and to prevent secondary receiver and copied messages from being processed by the NetView automation table. This bit can be set to 1 to prevent the NetView automation table from processing this message. IFRAUMTB is turned off when received from an NNT to automate cross-domain messages. <strong>Note:</strong> Secondary receiver and copied messages can be automated when received as cross-domain messages. If IFRAUMTB is on, WTOs issued by the NetView program must be issued with descriptor code 13 to prevent re-automation, if the AIFR message is written to a system console.</td>
</tr>
<tr>
<td>. . . 1</td>
<td>IFRAUPPT</td>
<td></td>
<td>Indicates that a message originated in the PPT. This bit replaces the P designation in byte 7 of HDRDOMID. The P is displayed and logged only as part of the domain name but is never in the buffer header.</td>
</tr>
<tr>
<td>. . . . 1</td>
<td>IFRAUXDM</td>
<td></td>
<td>Indicates that a message was received from an NNT or an RMTCMD autotask. This bit can be used to determine if a message came from another domain.</td>
</tr>
<tr>
<td>. . . 1</td>
<td>IFRAUDOM</td>
<td></td>
<td>Indicates that this is a system delete message request. The system deletes messages by SMSGID values, by ASID and TCB address, or by ASID alone. This bit can be set only by the NetView program. If GOJGDOM is set on, there are additional types of DOMs as defined by DSIAIFRO. <strong>Note:</strong> If you use GOJGDOM, turn off IFRAUWDO. GOJGDOM indicates the message data block (MDB) form of DOM, while IFRAUWDO indicates the subsystem interface form of DOM. You keep IFRAUDOM on if either IFRAUWDO or GOJGDOM is on.</td>
</tr>
<tr>
<td>. . . 1</td>
<td>IFRAUDFL</td>
<td></td>
<td>IFRAUDFL and IFRAUDF2 are valid when IFRAUWDO is on. IFRAUDFL=OFF and IFRAUDF2=OFF means delete by SMSGID. IFRAUDFL=ON and IFRAUDF2=OFF means delete by ASID. IFRAUDFL=ON and IFRAUDF2=ON means delete by ASID and TCB address. <strong>Note:</strong> If you use IFRAUWDA or IFRAUWDT, you must set IFRAUDFL on. Otherwise, set IFRAUDFL off.</td>
</tr>
</tbody>
</table>

**IFRAUIN2** 1 byte  The second byte of indicator flags is as follows:
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUDFL</td>
<td>.1...</td>
<td>IFRAUDF2</td>
<td>IFRAUDFL and IFRAUDF2 are valid when IFRAUWDO is on. IFRAUDFL=OFF and IFRAUDF2=OFF means delete by SMSGID. IFRAUDFL=ON and IFRAUDF2=OFF means delete by ASID. IFRAUDFL=ON and IFRAUDF2=ON means delete by ASID and TCB address. <strong>Note:</strong> If IFRAUWDT is on, IFRAUDF2 must be set on. If IFRAUWDA is on, IFRAUDF2 must be off.</td>
</tr>
<tr>
<td>IFRAUEX2</td>
<td>..1...</td>
<td>Used to prevent calling DSIEX02A, DSIEX04, or DSIEX09 a second time. This bit is on when these exits are called to ensure that DSIMQS queuing of buffers from them does not cause re-drives. IFRAUEX2 is reset when received as a cross-domain message.</td>
<td></td>
</tr>
<tr>
<td>IFRAUPRI</td>
<td>...1...</td>
<td>Indicates that a message was routed using ASSIGN PRI. This is the primary copy of the message and is subject to automation. This bit replaces the indicator in HDRDOMID. Prevents ASSIGN COPY.</td>
<td></td>
</tr>
<tr>
<td>IFRAUSEC</td>
<td>...1...</td>
<td>Indicates that a message was routed using ASSIGN SEC. This is a secondary copy of the message and is not subject to automation. This bit replaces the indicator in HDRDOMID. IFRAUIMTB is also set on. Prevents ASSIGN COPY.</td>
<td></td>
</tr>
<tr>
<td>IFRAUCPY</td>
<td>.... 1...</td>
<td>Indicates that a message was routed using ASSIGN COPY. This is a secondary copy of the message and is not subject to automation. This bit replaces the indicator in HDRDOMID. IFRAUIMTB is set on also. Prevents ASSIGN COPY of copy.</td>
<td></td>
</tr>
<tr>
<td>IFRAUAUT</td>
<td>.... .1...</td>
<td>Indicates that the buffer was sent using DSIMQS using authorized receiver routing. This bit replaces the indicator in HDRDOMID.</td>
<td></td>
</tr>
<tr>
<td>IFRAUDDL</td>
<td>.... ..1.</td>
<td>Indicates that a message was received from a down-level domain. The AIFR indicators HDRDOMID and HDRSENDR are not reliable.</td>
<td></td>
</tr>
<tr>
<td>IFRAUNSL</td>
<td>.... ...1</td>
<td>Indicates that the message is routed using unsolicited receiver rules. This applies to unsolicited subsystem interface traffic, DSIMQS AUTHRCV, and unsolicited VTAM messages.</td>
<td></td>
</tr>
<tr>
<td>IFRAUWID</td>
<td>4 bytes</td>
<td>NA</td>
<td>The SMSGID value used for DOM purposes and to collect MLWTO buffers. This is a copy of write-to-operator queue element (WQE) data. <strong>Note:</strong> If you use IFRAUWI, set IFRAUWID to the same value as IFRAUWI.</td>
</tr>
</tbody>
</table>
Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUTCB</td>
<td>4 bytes</td>
<td>NA</td>
<td>The job step TCB address for the issuer of the write-to-operator (WTO). It is used as a correlation value and cannot be an address in the current memory (or machine).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use IFRAUWJT, set IFRAUTCB to the same value as IFRAUWJT.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. If you use CPOCTCB, set IFRAUTCB and IFRAUWJT to the same value.</td>
</tr>
<tr>
<td>IFRAUASI</td>
<td>2 bytes</td>
<td>NA</td>
<td>The address space ID for the issuer of the WTO. It is used as a correlation value and cannot be an address space in this machine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use IFRAUWAS, set IFRAUASI to the same value as IFRAUWAS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. If you use CPOCASID, set IFRAUASI and IFRAUWAS to the same value.</td>
</tr>
<tr>
<td>IFRAUSR2</td>
<td>2 bytes</td>
<td>NA</td>
<td>Provided for the user. It is initialized to zeros when created by the NetView program and is copied into all copies of the original. It is not changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. You must have a BUFHDR and HDRMCEXT on all buffers chained to IFRAUTBA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. If you use IFRAUMVI, you must also have a presentation vector as mapped by the HDRMDBVI field within BUFHDR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. IFRAUSR2 and IFRAUSB2 refer to the same user field in the message, but return the value in different formats.</td>
</tr>
<tr>
<td>IFRAUTBA</td>
<td>4 bytes</td>
<td>NA</td>
<td>The pointer to data buffers. These are queued using HDRNEXTM to point to the next one. An entire MLWTO or title-line message is chained to this field. The buffers must have standard NetView buffer headers and message buffer header extensions. If IFRAUMVI is on, there must also be a presentation vector as mapped by HDRMDBVI in BUFHDR. See BUFHDR within the DSITIB macro.</td>
</tr>
<tr>
<td>IFRAUTBL</td>
<td>4 bytes</td>
<td>NA</td>
<td>Points to the last buffer in the chain and is primarily intended for allowing buffers to be added to the end of a message without scanning the chain looking for the end.</td>
</tr>
</tbody>
</table>
### Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUTA1</td>
<td>1 byte</td>
<td>NA</td>
<td>IFRAUTA1 and IFRAUTA2 provide the NetView automation table processing values. They can be set by DSIEX02A, but are subject to NetView automation table and OVERRIDE command changes. IFRAUACT must be ON and IFRAUCMD must be OFF for these actions to take effect for a message. See the IFRAUIND field notes for additional IFRAUACT information. The force flags in field IFRAUTA4 control whether OVERRIDE settings can alter the effect of the IFRAUTA1 and IFRAUTA2 indicators.</td>
</tr>
</tbody>
</table>

- **HOLD**: default
- **YES**: HOLD - YES
- **NO**: HOLD - NO
- **Reserved**: 11

- **Message from MVS**: 10

- **SYSLOG**: default
- **YES**: SYSLOG - YES
- **NO**: SYSLOG - NO
- **Reserved**: 11

- **NETLOG**: default
- **YES**: NETLOG - YES
- **NO**: NETLOG - NO
- **YES, IND, CLS**: NETLOG - YES, IND, CLS

**Notes:**

1. To specify a setting for HOLD, you must use the first two bits of IFRAUTA1 in conjunction with IFRAUACT. The GOJGHOLD setting, included in the AIFR extension mapped by macro DSIAIFRO, shows the HOLD setting of the message as received from an MVS extended console, and is not used by the NetView program after its initial examination when the message is placed in an AIFR.

2. When an automated message is sent across domains over an OST-NNT session, all automation table processing values are reset except for HOLD, DISPLAY, and BEEP.
### Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUTA2</td>
<td>1 byte</td>
<td>NA</td>
<td>Option bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00.. ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10.. ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01.. ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11.. ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>..00 ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>..10 ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>..01 ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>..11 ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... 00..</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... 10..</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... 01..</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... 11..</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... ...1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... ...1</td>
</tr>
</tbody>
</table>

**Note:** The IFRAUTA2 BEEP flags must be set when the AIFR is created to BEEP - YES if any of the MDB beep indicators are on. See the HDRSNALM field within BUFHDR. After the AIFR is made, the OVERRIDE flag, IFRAUTA2 flags, force flags, and DEFAULTS flag determine if the terminal beeps. Because IFRAUTA1 and IFRAUTA2 are related, see also the usage notes for IFRAUTA1.

<table>
<thead>
<tr>
<th>IFRAUTA3</th>
<th>1 byte</th>
<th>NA</th>
<th>More display flags, as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1... ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1... ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>..1. ....</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...1 ....</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length or Mask</td>
<td>Subfield or EQU</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>.... 1...</td>
<td>IFRAUWAT</td>
<td></td>
<td>Indicates that a message has satisfied a wait in a command list. (This bit indication replaces changing HDRM TYPE to HDRTYPWT, which obscured the HDRM TYPE value.) DSIEX16 is driven with the message, and upon return the NetView program frees the buffer structures if all the display and log action indicators still say no-display, no-log, and suppress-override options. The driving of DSIEX16 for wait-suppressed messages is intended for accounting purposes, although resetting the indicators enables logging of wait-suppressed messages, for example.</td>
</tr>
<tr>
<td>.... .1..</td>
<td>IFRAUX16</td>
<td></td>
<td>Indicates that DSIEX16 has been driven and prevents DSIEX16 from being driven again for this message. This bit is set to 0 when received on a cross-domain session to allow processing in the new domain.</td>
</tr>
<tr>
<td>.... ..1.</td>
<td>IFRAUVSE</td>
<td></td>
<td>Indicates that the message was received in VSE format (partition ID, reply ID, message ID). This bit is not reset when received cross-domain, allowing messages to retain their format integrity.</td>
</tr>
<tr>
<td>.... ...1</td>
<td>IFRAUACN</td>
<td></td>
<td>Indicates that the message is an action message. <strong>Note:</strong> If IFRAUACN is on, either IFRAUWWR and CPOMLR (if it exists) is on, or descriptor code 1, 2, 3, or 11 is on in IFRAUWDS and CPOCDESC (if it exists).</td>
</tr>
</tbody>
</table>
Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUTA4</td>
<td>1 byte</td>
<td>NA</td>
<td>The first six flags force the actions specified in IFRAUTA1 and IFRAUTA2 to be in effect regardless of the OVERRIDE command options specified for this task. If the bit is 1, the override is ignored. If the related IFRAUTA1 or IFRAUTA2 flags are B'00', the action imposed by the DEFAULTS command or initial setting applies. The IFRAUFHD and IFRAUFBP, when specified with nonzero values for the related IFRAUTA1 or IFRAUTA2 flags, cause the BEEP or HOLD action to occur regardless of the initial settings or DEFAULTS command options. These flags can be set by the NetView program and are not reset when received cross-domain, so that DSIEX02A, in the new domain, can get a true picture of the original message. The NetView program sets IFRAUFSL, IFRAUFNL, and does so in multiple-routed copies. When force flags are set before NetView automation occurs, either by the NetView program before DSIEX02A or by DSIEX02A, the automation table values for that option are ignored. IFRAUACT must be set ON to make IFRAUTA1 and IFRAUTA2 actions have an effect.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1... .. IFRAUFHD Forces HOLD actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1... .. IFRAUFSL Forces SYSLOG actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>..1 .. IFRAUFNL Forces NETLOG actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...1 .. IFRAUFHL Forces HARDCOPY actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... 1.. IFRAUFD5 Forces DISPLAY actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... .1.. IFRAUFBP Forces BEEP actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... ..1. IFRNLYES Sends this or an associated IFR to the NETLOG.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.... ....1 IFRAUFLO Displays only the first line of AIFR.</td>
</tr>
<tr>
<td>IFRAUTA5</td>
<td>1 byte</td>
<td>NA</td>
<td>These flags indicate automation settings for hardware monitor MSUs that are processed by the automation table. These flags can be accepted and changed in installation exit DSIEX16B.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1... .... IFRAUHMH An MSU had one or more matches in the automation table. The hardware monitor uses this indicator to put a %:; in the column farthest to the right in certain alert panels. Visible to exit DSIEX16B.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length or Mask</td>
<td>Subfield or EQU</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| IFRAUPES   | .1....         | IFRAUPES        | Hardware monitor settings of the ESREC filter for primary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAUPAR   | ..1....        | IFRAUPAR        | Hardware monitor settings of the AREC filter for primary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAUPOP   | ...1....       | IFRAUPOP        | Hardware monitor settings of the OPER filter for primary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAURO    | ....1...       | IFRAURO         | Hardware monitor settings of the ROUTE filter for primary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAUSES   | ....1...       | IFRAUSES        | Hardware monitor settings of the ESREC filter for secondary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAUSAR   | .... .1...     | IFRAUSAR        | Hardware monitor settings of the AREC filter for secondary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAUSOP   | .... ...1      | IFRAUSOP        | Hardware monitor settings of the OPER filter for secondary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     
|            |                |                 | Visible to exit DSIEX16B. |
| IFRAUTA6   | 1 byte         | NA              | These flags indicate automation settings for hardware monitor MSUs that are processed by the automation table. These flags can be accepted and changed in installation exit DSIEX16B. |
| IFRAUSRO   | 1... ....      | IFRAUSRO        | Hardware monitor settings of the ROUTE filter for secondary events.  
|            |                |                 | 1=PASS      
|            |                |                 | 0=BLOCK     |
| IFRAUXLO   | .1....         | IFRAUXLO        | Hardware monitor settings for the external log only.  
|            |                |                 | 1=YES       
|            |                |                 | 0=NO
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUCOL</td>
<td>...1. ....</td>
<td>IFRAUCOL</td>
<td>The automation table set the color for the MSU. 1=YES 0=NO</td>
</tr>
<tr>
<td>IFRAUXAT</td>
<td>...1 ....</td>
<td>IFRAUXAT</td>
<td>The automation table set the extended highlighter attributes for the MSU. 1=YES 0=NO</td>
</tr>
<tr>
<td>IFRAUINT</td>
<td>.... 1...</td>
<td>IFRAUINT</td>
<td>The automation table set the intensity for the MSU. 1=YES 0=NO</td>
</tr>
<tr>
<td>IFRAUBEP</td>
<td>.... .1..</td>
<td>IFRAUBEP</td>
<td>The automation table set the beep for the MSU. 1=YES 0=NO</td>
</tr>
<tr>
<td>IFRAUVEC</td>
<td>.... ...1.</td>
<td>IFRAUVEC</td>
<td>Indicates that automation vector extensions exist starting at label IFRAUVCX. Other vectors are mapped by the DSIMCO macro. The automation vector at label IFRAUVCX contains only the next extension locator so that the AIFR is fixed at 256 bytes. Any other vectors are chained from IFRAUVPT. These are mapped by DSIAIFRO.</td>
</tr>
<tr>
<td>IFRAUMVI</td>
<td>.... ...1</td>
<td>IFRAUMVI</td>
<td>Indicates that presentation vectors exist in the data buffers chained from IFRAUTBA. These presentation vectors are included in an expanded form of the BUFHDR. See the HDRMDBVI field in BUFHDR.</td>
</tr>
<tr>
<td>IFRRIDLN</td>
<td>2 bytes</td>
<td>NA</td>
<td>The length of the REPLYID from the message text. The value is 0 under the following conditions: 1. No REPLYID exists. 2. HDRMTYPE = HDRTYPEY.</td>
</tr>
<tr>
<td>IFRAUIRT</td>
<td>4 bytes</td>
<td>NA</td>
<td>Indicates an important message indicator is sent to operators listed by the routing buffer chained to IFRAUIRT. IFRAUIRT must always be zero when IFRCODAI buffers are sent cross-domain. The following bytes are found in the buffer pointed to by IFRAUIRT and are preceded by a standard 24-byte BUFHDR.</td>
</tr>
</tbody>
</table>

Field(Length) Description
IFRAUSMI(2) Status monitor indicator number
IFRAUIRC(1) Number in the route list
IFRAUIR(\*)(8) Route list values
### Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUCMB</td>
<td>4 bytes</td>
<td>NA</td>
<td>Points to an internal function request (DSIIFR) that has an IFRCODE of IFRCODCR unless the command buffer was received into the current domain through an NNT task across a NetView-to-NetView (NNT) session. A command buffer received by an NNT will have a HDRMTYPE of HDRTYPEX (cross-domain command) rather than a HDRTYPEI (DSIIFR).</td>
</tr>
<tr>
<td>IFRAUPRS</td>
<td>4 bytes</td>
<td>NA</td>
<td>Built and used only during NetView automation table processing.</td>
</tr>
<tr>
<td>IFRAUSRC</td>
<td>16 bytes</td>
<td>NA</td>
<td>Provided for 16 bytes of user data. It is set to zeros when the AIFR is created and is copied from the original buffer without change. <strong>Note:</strong> IFRAUSRC and IFRAUSC2 refer to the same user field in the message, but return the value in different forms.</td>
</tr>
<tr>
<td>IFRAUSDR</td>
<td>8 bytes</td>
<td>NA</td>
<td>Used to retain the HDRSENDR value when sending AIFRs from one domain to another.</td>
</tr>
<tr>
<td>IFRAUTAF</td>
<td>8 bytes</td>
<td>NA</td>
<td>Used to retain the terminal access facility (TAF) session name when replacing it with the domain ID in AIFR HDRDOMID.</td>
</tr>
<tr>
<td>IFRAURTL</td>
<td>4 bytes</td>
<td>NA</td>
<td>The address of the route action list used only during calls to DSIEX16. The routing list is mapped by IFRAURTB. The IFRAUAOI bit determines whether the first or all active tasks in the list specified are sent to the buffer.</td>
</tr>
<tr>
<td>IFRAUTBC</td>
<td>NA</td>
<td>NA</td>
<td>Indicates the MQS receipt of a computed message.</td>
</tr>
<tr>
<td>IFRAUTBN</td>
<td>NA</td>
<td>NA</td>
<td>Indicates the MQS receipt of an abended message.</td>
</tr>
<tr>
<td>IFRAULBR</td>
<td>NA</td>
<td>NA</td>
<td>The last buffer received.</td>
</tr>
<tr>
<td>IFRAUIN3</td>
<td>4 bytes</td>
<td>NA</td>
<td>The indicator flags are as follows:</td>
</tr>
<tr>
<td>IFRAUI3X</td>
<td></td>
<td></td>
<td>IFRAUPTS: NNT to OST test priority.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IFRAUPHI and IFRAUPLO indicate the queuing priority of a command sent from an NNT task to its associated OST.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If neither bit is ON, a default of LOW for a regular CMD and HIGH for an IMMEDIATE or BOTH command is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If both bits are ON, the CMD is processed at the command priority of the receiving OST. See the TEST option of DSIMQS on page 211 for more information. <strong>Note:</strong> Test for IFRAUPTS before testing the IFRAUPHI or IFRAUPLO bits separately to avoid errors.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length or Mask</td>
<td>Subfield or EQU</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1... .</td>
<td>IFRAUPHI</td>
<td></td>
<td>NNT to OST high-command priority.</td>
</tr>
<tr>
<td>.1... .</td>
<td>IFRAUPO</td>
<td></td>
<td>NNT to OST low-command priority.</td>
</tr>
<tr>
<td>00... .</td>
<td></td>
<td>IFRAUPHI and IFRAUPO OFF - Use default priority</td>
<td></td>
</tr>
<tr>
<td>10... .</td>
<td></td>
<td>IFRAUPHI ON, IFRAUPO OFF - Use HIGH priority</td>
<td></td>
</tr>
<tr>
<td>01... .</td>
<td></td>
<td>IFRAUPHI OFF, IFRAUPO ON - Use LOW priority</td>
<td></td>
</tr>
<tr>
<td>11... .</td>
<td></td>
<td>IFRAUPHI and IFRAUPO ON - Use receiving OST's priority</td>
<td></td>
</tr>
<tr>
<td>..1... .</td>
<td>IFRAUPMX</td>
<td></td>
<td>Indicates the message was received from VM PMX using the external form of DSIMQS.</td>
</tr>
<tr>
<td>...1...</td>
<td>IFRAUNVD</td>
<td></td>
<td>NetView held message. When IFRAUNVD is set in a DOM, it is a NetView internal DOM and other DOM fields are not valid. In NetView internal DOMs, the combination of domain ID (HDRDOMID) and message time value (IFRAUGMT) uniquely identify the message to be deleted.</td>
</tr>
<tr>
<td>.... 1...</td>
<td>IFRAUCLR</td>
<td></td>
<td>Clear the screen after all queued messages are sent. IFRAUCLR causes the screen to be cleared so that the message is at the top of the screen output area. If IFRAUPSH or IFRAULCK are on, the screen is locked or held and then cleared when the lock of HELD condition is completed. Messages which were issued prior to this one are displayed prior to the lock, HELD or Clear screen occurs.</td>
</tr>
<tr>
<td>.... .1...</td>
<td>IFRAUPSH</td>
<td></td>
<td>Lock and HOLD screen after all pending normal queued messages are sent. IFRAUPSH causes the screen to be held after all previous messages are written. Any buffer queued to IFRAUTBA is displayed in the area indicated by IFRAUIMD after the HOLD is complete. Supersedes IFRAULCK.</td>
</tr>
<tr>
<td>.... .1...</td>
<td>IFRAULCK</td>
<td></td>
<td>Lock screen using normal locking rules after all pending normal queued messages are sent. Ignored if IFRAUPSH is on. IFRAULCK causes the screen to be locked after all previous messages are written. Any buffer queued to IFRAUTBA is displayed in the area indicated by IFRAUIMD after the lock is complete. Ignored if IFRAUPSH is on.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length or Mask</td>
<td>Subfield or EQU</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IFRAUIMD</td>
<td>. . . . 1</td>
<td>IFRAUIMD</td>
<td>Message is written to the immediate message area after all pending normal messages RESERVED for IFRAUIN3. IFRAUIMD indicates that a message is written to the immediate message area. If the message was issued using DSIPSS TYPE=IMMED on an OST, the message appears immediately. If issued by DSIPSS TYPE=OUTPUT, the message appears in the immediate area after all pending queued messages. A possible use is to display a message describing a held condition when using IFRAUPSH.</td>
</tr>
<tr>
<td>IFRAUNSB</td>
<td>. . . . 1</td>
<td>IFRAUNSB</td>
<td>Command will not break the message stifle setting of the OST. IFRAUNSB causes all command output messages to be suppressed if the operator is in a non-interruptible display mode. The output messages are displayed when the operator switches back to the NCCF panel.</td>
</tr>
<tr>
<td>IFRAUDMA</td>
<td>1 . . .</td>
<td>IFRAUDMA</td>
<td>Automate the removal of the message. The message will redrive the automation table when it is removed from the held queue.</td>
</tr>
<tr>
<td>IFRAUDMN</td>
<td>. . . . 1</td>
<td>IFRAUDMN</td>
<td>A DOM is not expected and the queue element that waits for the DOM should be removed.</td>
</tr>
<tr>
<td>IFRAUDMA</td>
<td>00 . . .</td>
<td>IFRAUDMA, IFRAUDMN</td>
<td>Normal DOM processing with no automation when message is deleted.</td>
</tr>
<tr>
<td>IFRAUDTO</td>
<td>. . . . 1</td>
<td>IFRAUDTO</td>
<td>MVS DOM-by-Token.</td>
</tr>
<tr>
<td>IFRAUDLO</td>
<td>. . . . 1</td>
<td>IFRAUDLO</td>
<td>NetView LOCAL delete.</td>
</tr>
<tr>
<td>IFRAUSIR</td>
<td>. . 1 . .</td>
<td>IFRAUSIR</td>
<td>Duplicate AUTO(YES) message used for DOM routing. See &quot;DSIEX17: MVS Message and DOM Receive&quot; on page 48 for more information.</td>
</tr>
<tr>
<td>IFRAUWQD</td>
<td>88 bytes</td>
<td>NA</td>
<td>Maps the WQE data that has been processed into a format independent of levels of MVS and JES.</td>
</tr>
<tr>
<td>IFRAUWHD</td>
<td>4 bytes</td>
<td>NA</td>
<td>Eye catcher 'MSG' or 'DOM'.</td>
</tr>
<tr>
<td>IFRAUWFI</td>
<td>1 byte</td>
<td>NA</td>
<td>Flags 1:</td>
</tr>
<tr>
<td>IFRAUWFR</td>
<td>1 . . . .</td>
<td>IFRAUWFR</td>
<td>MLWTO first. <strong>Note:</strong> This bit is not useful. See the IFRAUTBA chain.</td>
</tr>
<tr>
<td>IFRAUWMD</td>
<td>. . . . 1</td>
<td>IFRAUWMD</td>
<td>MLWTO middle. <strong>Note:</strong> This bit is not useful. See the IFRAUTBA chain.</td>
</tr>
<tr>
<td>IFRAWLS</td>
<td>. . . . 1</td>
<td>IFRAWLS</td>
<td>MLWTO last. <strong>Note:</strong> This bit is not useful. See the IFRAUTBA chain.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length or Mask</td>
<td>Subfield or EQU</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>...1 ....</td>
<td>IFRAUWDO</td>
<td></td>
<td>DOM. Note: If you use GOJGDOM, turn off IFRAUWDO. GOJGDOM indicates the MDB format of DOM, while IFRAUWDO indicates the subsystem interface form of DOM. If GOJGDOM is on, turn on IFRAUDOM.</td>
</tr>
<tr>
<td>.... 1...</td>
<td>IFRAUWSI</td>
<td></td>
<td>Single message. Note: This bit is not useful. See the IFRAUTBA chain.</td>
</tr>
<tr>
<td>.... .1..</td>
<td>IFRAUWWR</td>
<td></td>
<td>WTOR. Note: If you use CPOMLR, set IFRAUWWR to the same value as CPOMLR. If IFRAUWWR is on, turn on IFRAUACN.</td>
</tr>
<tr>
<td>.... .1..</td>
<td>IFRAUWSP</td>
<td></td>
<td>Message suppressed.</td>
</tr>
<tr>
<td>.... .1..</td>
<td>IFRAUWBD</td>
<td></td>
<td>Broadcast to all. Note: If you use CPOMLBC, set IFRAUWBD to the same value as CPOMLBC.</td>
</tr>
<tr>
<td>IFRAUWF2</td>
<td>1 byte</td>
<td></td>
<td>Flags 2:</td>
</tr>
<tr>
<td>1... ....</td>
<td>IFRAUWJN</td>
<td></td>
<td>Display job names. Note: If you use CPOMSGTA, set IFRAUWJN to the same value as CPOMSGTA.</td>
</tr>
<tr>
<td>.1... ....</td>
<td>IFRAUWST</td>
<td></td>
<td>Display status. Note: If you use CPOMSGTB, set IFRAUWST to the same value as CPOMSGTB.</td>
</tr>
<tr>
<td>.... .1..</td>
<td>IFRAUWSS</td>
<td></td>
<td>Display session. Note: If you use CPOMSGTF, set IFRAUWSS to the same value as CPOMSGTF.</td>
</tr>
<tr>
<td>IFRAUWF3</td>
<td>1 byte</td>
<td>NA</td>
<td>Flags 3:</td>
</tr>
<tr>
<td>1... ....</td>
<td>IFRAUWTA</td>
<td></td>
<td>Control line. Note: This bit is not useful. See the HDRLNCTL field in BUFHDR.</td>
</tr>
<tr>
<td>.1... ....</td>
<td>IFRAUWTB</td>
<td></td>
<td>Label line. Note: This bit is not useful. See the HDRLNLBL field in BUFHDR.</td>
</tr>
<tr>
<td>..1... ....</td>
<td>IFRAUWTC</td>
<td></td>
<td>Data line. Note: This bit is not useful. See the HDRLNDAT field in BUFHDR.</td>
</tr>
<tr>
<td>...1 ....</td>
<td>IFRAUWTD</td>
<td></td>
<td>End line. Note: This bit is not useful. See the HDRLNEND field in BUFHDR.</td>
</tr>
<tr>
<td>IFRAUWF4</td>
<td>1 byte</td>
<td></td>
<td>Flags 4:</td>
</tr>
<tr>
<td>1... ....</td>
<td>IFRAUWDT</td>
<td></td>
<td>DOM by ASID and TCB Note: If you use IFRAUWDT, set IFRAUDFL and IFRAUDF2 to corresponding meanings. The bits do not correspond one-for-one.</td>
</tr>
<tr>
<td>.1... ....</td>
<td>IFRAUWDA</td>
<td></td>
<td>DOM by ASID. Note: If you use IFRAUWDA, set IFRAUDF2 and IFRAUDFL to corresponding meanings. The bits do not correspond one-for-one.</td>
</tr>
</tbody>
</table>
Table 30. Automation Internal Function Request Fields  (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or</th>
<th>Subfield or</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUMCS</td>
<td>2 bytes</td>
<td>NA</td>
<td>MCS flags.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. Set IFRAUMCS(1) on if any of the following have nonzero bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPOCERC or IFRAUWRT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPODESC or IFRAUWDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. If you use CPOCMCSC, set IFRAUMCS(3) to the same value as CPOCMCSC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. If you use CPOMLBC, set IFRAUMCS(6) to the same value as CPOMLBC.</td>
</tr>
<tr>
<td>IFRAUNVF</td>
<td>1 byte</td>
<td>NA</td>
<td>NetView-specific.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> These 3 flags correspond to 3 flags defined in the MVS WQE control block when NetView is using the SSI interface, and correspond to 3 similar flags in the MDB when running Extended Console Mode. The exact meaning and use of the flags is a property of the operating system. You should refer to operating system documentation for specific details.</td>
</tr>
<tr>
<td>IFRAURET</td>
<td></td>
<td></td>
<td>AMRF retained message</td>
</tr>
<tr>
<td>IFRAURRT</td>
<td></td>
<td></td>
<td>Retain in AMRF</td>
</tr>
<tr>
<td>IFRAUNRT</td>
<td></td>
<td></td>
<td>Do not retain in AMRF</td>
</tr>
<tr>
<td>IFRAUNVO</td>
<td></td>
<td></td>
<td>NetView-designated.</td>
</tr>
<tr>
<td>IFRAUFL5</td>
<td>1 byte</td>
<td>NA</td>
<td>Flags 5:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.1. . . . . .</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IFRAURET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AMRF retained message</td>
</tr>
<tr>
<td>IFRAUWMA</td>
<td>1 byte</td>
<td>NA</td>
<td>Area ID.</td>
</tr>
<tr>
<td>IFRAUWDS</td>
<td>2 bytes</td>
<td>NA</td>
<td>Descriptor codes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use CPOCDESC, set IFRAUWDS to the same value as CPOCDESC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. You must set IFRAUMCS(1) on if any of the following have nonzero bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPOCERC or IFRAUWRT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPODESC or IFRAUWDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. If descriptor code 1, 2, 3, or 11 is on in IFRAUWDS or CPODESC, set IFRAUACN on.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4. If you use CPOMLIA, set IFRAUWDS(2) to the same value as CPOMLIA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5. If you use CPOMLCE, set IFRAUWDS(3) to the same value as CPOMLCE.</td>
</tr>
<tr>
<td>IFRAUWAS</td>
<td>2 bytes</td>
<td>NA</td>
<td>ASID of issuer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use IFRAUWAS, set IFRAUASI to the same value as IFRAUWAS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. If you use CPOCASID, set IFRAUWAS and IFRAUASI to the same value.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length or Mask</td>
<td>Subfield or EQU</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IFRAUWJT</td>
<td>4 bytes</td>
<td>NA</td>
<td>JSTCB of issuer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use IFRAUWJT, set IFRAUTCB to the same value as IFRAUWJT.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. If you use CPOCTCB, set IFRAUWJT and IFRAUTCB to the same value.</td>
</tr>
<tr>
<td>IFRAUWWI</td>
<td>4 bytes</td>
<td>NA</td>
<td>SMSGID value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> If you use IFRAUWWI, set IFRAUWID to the same value as IFRAUWWI.</td>
</tr>
<tr>
<td>IFRAUWUC</td>
<td>4 bytes</td>
<td>NA</td>
<td>Target console UCMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use CPOCCNID, set IFRAUWUC to the same value as CPOCCNID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. When setting IFRAUWUC to a value, set IFRAUCON to the corresponding MVS console name or number.</td>
</tr>
<tr>
<td>IFRAUWSN</td>
<td>8 bytes</td>
<td>NA</td>
<td>System name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> If you use GOJGOSNM, set IFRAUWSN to the same value.</td>
</tr>
<tr>
<td>IFRAUWRT</td>
<td>16 bytes</td>
<td>NA</td>
<td>Route codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. If you use CPOCERC, set IFRAUWRT to the same value as CPOCERC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. Set IFRAUMCS(1) on if any of the following have nonzero bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPOCERC or IFRAUWRT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPODESC or IFRAUWDS.</td>
</tr>
<tr>
<td>IFRAUWTL</td>
<td>4 bytes</td>
<td>NA</td>
<td>Text length</td>
</tr>
<tr>
<td>IFRAUWJU</td>
<td>8 bytes</td>
<td>NA</td>
<td>Job number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> If you use CPOCOJID, set IFRAUWJU to the same value as CPOCOJID.</td>
</tr>
<tr>
<td>IFRAUWJA</td>
<td>8 bytes</td>
<td>NA</td>
<td>Job name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> If you use CPOCOJBN, set IFRAUWJA to the same value as CPOCOJBN.</td>
</tr>
<tr>
<td>IFRAUTOK</td>
<td>8 bytes</td>
<td>NA</td>
<td>MVS message processing facility token.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> If you use CPOCAUTO, set IFRAUTOK to the same value.</td>
</tr>
<tr>
<td>IFRAUGMT</td>
<td>8 bytes</td>
<td>NA</td>
<td>The Greenwich Mean Time store clock value at the time the AIFR was created. This value is preserved on all subsequent copies of the original information, and is preserved on cross-domain sessions. You can set this field only once for each AIFR.</td>
</tr>
<tr>
<td>IFRAUCON</td>
<td>8 bytes</td>
<td>NA</td>
<td>The EBCDIC decimal name or number of the console. It is the name associated with the console number in IFRAUWUC.</td>
</tr>
<tr>
<td>IFRAUVCX</td>
<td>16 bytes</td>
<td>NA</td>
<td>The AIFR extension object.</td>
</tr>
<tr>
<td>IFRAUVAS</td>
<td>NA</td>
<td>NA</td>
<td>The next extension locator ASID (ALET).</td>
</tr>
</tbody>
</table>
### Table 30. Automation Internal Function Request Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length or Mask</th>
<th>Subfield or EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAUVPT</td>
<td>NA</td>
<td></td>
<td>The next extension locator address.</td>
</tr>
</tbody>
</table>

**Notes:**
1. The AIFR length is fixed at 256 bytes, including the BUFHDR and HDRMCEXT. Do not expand it.
2. The length of IFRRIDLN is zero when there is not a reply ID or when HDRMTYPE is not HDRTYPEY.

### Automation Internal Function Request Routing List

IFRAURTBL is the mapping of the route action list that is used during DSIEX16 processing. IFRAURTBL is the address of a standard NetView buffer with a standard buffer header and a standard buffer header extension. The route list within this buffer contains a list of names that get identical copies of the IFRCODAI structure. Bit IFRAUAOI determines whether all active or the first active name receives a copy of the IFRCODAI. HDRMSG is a field within BUFHDR that is part of the DSITIB control block macro. This buffer is addressed separately from the IFRCODAI buffer. HDRBLENG, HDRMLENG, and HDRTDISP are the only fields that are initialized in this buffer header.

The following are relevant fields:

### Table 31. Automation Internal Function Request Routing List Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFRAURCC</td>
<td>2 bytes</td>
<td>A halfword count of the number of route names that follow. The HDRBLENG value must be sufficient to contain this number of names plus the buffer headers.</td>
</tr>
<tr>
<td>IFRAURCI</td>
<td>10 bytes</td>
<td>A variable-sized array of 10-character names. These must contain an operator ID of no more than 8 characters padded with blanks to a length of 10. Multiple 10-character fields continue here.</td>
</tr>
</tbody>
</table>

### Usage Notes

Setting the message action flags in the buffer in DSIEX02A is equivalent to specifying the options in the NetView automation table. If the message is selected by the NetView automation table, any options specified by the table overlay the DSIEX02A values. After DSIEX02A and table processing, the message is evaluated against criteria specified by DEFAULTS and OVERRIDE commands to determine display and logging actions. IFRAUACT must be set to 1 if any ACTIONS are selected by DSIEX02A. When obtaining storage for an AIFR, separate invocations of DSIGET are used for each buffer chained off the AIFR. This includes the buffers in the IFRAUTBA chain. For each DSIGET, specify nonqueued subpool zero storage.

In the BUFHDR that precedes the IFR:
1. HDRMTYPE is specified as I (HDRMTYPE=X'C9'; symbol HDRTYPEI).
2. For a DSIIFR received on a task message queue, HDRTDISP contains the displacement to the IFRCODE. For IFRCODCR and IFRCODUS, the NetView program modifies HDRTDISP and HDRMLENG so that all commands appear
the same to the command processor. The command verb is followed by the parameters. The IFR section is logically removed.

**DSILOGDS: NetView Log**

The NetView log maps the record written to the network log. The fields in the DSILOGDS are:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGDATE</td>
<td>4</td>
<td>Date of record; format is 0CyydddC, where yy is the year, ddd is the day, and C defines the field as packed decimal.</td>
</tr>
<tr>
<td>LOGDISP</td>
<td>2</td>
<td>Record text displacement.</td>
</tr>
<tr>
<td>LOGDOMID</td>
<td>8</td>
<td>Domain ID of record originator.</td>
</tr>
<tr>
<td>LOGKEYDT</td>
<td>4</td>
<td>Same as LOGDATE.</td>
</tr>
<tr>
<td>LOGKEYTM</td>
<td>4</td>
<td>Time of record; format is hhmmss0c, where hh is the hour, mm is the minute, ss is the second, and 0c defines this field as packed decimal.</td>
</tr>
<tr>
<td>LOGLUNAM</td>
<td>8</td>
<td>Task name of record originator.</td>
</tr>
<tr>
<td>LOGMTCHE</td>
<td>1</td>
<td>Message type of record.</td>
</tr>
<tr>
<td>LOGOPID</td>
<td>8</td>
<td>Operator ID of record originator.</td>
</tr>
<tr>
<td>LOGSEQ#</td>
<td>4</td>
<td>Sequence number for VSAM key.</td>
</tr>
<tr>
<td>LOGTEXT</td>
<td></td>
<td>Text of network log record.</td>
</tr>
<tr>
<td>LOGTIME</td>
<td>4</td>
<td>Same as LOGKEYTM.</td>
</tr>
</tbody>
</table>

**DSIMVT: Main Vector Table**

The main vector table (DSIMVT) is the main control block for information throughout NetView. It contains global information, such as the domain name, the status of NetView, and pointers to other tables and subtasks.

Each NetView program has one MVT. You can locate the MVT from a subtask through the TVBMVT, a pointer in the task vector block (DSITVB).

The following are relevant fields in the MVT:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVTART</td>
<td>2</td>
<td>In prior releases, a pointer to DSIART. The contents of this field are set to zero. This symbolic name has been removed from DSIMVT, and the field usage changed to reserved to preserve other field offsets in the MVT.</td>
</tr>
<tr>
<td>MVTARTLN</td>
<td>2</td>
<td>In prior releases, the length of each entry in DSIART. The contents of this field are set to zero. This symbolic name has been removed from DSIMVT, and the field usage changed to reserved to preserve other field offsets in the MVT.</td>
</tr>
<tr>
<td>MVTCBH</td>
<td>4</td>
<td>A standard control block header.</td>
</tr>
<tr>
<td>MVTCDSSES</td>
<td>2</td>
<td>Obsolete.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MVTCLOSE</td>
<td>1</td>
<td>A flag bit indicating the CLOSE command has been issued. The state of the bit can be checked with the NVCLOSE automation table condition item.</td>
</tr>
<tr>
<td>MVTCURAP</td>
<td>9</td>
<td>The name of the NetView domain as defined by the started procedure or CNMSTYLE, as follows:</td>
</tr>
<tr>
<td>MVTCURAL</td>
<td></td>
<td>1-byte field that shows the length of the domain name(1–5 characters).</td>
</tr>
<tr>
<td>MVTCURAN</td>
<td></td>
<td>8-byte field that contains the domain name padded with blanks.</td>
</tr>
<tr>
<td>MVTDRETRY</td>
<td>2</td>
<td>The number of times an I/O operation is retried before it becomes a permanent error.</td>
</tr>
<tr>
<td>MVTGFMG1</td>
<td>4</td>
<td>A pointer to a write-to-operator parameter list containing the message DS1124I STORAGE REQUEST FAILED FOR NCCF. This message can be used by any WTO macro with MF=E. No additional storage is required. The routing code is (2,11); the descriptor code is 11.</td>
</tr>
<tr>
<td>MVTGFMG2</td>
<td>4</td>
<td>A pointer to a write-to-operator parameter list containing the message DS1125I CRITICAL STORAGE SHORTAGE FOR NCCF. This message can be used by any WTO macro with MF=E. No additional storage is required. The routing code is (2,11); the descriptor code is 11.</td>
</tr>
<tr>
<td>MVTGMSG</td>
<td>4</td>
<td>Pointer to a buffer containing the message DSI073A COMMAND PROCESSOR UNABLE TO BUILD RESPONSE MESSAGE.</td>
</tr>
<tr>
<td>MVTMETH</td>
<td>1</td>
<td>Indicates that the access method is VTAM (V).</td>
</tr>
<tr>
<td>MVTMLGON</td>
<td>2</td>
<td>The number of times incorrect logon information is processed before that terminal session ends. This number is specified in the MAXLOGON definition statement.</td>
</tr>
<tr>
<td>MVTMRC</td>
<td>2</td>
<td>The number of times an OST or PPT can end abnormally before it is terminated. This number is specified in the MAXABEND definition statement. The MAXABEND count for a task will be reset to zero if the task has run for at least one hour since the last abend.</td>
</tr>
<tr>
<td>MVTNAUNM</td>
<td>8</td>
<td>The correct origin LU name (either VTAM CP name or NetView LU name) while the DS16DST task is active. If DS16DST is not active, this field is blank. This field is useful for building complete MDS-MUs.</td>
</tr>
<tr>
<td>MVTNCCFQ</td>
<td>8</td>
<td>The QNAME value for macros ENQ and DEQ.</td>
</tr>
<tr>
<td>MVTNETID</td>
<td>8</td>
<td>The name of the network ID.</td>
</tr>
<tr>
<td>MVTOIT</td>
<td>4</td>
<td>In prior releases, a pointer to DSIOIT. The contents of this field are set to zero. This symbolic name has been removed from DSIMVT, and the field usage changed to reserved to preserve other field offsets in the MVT.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MVTPRDSF</td>
<td>16</td>
<td>Three common architected product-unique subfields that are used to communicate information to other products. For example, this release of NetView stores 005697B82NETVIEW in this field.</td>
</tr>
<tr>
<td>MVTMODNU</td>
<td></td>
<td>Two-character field that contains the common modification identifier. MVTMODNU used with MVTVER will also identify the version and release.</td>
</tr>
<tr>
<td>MVTPRNUM</td>
<td></td>
<td>Seven-character field that contains the program product number.</td>
</tr>
<tr>
<td>MVTPRNAM</td>
<td></td>
<td>Seven-character field that contains the software product common name.</td>
</tr>
<tr>
<td>MVTRCNT</td>
<td>4</td>
<td>The total number of unique specific and wildcard resource names defined in the span of control table. This count does not include view names defined in the NetView span table.</td>
</tr>
<tr>
<td>MVTSCNT</td>
<td>2</td>
<td>The total number of span names defined for span of control. This count does not include span names defined in the NetView span table that only contain view identifiers.</td>
</tr>
<tr>
<td>MVTSNT</td>
<td>4</td>
<td>In prior releases, a pointer to DSISNT. The contents of this field are set to zero. This symbolic name has been removed from DSIMVT, and the field usage changed to reserved to preserve other field offsets in the MVT.</td>
</tr>
<tr>
<td>MVTSNNTLN</td>
<td>2</td>
<td>In prior releases, the length of each entry in DSISNT. The contents of this field are set to zero. This symbolic name has been removed from DSIMVT, and the field usage changed to reserved to preserve other field offsets in the MVT.</td>
</tr>
<tr>
<td>MVTSPANA</td>
<td>1</td>
<td>In prior releases, a flag bit indicating that the authorization and routing table (DSIART) resides above the 16Mb line. The contents of this field are set to zero. This symbolic name has been removed from DSIMVT, and the field usage changed to reserved to preserve other field offsets in the MVT.</td>
</tr>
<tr>
<td>MVTSVL</td>
<td>4</td>
<td>The address of the service routine vector list (SVL) that contains the addresses of the service routines.</td>
</tr>
<tr>
<td>MVTSYMbam</td>
<td>4</td>
<td>The address of the MVS or NetView service routine (ASASYMBM or DSISYMBM) that performs substitution for MVS symbols. This field contains the address of the NetView service routine or zeroes if NetView symbolic substitution is disabled. Refer to the MVS/ESA library for information on how to call the ASASYMBM service routine. (The DSISYMBM service routine uses the same interface that the ASASYMBM service routine uses.)</td>
</tr>
<tr>
<td>MVTSYMbt</td>
<td>4</td>
<td>The address of the NetView logical extension to the MVS system-provided symbol table mapped by ASASYMBT.</td>
</tr>
<tr>
<td>MVTTOD</td>
<td>8</td>
<td>The system time-of-day clock when the NetView program was started.</td>
</tr>
<tr>
<td>MVTTVB</td>
<td>4</td>
<td>The address of the first TVB in the TVB chain.</td>
</tr>
<tr>
<td>MVTTVBRN</td>
<td>18</td>
<td>The RNAME value for the TVB chain.</td>
</tr>
</tbody>
</table>
Table 33. Main Vector Table (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVTUFLD</td>
<td>4</td>
<td>A user-defined field. See a description of user-defined fields on page 108.</td>
</tr>
<tr>
<td>MVTVER</td>
<td>4</td>
<td>Contains a displayable identifier for the NetView release under which your code is running.</td>
</tr>
</tbody>
</table>

**MVTIDFT—DSIEX16 Interface Data**

For field name MVTIDFT, the following defaults are set in DSIMVT and relate to DSIEX16:

Table 34. DSIMVT Defaults that Relate to DSIEX16

<table>
<thead>
<tr>
<th>Length or Mask</th>
<th>Start-up Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1... ....</td>
<td>1 = Messages can be held on the screen. 0 = Messages cannot be held on the screen.</td>
<td></td>
</tr>
<tr>
<td>.x... ....</td>
<td>Reserved.</td>
<td></td>
</tr>
<tr>
<td>.0... ....</td>
<td>0 = Messages other than NetView WTO and WTOR are not written to the system log. 1 = NetView messages are written to the system log.</td>
<td></td>
</tr>
<tr>
<td>...1 ....</td>
<td>1 = Messages are written to the NetView log if the NetView log is active. 0 = Messages are not written to the NetView log.</td>
<td></td>
</tr>
<tr>
<td>.... 1...</td>
<td>1 = Messages can be written to the hardcopy log if one is started for the operator. 0 = Messages are not written to the hardcopy log.</td>
<td></td>
</tr>
<tr>
<td>.... .1...</td>
<td>1 = Messages can be displayed on the NetView operator’s station. 0 = Messages cannot be displayed on the NetView operator’s station.</td>
<td></td>
</tr>
<tr>
<td>.... ..1...</td>
<td>1 = Messages can beep on the NetView operator’s station. 0 = Messages cannot beep on the operator’s station.</td>
<td></td>
</tr>
<tr>
<td>.... ....x</td>
<td>Reserved.</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. DISPLAY, BEEP, HOLD, and HCYLOG actions are the same in each IFRCODAI structure on the chain pointed to by the original AIFR’s HDRNEXTM field. However, the original AIFR can be different from those on the chain.
2. You can change the settings using the DEFAULTS command.
3. The interpretation of the results of the settings is affected by the settings of AIFR flags for a message and by the OVERRIDE command settings currently in effect for each operator.
4. Some messages are not affected by any of the above.
The parse descriptor block (DSIPDB) contains parse information for a command. It is pointed to by CWBPDB in the CWB or by USERPDB in the USE. The PDB has no fixed length. The following are fields in DSIPDB:

### Table 35. Parse Descriptor Block Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDBCBH</td>
<td>4</td>
<td>Standard control block header.</td>
</tr>
<tr>
<td>PDBBUFA</td>
<td>4</td>
<td>The address of the command buffer. CWBBUF also contains this address.</td>
</tr>
<tr>
<td>PDBCMDA</td>
<td>4</td>
<td>A pointer to the entry in the system command table (SCT) for the verb in the buffer that caused this command processor to be called. Macro DSIPAS (parameter alias services) and DSIKVS (keyword and value security services) use this entry as a parameter.</td>
</tr>
<tr>
<td>PDBFLAGS</td>
<td>1</td>
<td>Indicator byte for command processors. <strong>PDBIMMED</strong> 1 runs as an immediate command. 0 runs as a regular command or a data services command.</td>
</tr>
<tr>
<td>PDBNOENT</td>
<td>2</td>
<td>The number of syntactical element entries in the PDB, including the verb and all parameters.</td>
</tr>
<tr>
<td>PDBTABLE</td>
<td>0</td>
<td>Label to indicate the beginning of PDB entries.</td>
</tr>
</tbody>
</table>
### Table 35. Parse Descriptor Block Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDBENTRY</td>
<td>4</td>
<td>Each syntactical element creates one entry in this portion of the PDB. The verb is always the first entry. Each entry contains the length, the delimiter, and the offset from the beginning of the buffer, as specified in the following three fields:</td>
</tr>
</tbody>
</table>

- **Field Name**
- **Description**

**PDBDISP**
A 2-byte field specifying the offset from the start of the buffer to the first character of the \( n \)th syntactical element. For example, \( \text{addr}(n) = \text{PDBBUFA} + \text{PDBDISP}(n) \).

**PDBLENG**
A 1-byte field specifying the length of the particular syntactical element. This field does not include the length of the delimiter. When two delimiters other than blanks occur sequentially, the length is 0. Also, when two delimiters are separated by a blank or blanks, the length is 0. The offset is set to point to the second delimiter.

**PDBTYPE**
A 1-byte field specifying the delimiter character that separates this element from the next one. The standard parsing delimiters are blank, comma, period, and equal sign. The end of the record is treated as if it were delimited by a blank. Multiple blanks are treated as one blank. Blanks preceding a syntactical element are ignored.

For example,

```
verb parameter1,parameter2
```

creates the following:

1. An entry for the verb (preceding blanks are ignored)
2. An entry for `parameter1`, delimited by a comma
3. An entry for `parameter2`, delimited by a blank.

**PDBENTND**
A label at the end of PDBENTRY for use in computing the length of PDBENTRY.

---

### DSISCE: System Command Entry

The system command entry (DSISCE) contains information about a command. Macro DSICES returns the SCE of a verb or command processor. The address of the SCE is also input to macro DSIPAS.
The fields in the DSISCE are:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCECADDR</td>
<td>4</td>
<td>For MVS/ESA, this is the address of the linkage assist routine for command processors, DSICMDLD. For operating systems other than MVS/ESA, this is the address of the command processor’s entry point.</td>
</tr>
<tr>
<td>SCELNAME</td>
<td>8</td>
<td>The load module or phase name of the command processor to be called for the verb.</td>
</tr>
<tr>
<td>SCERCADR</td>
<td>4</td>
<td>The address of the command processor’s entry point for MVS/ESA. For operating systems other than MVS/ESA, this field has no meaning.</td>
</tr>
<tr>
<td>SCEVERB</td>
<td>8</td>
<td>The command verb, left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>

**DSISCT: System Command Table**

The system command table (DSISCT) contains a system command entry (DSISCE) for each command defined to the NetView program in the CMDMDL definition statement. The table contains no user fields. However, you must include this control block to get a dummy control section (DSECT) of the SCE.

**DSISVL: Service Routine Vector List**

NetView macros use the service routine vector list (DSISVL) to call the requested service routine. For example, if you run a program that issues DSIPSS, DSISVL is the means by which DSIPS1 (NetView presentation services main service routine) is found.

The SVL contains the addresses of many service routines. The SVL has no user fields. However, include this control block to use NetView services.

**DSISWB: Service Work Block**

The service work block (DSISWB) contains the parameter list for most service facilities used in user-written code.

The following are fields in DSISWB:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWBADATD</td>
<td>256</td>
<td>The automatic work area to be used for reentrant variable definition.</td>
</tr>
<tr>
<td>SWBLRCPL</td>
<td>*</td>
<td>An area where the caller builds parameter lists for DSIPUSH, DSIPOP, and DSIFIND.</td>
</tr>
<tr>
<td>SWBPLIST</td>
<td>256</td>
<td>The parameter list area.</td>
</tr>
<tr>
<td>SWBSAVEA</td>
<td>72</td>
<td>A standard save area.</td>
</tr>
<tr>
<td>SWBTIB</td>
<td>4</td>
<td>The pointer to the caller’s TIB.</td>
</tr>
<tr>
<td>MQSENTTO</td>
<td>8</td>
<td>The operator ID of the operator to whom the message was sent. Returns to the issuer if DSIMQS is issued with a LIST of type 1ST.</td>
</tr>
</tbody>
</table>
For a description of SWBLRCPL, see “DSIPUSH: Establish Long-Running Command” on page 224, “DSIPOP: Remove Long-Running Command” on page 217, and “DSIFIND: Find Long-Running Command Storage” on page 175.

**DSITECBR: Branch Table of ECB Processor Load Modules**

The branch table of the ECB processor load modules (DSITECBR) control block contains the addresses of load modules that are ECB processors for a DST. When a private or dynamic ECB is posted in a DST task list, the ECB processor in the DST branch table is called by DSITECBR.

DSITECBR also contains the environment area that is pushed down with the DSITECBS macro and returned when the ECB is posted.

The fields in the DSITECBR are:

*Table 38. Branch Table of ECB Processor Load Modules Control Block Fields*

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TECBPRMS</td>
<td>8</td>
<td>The returned label that maps the start of the parameter list pointed to by the CWBPECB field in the DSICWB control block. See “DSICWB: Command Work Block” on page 121 for more information.</td>
</tr>
<tr>
<td>TECBPTR</td>
<td>4</td>
<td>The posted ECB address.</td>
</tr>
<tr>
<td>TECBENV</td>
<td>4</td>
<td>The user’s environment that was pushed with the DSITECBS macro.</td>
</tr>
</tbody>
</table>

**DSITIB: Task Information Block**

The task information block (DSITIB) keeps information about an attached subtask. The TIB is acquired and freed by the main task.

The following DSITIB fields apply to all NetView tasks. These fields are read-only unless otherwise specified. Fields that are useful when writing optional subtasks are noted. These fields should not be modified unless you are writing an optional task.

*Table 39. Task Information Block Fields*

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCBH</td>
<td>4</td>
<td>A standard control block header. The CBHTYPE field contains the same information as the CBHTYPE field in the TVB.</td>
</tr>
<tr>
<td>TIBACB</td>
<td>4</td>
<td>A pointer to a VTAM ACB for NetView subtasks. This field can be modified by a user-written optional task.</td>
</tr>
<tr>
<td>TIBAPID</td>
<td>9</td>
<td>The VTAM application program name for a NetView subtask. This field can be modified by a user-written optional task.</td>
</tr>
<tr>
<td>TIBAPWD</td>
<td>9</td>
<td>The VTAM password for NetView subtasks. This field can be modified by a user-written optional task.</td>
</tr>
<tr>
<td>TIBAREA1</td>
<td>62</td>
<td>Pointers to other control blocks such as CWB, SWB, or PDB for NetView subtasks. This field can be modified by a user-written optional task.</td>
</tr>
</tbody>
</table>
Table 39. Task Information Block Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBBITS</td>
<td>1</td>
<td>3270 address mode support flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIB12BIT = 1 means 12-bit addresses are valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIB14BIT = 1 means 14-bit addresses are valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIB16BIT = 1 means 16-bit addresses are valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: More than one bit can be on at the same time to indicate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the allowed modes.</td>
</tr>
<tr>
<td>TIBCLRDF</td>
<td>1</td>
<td>Default color:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'  Color field not found</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'F4'  Green</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'FA'  Orange</td>
</tr>
<tr>
<td>TIBCLR#</td>
<td>1</td>
<td>Number of colors supported.</td>
</tr>
<tr>
<td>TIBEDATD</td>
<td>256</td>
<td>Reserved for NetView in NetView tasks. This area can be used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for reentrant storage in a user-written task.</td>
</tr>
<tr>
<td>TIBECBPO</td>
<td>1</td>
<td>A constant to test whether or not an ECB has been posted.</td>
</tr>
<tr>
<td>TIBELT</td>
<td>4</td>
<td>Read-only for applications in NetView tasks. A pointer to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subtask ECB list. This field can be modified by a user-written optional task.</td>
</tr>
<tr>
<td>TIBEXLST</td>
<td>4</td>
<td>Read-only for applications in NetView tasks. A pointer to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subtask VTAM EXLST. This field can be modified by a user-written optional task.</td>
</tr>
<tr>
<td>TIBFLGS</td>
<td>1</td>
<td>TIBLRCNP is superseded by the TIBSCRID field.</td>
</tr>
<tr>
<td>TIBHLITE</td>
<td>1</td>
<td>Highlight support flags.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIBBGCOL = 1 means background color is supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIBBLINK = 1 means blinking is supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIBREVRS = 1 means reverse video is supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIBREPEX = 1 means extended field mode is supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIBUNSCR = 1 means underscore is supported.</td>
</tr>
<tr>
<td>TIBMSGNM</td>
<td>8</td>
<td>The operator identifier of the subtask that issued the START TASK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>command. If the subtask was started automatically with INIT=Y, this field contains zeros.</td>
</tr>
<tr>
<td>TIBMUXIT</td>
<td>1</td>
<td>A counter used to track the level of multiple asynchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupts.</td>
</tr>
<tr>
<td>TIBNDATD</td>
<td>256</td>
<td>Reserved for NetView in NetView tasks. This area can be used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for reentrant storage in a user-written task.</td>
</tr>
<tr>
<td>TIBOSEXT</td>
<td>4</td>
<td>Reserved for NetView in NetView tasks. A pointer to an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>optional subtask extension to the TIB. The optional subtask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>releases any storage pointed to by this area at task end.</td>
</tr>
<tr>
<td>TIBOSLST</td>
<td>4</td>
<td>Reserved for NetView in NetView tasks. This field can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modified by a user-written optional task, but must be either</td>
</tr>
<tr>
<td></td>
<td></td>
<td>zero or contain the address of a user-NetView buffer. The data is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>displayed by the NetView LIST taskname command in addition to the normal LIST command data.</td>
</tr>
<tr>
<td>TIBPROFL</td>
<td>8</td>
<td>The profile name for the operator that logged on to this task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The first byte of this field contains binary zero or a blank if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NetView security options in CNMSTYLE indicate that no profile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is used. OPERSEC=SAFDEF and OPERSEC=MINIMAL are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>examples of security options that do not use NetView operator profiles.</td>
</tr>
</tbody>
</table>
Table 39. Task Information Block Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBQUERY</td>
<td>4</td>
<td>The address of a Query Reply 3270 data stream, which was read by NetView if the logmode indicated that Query was supported. The data starts with an AID byte of X'88', and is followed by Query Reply structured fields as described in the Data Stream library.</td>
</tr>
<tr>
<td>TIBSAVEE</td>
<td>72</td>
<td>A save area for subtask use. This field is reserved for NetView in NetView tasks. This field can be modified by a user-written optional task.</td>
</tr>
<tr>
<td>TIBSAVEES</td>
<td>72</td>
<td>A save area for subtask use. This field is reserved for NetView in NetView tasks. This field can be modified by a user-written optional task.</td>
</tr>
</tbody>
</table>
| TIBSCRID     | 4      | Screen identifier: TIBSCRMM  
TIBSCRSM  
3-byte screen modification type.  
TIBSCRSN  
3-byte screen serialization number. |
| TIBTVB       | 4      | A pointer to the TVB. The address of the MVT can be obtained from the TVB. MVT locates all other control blocks. |
| TIBUFLD      | 4      | This field is not referenced or changed by the NetView program. This field can be defined by the subtask. See a description of user-defined fields on page 109. |
| TIBXECB      | 4      | The NetView program uses this field as an ECB for cross-domain communication in OST and NNT. This field is reserved for NetView in NetView tasks. This field can be modified by a user-written optional task. |

---

**DSITVB: Task Vector Block**

The task vector block (DSITVB) contains information about the status of the subtask. Each subtask in NetView has one TVB. Certain service routines, such as DSIPSS, use the TVB to store control information that is important for processing their code.

The TVB contains pointers to the MVT and the TIB. You can obtain the addresses of other important control blocks from these control blocks. The TIB, an extension of the TVB, represents an active task. TVBs are chained together through the TVBNEXT field. MVTTVB points to the beginning of the chain.
The relevant fields in the DSITVB are:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVBCBH</td>
<td>4</td>
<td>A standard control block header. The CBHTYPE byte indicates the subtask type: X'00' PPT X'01' NNT X'02' OST X'03' HCT X'05' Optional subtask To distinguish the different types of optional subtasks, refer to the field TVBMODNM.</td>
</tr>
<tr>
<td>TVBMECB</td>
<td>4</td>
<td>An event control block (ECB) that notifies the subtask that a message or a queue of messages has been sent using macro DSIMQS.</td>
</tr>
<tr>
<td>TVBMECBH</td>
<td>4</td>
<td>An ECB that notifies the subtask that a high-priority message or queue of messages has been queued to the high-priority public message queue (TVBMPUBH).</td>
</tr>
<tr>
<td>TVBMECBL</td>
<td>4</td>
<td>An ECB that notifies the subtask that a low-priority message or queue of messages has been queued to the low-priority public message queue (TVBMPUBL).</td>
</tr>
<tr>
<td>TVBMPRIQ</td>
<td>4</td>
<td>The address of the normal-priority private message queue.</td>
</tr>
<tr>
<td>TVBMPMREQ</td>
<td>4</td>
<td>The address of the normal-priority private message queue.</td>
</tr>
<tr>
<td>TVBMPMREQH</td>
<td>4</td>
<td>The address of the normal-priority private message queue.</td>
</tr>
<tr>
<td>TVBMPMREQL</td>
<td>4</td>
<td>The address of the normal-priority private message queue.</td>
</tr>
<tr>
<td>TVBMPUBH</td>
<td>4</td>
<td>The address of the high-priority public message queue. See &quot;Intertask Communication&quot; on page 86 for information on processing public and private message queues.</td>
</tr>
<tr>
<td>TVBMPUBL</td>
<td>4</td>
<td>The address of the low-priority public message queue. See &quot;Intertask Communication&quot; on page 86 for information on processing public and private message queues.</td>
</tr>
<tr>
<td>TVBMPUBQ</td>
<td>4</td>
<td>The address of the normal-priority public message queue. See &quot;Intertask Communication&quot; on page 86 for information on processing public and private message queues.</td>
</tr>
<tr>
<td>TVBMTIB</td>
<td>4</td>
<td>A pointer to the TIB.</td>
</tr>
<tr>
<td>TVBNEXT</td>
<td>4</td>
<td>A pointer to the next TVB on the TVB chain. The TVB chain is anchored from MVTTVB.</td>
</tr>
<tr>
<td>TVBPRIQ</td>
<td>1</td>
<td>Priority queuing flags: TVBMM = 1 The subtask provides services for the high-priority and low-priority message queues as well as the normal queue.</td>
</tr>
<tr>
<td>TVBRESTE</td>
<td>4</td>
<td>An ECB that notifies the subtask that RESET command has been entered.</td>
</tr>
<tr>
<td>TVBTECB</td>
<td>4</td>
<td>The MVS task control block (TCB) address.</td>
</tr>
<tr>
<td>TVBTECB</td>
<td>4</td>
<td>An ECB that requests that the subtask shut down as soon as possible. Include TVBECB in every subtask ECB list. A subtask can use this ECB to shut down.</td>
</tr>
<tr>
<td>TVBTIB</td>
<td>4</td>
<td>A pointer to the TIB for the subtask.</td>
</tr>
</tbody>
</table>
The subtask uses the following bit fields. Some of these flag bits are defined by the subtask; others are defined by the main task.

**Table 41. Task Vector Block Subtask Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVBAIIFR</td>
<td>4</td>
<td>Address of AIFR buffer structure when a command is driven from the NetView automation table, the MS transport, or the NetView high-performance transport. Otherwise, the address is zero (0).</td>
</tr>
<tr>
<td>TVBHCUSE</td>
<td>4</td>
<td>Can be defined by the subtask. For an HCT, this field tracks the number of subtasks currently using the hardcopy subtask.</td>
</tr>
<tr>
<td>TVBIND1</td>
<td>1</td>
<td>Indicator byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBTERM</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A value of 1 indicates that the subtask has ended normally, and the subtask has released all resources. This bit must be supported by the subtask. If the bit is set on by the main task before attaching the subtask, a value of 1 indicates to the subtask that it has been attached for cleanup. The subtask is to release all resources and return control to the main task with this bit still set to 1.</td>
</tr>
<tr>
<td>TVBIND2</td>
<td>1</td>
<td>Indicator byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBVCLOS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This flag bit can be defined by the subtask.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBABLOG</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A 1 indicates a task is being reinitialized just after an abend.</td>
</tr>
<tr>
<td>TVBIND3</td>
<td>1</td>
<td>Indicator byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBACTV = 1</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The subtask is active. This bit is set by the subtask. While this bit is on, messages can be sent to the subtask using macro DSIMQS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBINXIT = 1</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>An IRB exit routine is running.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBLGOFF = 1</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The subtask is ending upon request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBLGON = 1</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The subtask is starting.</td>
</tr>
<tr>
<td>TVBIND4</td>
<td>1</td>
<td>Indicator byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TVBRCVRY = 1</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recovery is in progress for this subtask.</td>
</tr>
<tr>
<td>TVBLUNAM</td>
<td>8</td>
<td>The name of the task specified by the TASK statement in CNMSTYLE. This field is initialized before the subtask is attached.</td>
</tr>
</tbody>
</table>
Table 41. Task Vector Block Subtask Fields (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVBMEMNM</td>
<td>8</td>
<td>This field is initialized with the MEM parameter of the TASK definition statement in CNMSTYLE. It is the name of the member of the data set DSIPARM that contains the initialization parameters for an optional subtask.</td>
</tr>
<tr>
<td>TVBMODNM</td>
<td>8</td>
<td>The name of the module to be attached as a subtask as specified in the MOD parameter of the TASK definition statement. This field can be used to determine the type of optional subtask.</td>
</tr>
<tr>
<td>TVBOPID</td>
<td>8</td>
<td>The unique subtask identifier. This name can be the same as TVBLUNAM. TVOPID is set up by the subtask when initialization is complete.</td>
</tr>
<tr>
<td>TVBQCNT</td>
<td>4</td>
<td>If TVB374CT is on, the total number of buffers on the public message queues are stored in this counter. If TVBQCNT reaches the threshold value, message DSI374A is issued. Threshold values for the public message queues are described in the <em>Tivoli NetView for z/OS Administration Reference</em>.</td>
</tr>
<tr>
<td>TVBUFLD</td>
<td>4</td>
<td>A user field that can be defined by the subtask. A description of user-defined fields on page 108.</td>
</tr>
</tbody>
</table>
| TVBZIND1     | 1      | Indicator byte:  
|              |        | **TVBZPUP** Primary VSAM data set in use  
|              |        | **TVBZSUP** Secondary VSAM data set in use  |
| TVBZIND2     | 1      | Indicator byte:  
|              |        | **TVB374CT** The total number of buffers on the public message queues are stored in TVBQCNT if this bit is on. |
| TVBZIND3     | 1      | Indicator byte:  
|              |        | **TVB374CT** The total number of buffers on the public message queues are stored in TVBQCNT if this bit is on. |
| TVBZIND4     | 1      | Indicator byte:  
|              |        | **TVBAUTOO** User code is running under an unattended operator task.  
|              |        | **TVBAUTVE** The task depends on VTAM and must be terminated by the main task when VTAM ends.  
|              |        | **TVBAUTVS** The task depends on VTAM and must be reattached by the main task when VTAM starts. When VTAM ends, the task terminates. |
DSIUSE: Installation Exit Parameter List

The installation exit parameter list (DSIUSE) contains addresses for the following:

- Buffer containing the message or MSU
- LU name associated with the message or MSU
- Operator identification
- Service work block (DSISWB)
- Task vector block (DSITVB)
- Parse descriptor block (DSIPDB)

An extension of USE is present for DSIEX12 and the DST exit routines involved with input and output (XITCO, XITCI, XITVN, XITVI, XITVO, and XITXL). For DSIEX12, the password, hardcopy printer name, and profile name are given. For the DST exit routines, the address of the DSRB is given.

An extension of USE is present to define the information provided to the resource status manager installation exit (XITST).

The fields in the DSIUSE are:

Table 42. DSIUSE Field Names and Descriptions

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USEFPEX</td>
<td>35</td>
<td>An extension for the XITST exit routine. If present, this extension contains the following fields:</td>
</tr>
<tr>
<td>USEFPNAM(8)</td>
<td></td>
<td>Resource name.</td>
</tr>
<tr>
<td>USEFPNET(8)</td>
<td></td>
<td>Native network identifier. The network identifier for resources that are defined in a gateway NCP or through a non-native attachment are always under the native network. Neither identifier is under the network identifier that is specified on the LINE or PU definition statements.</td>
</tr>
<tr>
<td>USEFPDOM(8)</td>
<td></td>
<td>Domain identifier.</td>
</tr>
<tr>
<td>USEFPTIM(8)</td>
<td></td>
<td>Time stamp (Greenwich Mean Time in STCK format).</td>
</tr>
<tr>
<td>USEFPREP(1)</td>
<td></td>
<td>Reported resource status. The reported status is the status of the resource as perceived from the VTAM program in the domain identifier field.</td>
</tr>
</tbody>
</table>
Table 42. DSIUSE Field Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USEFPEX</td>
<td>(continued)</td>
<td>Field(Length) Description</td>
</tr>
<tr>
<td><strong>USEFPCAL(1)</strong></td>
<td></td>
<td>Calculated resource status. The calculated status is the status of the resources from all domains. The status of each domain is considered when determining the calculated status. The STATUSTABLE keyword, defined in DUIFPMEM, determines the calculated status for the resource across multiple domains. Refer to the <a href="#">Tivoli NetView for z/OS Administration Reference</a> for a description of the STATUSTABLE keyword.</td>
</tr>
<tr>
<td>Constants for setting/testing USEFPREP and USEFPCAL:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>X'00'</td>
<td>USEDELT</td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td>USERACT</td>
<td></td>
</tr>
<tr>
<td>X'02'</td>
<td>USERNACT</td>
<td></td>
</tr>
<tr>
<td>X'03'</td>
<td>USERNVAC</td>
<td></td>
</tr>
<tr>
<td>X'04'</td>
<td>USERPEND</td>
<td></td>
</tr>
<tr>
<td>X'05'</td>
<td>USEROTHR</td>
<td></td>
</tr>
<tr>
<td>X'06'</td>
<td>USERCONN</td>
<td></td>
</tr>
<tr>
<td>X'07'</td>
<td>USERPINA</td>
<td></td>
</tr>
<tr>
<td>X'08'</td>
<td>USERPACT</td>
<td></td>
</tr>
<tr>
<td>X'09'</td>
<td>USERROUT</td>
<td></td>
</tr>
<tr>
<td>X'0A'</td>
<td>USERUNKN</td>
<td></td>
</tr>
<tr>
<td>X'0B'</td>
<td>USERRELS</td>
<td></td>
</tr>
<tr>
<td>X'0C'</td>
<td>USERRSET</td>
<td></td>
</tr>
<tr>
<td>X'0D'</td>
<td>USERAINS</td>
<td></td>
</tr>
</tbody>
</table>

**USEFPFLG(1)**

Resource status flag byte.

Constants for setting/testing USEFPFLG:

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'80'</td>
<td>USEFPNEW - Represents an initial status for a resource if ON; represents an updated status for a resource if OFF.</td>
</tr>
<tr>
<td>X'40'</td>
<td>USEFPDYN - Represents a dynamic resource if ON.</td>
</tr>
</tbody>
</table>

**USERCBH**

| 4 | A standard control block header. The second byte, USERCODE, indicates the exit routine that is called. Values X'01'–X'15' correspond to installation exits DSIEX01–DSIEX19. The DST installation exit codes are defined in the USE control block. |
### Table 42. DSIUSE Field Names and Descriptions (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERLGON</td>
<td>36</td>
<td>An extension for DSIEX12 and the DST exit routines. If present, this extension contains the following fields:</td>
</tr>
<tr>
<td>Field(Length)</td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>field(4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USEDSRB</td>
<td></td>
<td>For DST exit routines, the fields XITVI, XITVO, XITCI, XITCO, and XITXL point to the DSRB associated with the DST I/O request.</td>
</tr>
<tr>
<td>USETOTVB</td>
<td></td>
<td>For DSIEX12 only. For other exit routines, this field is not initialized.</td>
</tr>
<tr>
<td>USENPSWD(8)</td>
<td></td>
<td>For DSIEX12 only. This field contains the new password successfully entered by the operator at logon when OPERSEC (in CNMSTYLE) is specified as SAFPW, SAFCHECK or SAFDEF. If OPERSEC=MINIMAL or if OPERSEC=NETVPW is specified, USENPSWD contains blanks (X'40'). For exit routines other than DSIEX12, this field is not initialized.</td>
</tr>
<tr>
<td>USERHCPY(8)</td>
<td></td>
<td>For DSIEX12 only. This field contains the name of the hardcopy printer used by the operator for this session. If no hardcopy is used or if OPERSEC=MINIMAL (in CNMSTYLE) is specified, USERHCPY contains blanks (X'40'). For exit routines other than DSIEX12, this field is not initialized.</td>
</tr>
<tr>
<td>USERPROF(8)</td>
<td></td>
<td>For DSIEX12 only. This field contains the name of the profile used for this session. If OPERSEC=MINIMAL (in CNMSTYLE) or if OPERSEC=SAFDEF is specified, USERPROF contains blanks (X'40'). For exit routines other than DSIEX12, this field is not initialized.</td>
</tr>
<tr>
<td>USEPSWDB(8)</td>
<td></td>
<td>For DSIEX12 only. This field contains the operator password entered at logon. If OPERSEC=MINIMAL (in CNMSTYLE) is specified, USEPSWDB contains blanks (X'40'). For exit routines other than DSIEX12, this field is not initialized.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>USERLU</td>
<td>4</td>
<td>A pointer to an 8-byte area that contains the logical unit name (LUNAME) related to the subtask in control, as follows:</td>
</tr>
<tr>
<td>Task</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>MNT</td>
<td>The characters SYSOP</td>
<td></td>
</tr>
<tr>
<td>OST</td>
<td>The LU name of the operator's terminal</td>
<td></td>
</tr>
<tr>
<td>NNT</td>
<td>The APPL name of the OST that issued the START DOMAIN command (the domain name followed by a 3-digit number)</td>
<td></td>
</tr>
<tr>
<td>PPT</td>
<td>The domain name parameter followed by the characters PPT</td>
<td></td>
</tr>
<tr>
<td>HCT</td>
<td>The LU name of the hardcopy printer</td>
<td></td>
</tr>
<tr>
<td>DST</td>
<td>The name of the task specified by the TASK statement in CNMSTYLE</td>
<td></td>
</tr>
<tr>
<td>USERMSG</td>
<td>4</td>
<td>A pointer to a buffer in standard buffer format, consisting of a buffer header (BUFHDR) followed by text. For input-type exits, device dependencies have been removed. In exit routines DSIEX14, XITDI for end-of-file, and XITVN, this field is set to 0. In DSIEX04, the buffer is in the format set up by the caller. It has not yet been reformatted for the network log, MVS system log, or hardcopy log.</td>
</tr>
<tr>
<td>USEROPID</td>
<td>4</td>
<td>A pointer to an 8-byte area that contains a name related to the subtask in control, as follows:</td>
</tr>
<tr>
<td>Task</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>MNT</td>
<td>The characters SYSOP</td>
<td></td>
</tr>
<tr>
<td>OST</td>
<td>The operator identifier</td>
<td></td>
</tr>
<tr>
<td>NNT</td>
<td>The operator identifier</td>
<td></td>
</tr>
<tr>
<td>PPT</td>
<td>The NCCFID DOMAINID parameter followed by the characters PPT</td>
<td></td>
</tr>
<tr>
<td>HCT</td>
<td>The LU name of the hardcopy printer</td>
<td></td>
</tr>
<tr>
<td>USERPDB</td>
<td>4</td>
<td>A field that points to a PDB or contains 0. The PDB contains parse data that relates to the buffer to which USERMSG points. For exit routines DSIEX02A, DSIEX04, DSIEX07, DSIEX09, DSIEX10, DSIEX14, DSIEX16, DSIEX16B, DSIEX17, DSIEX19, XITXL, and XITDI for end-of-file, this field contains 0. A PDB is not available when calling these exit routines.</td>
</tr>
<tr>
<td>USERSWB</td>
<td>4</td>
<td>A pointer to the SWB, which the exit routine uses as a work area or to request services from the NetView program. If necessary, another SWB can be obtained using macro DSILCS.</td>
</tr>
<tr>
<td>USERTVB</td>
<td>4</td>
<td>A pointer to the TVB. The TVB contains information about the subtask under which the exit routine was called. The TVB obtains the addresses of the TIB, the MVT, and the SVL (through the MVT).</td>
</tr>
</tbody>
</table>
The RUNCMD installation exit buffer (DSIXRCMD) provides a mapping of the command buffer that is passed to the RUNCMD installation exit DSIEX19 as USERMSG.

The following fields are in DSIXRCMD:

Table 43. DSIXRCMD Field Descriptions

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCMDUSER</td>
<td>8</td>
<td>User ID which is the source of the RUNCMD.</td>
</tr>
<tr>
<td>*</td>
<td>8</td>
<td>A reserved 8-byte field.</td>
</tr>
<tr>
<td>RCMDUTOK</td>
<td>80</td>
<td>Security product token for the source issuer of RUNCMD, if available.</td>
</tr>
<tr>
<td>RCMDDSPNAM</td>
<td>8</td>
<td>The service point name entered on the RUNCMD.</td>
</tr>
<tr>
<td>RCMDDAPPL</td>
<td>8</td>
<td>The service point application entered on the RUNCMD.</td>
</tr>
<tr>
<td>RCMDDNTID</td>
<td>8</td>
<td>The NETID entered with the RUNCMD installation exit.</td>
</tr>
<tr>
<td>RCMDCLEN</td>
<td>2</td>
<td>Length of the service point command string.</td>
</tr>
<tr>
<td>RCMDCSTR</td>
<td></td>
<td>The service point command string. The length of the string varies, depending on the individual command string.</td>
</tr>
</tbody>
</table>
Chapter 8. Macros

This chapter describes the purpose and coding of the NetView program macros. You can use these macros to request various service facilities when writing your own installation exit routines, command processors, and subtasks. You must be in problem program state and user protection key for all these macros. Appendix D lists the corresponding PL/I command for each of these macros.

NetView macros overwrite registers 0, 1, 14, and 15. Prior to issuing any macro, except DSICBS, set register 13 to a standard 72-byte save area.

All return codes in this chapter are shown in decimal.

Use only the operands documented in this book. Any undocumented operands that a macro has are only for internal use by the NetView program and cannot be used in user-written programs.

DSIAUTO: Invoke Automation Services

Macro DSIAUTO provides a way for a program executing within the NetView address space in MVS to pass a management services unit (MSU) directly to the active automation table.

If there is an active automation table, the automation table statements are scanned for appropriate matches with the MSU passed by DSIAUTO. Any actions specified on matching automation statements are taken.

The PL/I and C equivalent to DSIAUTO is CNMAUTOTAB/CNMAUTO.

DSIAUTO issues a return code that indicates the result of the automation table scan.

The following format for the macro DSIAUTO is as follows:

```
DSIAUTO
```

```
label
```

```
DATA=atdata
```

```
,SWB=swb
```

Where:

DATA

Specifies the register containing the address or the symbolic name of the data item, which must be one of the following forms:

- A multiple-domain support message unit (MDS-MU)
- A control point management services unit (CP-MSU)
- A network management vector transport (NMVT)

This is a required operand.
DSIAUTO

SWB
Specifies either a register or the symbolic name of a fullword area containing the address of a service work block (DSISWB).

This is a required operand.

Return Codes in Register 15
The following are return codes for macro DSIAUTO:

0  One or more matches were found and the MSU was automated.
4  A match was not found and the MSU was not automated.
8  NetView automation inactive. There is no active automation table.
12  Storage failure.
16  Incorrect data item. The MSU was not recognized as one of the previously described types.
20  No entries match this type. The automation table has no MSU-oriented statements.
40  NetView internal failure.
100  Automated but rerouted. One or more matches were found and the MSU was automated. The automation action was rerouted to an OST for processing.
200  Automated and reroute failed. One or more matches were found and the MSU was automated. The automation action was rerouted to an OST, but the OST was not active.

Note: The address for the NMVT must have a 2-byte length field preceding the X'41038D' header. This length is similar to the 2-byte length field defined as the beginning of an MDS-MU or CP-MSU. However, the NMVT length field reflects the length starting at the X'41038D' header to the end of the NMVT. The length in MDS-MUs or CP-MSUs reflects the length of the MSU starting at the length field. The difference is that an NMVT length field does not include the length of the length field itself.

For more information about the formats of architected vectors, refer to the SNA library.

DSIBAM: Build Automation Message

Macro DSIBAM specifies the keyword=data tokens included in the alert automation message generated by CNMS4284.

Macro DSIBAM can be issued ONLY from CNMS4284.

The following format is the syntax for macro DSIBAM:

DSIBAM
DSIBAM

Subvector:

Where:

CONTROL
Specifies whether to add the keyword=data token to the control line of the alert automation message.
N  Do not put the keyword=data token on the control line. This is the default.
Y  Put the keyword=data token on the control line.

DATALIN
Specifies whether to add the keyword=data token to a data line of the alert automation message.
N  Do not put the keyword=data token on a data line.
Y  Put the keyword=data token on a data line. This is the default.

DOMID
Specifies whether to evaluate the domain ID in the major vector. You must specify DOMID only once.
N  Do not build DOMID=data token.
Y  Build DOMID=data token.

END
Causes the table to end. You must specify END only once, and it must be the last occurrence of DSIBAM.

HIER
Specifies whether to evaluate the hierarchy in the major vector. The data portion of the HIER keyword=data token is formatted as a list of up to five 8-character resource names/resource type pairs (NAME1, TYP1, NAME2, TYP2, ...). You must specify HIER only once.
N  Do not build HIER=data token.
DSIBAM

Y  Build HIER=data token.

MSGID
Specifies the unique 8-character message ID for the alert automation message. You must specify MSGID only once.

SFID
Specifies a 1-byte (2 alphanumeric characters) subfield ID. This subfield occurs within the subvector specified by the SVID. A keyword=data token is built for this subfield and added to the alert automation message.

SF_OCCUR
Specifies the occurrence of the SFID to be found. This is a 1- or 2-character numeric subvector occurrence number.

SVID
Specifies a 1-byte (2 alphanumeric characters) subvector ID. A keyword=data token is built for this subvector and added to the alert automation message.

SVOCCUR
Specifies the occurrence of the SVID to be found. This is a 1- or 2-character numeric subvector occurrence number.

XLATE
Specifies whether to translate the EBCDIC representation of hexadecimal data to hexadecimal.
N  Do not do the translation. This is the default.
Y  Do the translation.

DSIBAMKW: Build Automation Message Keyword

Macro DSIBAMKW establishes register conventions for a call to subroutine SR$TK in sample CNMS4284. SR$TK moves data to a keyword=data token for the alert automation message generated by CNMS4284.

Macro DSIBAMKW can be issued only from CNMS4284.

The following format is the syntax for macro DSIBAMKW:

DSIBAMKW

Where:

DATA
Is the number representing the address of the data to be put into the keyword=data token.

LENGTH
Is the number representing the fullword area containing the address of the length of the data to be put into the keyword=data token.
PREFIX
Specifies the characters to prefix DATA on the token.

TYPE
Specifies the data type to be moved:
CHARS
The data is in EBCDIC character format.
HEX
The data is in hexadecimal format.

DSICBS: Control Block Services
Macro DSICBS creates DSECTs for the control blocks required for user-written programs during assembly.

DSICBS ensures that a control block is included only once, inner control blocks are included if necessary, and each definition for an inner control block precedes the definition of the outer control block. DSICBS also controls the format and printing, or suppression, of DSECTs for the control blocks.

DSIMVT addressability is not required.

The following format is the syntax for macro DSICBS:

```
DSICBS
```

Where:

`cbname`
Is the name of a control block, starting with DSI, to be included. Names must be separated by commas.

`DEFER`
Defers control block expansions.

`ALL`
Specifies that all subsequent DSICBSs are not expanded until a DSICBS DEFER=INCLUDE is encountered. (If you specify this operand, be sure to code DSICBS DEFER=INCLUDE later in the program.)

`INCLUDE`
Specifies that any deferred control block expansions are to be expanded at this point in the program.
DSICBS

NO
Specifies that the control blocks are to be expanded immediately. This is the default.

THESE
Specifies that these control block expansions are delayed until a DSICBS DEFER=INCLUDE is encountered.

EJECT
Specifies whether EJECT statements are performed between each control block expansion and after the last expansion.

NO
Specifies that EJECT statements are not performed.

YES
Specifies that EJECT statements are performed. This is the default.

PRINT
Specifies whether control blocks are to be printed (expanded) in the assembler listing.

NO
Specifies that the control block expansion is not to be printed.

YES
Specifies that the control block expansion is to be printed. This is the default.

RSECT
Generates the appropriate section resumption.

TEST
Test for CSECT and RSECT and resume the appropriate section. This is the default.

NO
Generate CSECT resumption.

YES
Generate RSECT resumption.

DSICES: Command Entry Services

Macro DSICES uses a specified buffer, parse descriptor block (DSIPDB), or load module name to locate a command processor address and can be used to determine if a task is authorized to issue a command.

DSICES locates a system command entry (DSISCE) that corresponds to the command verb and returns the DSISCE address to a user-provided fullword area. In addition to performing an authorization check for the command, the macro can check whether the command is a valid command list name and whether the command should be processed as a regular or immediate command.

DSIMVT addressability is required.

The following format is the syntax for macro DSICES:

DSICES
Where:

BFR
Is the register, or symbolic name of a fullword area, containing the address of the buffer that contains the verb to be analyzed. This buffer must have an initialized BUFHDR.

CLISTCK
Specifies whether or not to check for a valid command list name if the command has no CMDMDL statement in DSICMD.

NO
Specifies that you do not need to check for a command list name if a CMDMDL statement is not found. This is the default.

YES
Specifies that you need to check for a command list name if a CMDMDL statement is not found.

MODNAME
Specifies the module name to be located in the system command table. You can specify the module name enclosed in single quotes.

PDB
Is the register, or symbolic name of a fullword area, containing the address of a completed parse descriptor block (DSIPDB) to be used as input.

SCTADDR
Is the register containing the address of a user-provided fullword area, or the symbolic name of that area, where the address of the system command entry corresponding to the module name or verb is to be returned. The returned pointer addresses an SCT entry (DSISCE) dummy control section (DSECT).

SWB
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB). SWBTIB identifies your task information block (DSITIB), which identifies your task type.

Notes:
1. If MODNAME is specified, the first entry that is found in the SCT corresponding to the module name is returned. If the module name specified is
defined in DSICMD on more than one CMDMDL statement, an unexpected address can be returned. In this situation, use the BFR operand instead.

2. Authorization checking will not be performed if MODNAME is used.

3. BFR, PDB, and MODNAME are mutually exclusive, and you must specify one of the three.

4. You cannot specify CLISTCK with MODNAME.

Return Codes in Register 15

The following return codes for macro DSICES are found in register 15:

0  The function is successful. One of the following describes what occurred:
    • A regular command is found in the system command table and the address of the SCT entry is returned.
    • The verb is not found in the SCT (if you specify CLISTCK, a command list is found with the specified name), and the dummy SCT entry for a command list is returned.

4  The command that is found can be processed as a regular or immediate command; the address is returned.

8  An immediate command is found in the system command table; the address is returned.

12 The module is not found, or there is an incorrect verb length; no address is returned.

16 The operator is not authorized to issue the command. This is caused by the security definitions that are in place for this command. No address is returned in SCTADDR.

This return code is not applicable if MODNAME was specified.

20 Either the command found is incompatible with the task type that called the routine, and the address is returned; or you specified CLISTCK=YES and the request is issued in an asynchronous exit, and the address is not returned.

24 You specified CLISTCK=YES but the command or command list is not found in DSISCT or DSICLD.

28 You specified CLISTCK=YES but storage requested for CLISTCK processing is not obtained.

32 NetView internal error.

36 An unexpected return code was received from the security authorization facility (SAF). Message BNH238E is issued with the SAF return code inserted. This return code is not applicable if MODNAME was specified as this specification causes no authorization checking to be performed.

40 Authorization to the command is not granted because the security environment for the operator cannot be established. Message BNH239E is issued when this condition is first encountered to provide the security product return code information. Message BNH273I is issued when the condition has been corrected. This return code is not applicable if MODNAME was specified as this specification causes no authorization checking to be performed.

44 Authorization to the command is not granted because an unexpected return code was received from the command authorization table. Message BNH199E is issued indicating the command identifier and the operator ID
being checked. This return code is not applicable if MODNAME was specified as this specification causes no authorization checking to be performed.

48 Authorization to the command is not granted because the NetView internal security information containing the source ID of the command could not be found. Message BNH277E is issued identifying the command being checked. This return code is not applicable if MODNAME was specified as this specification causes no authorization checking to be performed.

52 Authorization to the command is not granted because the source ID is blank in the NetView internal security information. Message BNH277E is issued identifying the command being checked. This return code is not applicable if MODNAME was specified as this specification causes no authorization checking to be performed.

**DSICVTHE: Convert to Hexadecimal**

Macro DSICVTHE establishes register conventions for a call to subroutine SR$HEX in sample CNMS4284. SR$HEX converts a hexadecimal input string to EBCDIC character format for the alert automation message generated by CNMS4284.

Macro DSICVTHE is issued ONLY from CNMS4284.

The following format is the syntax for macro DSICVTHE:

```
DSICVTHE

label DSICVTHE INPUT= (register) LENGTH= (register)

OUTPUT= (register)
```

Where:

**INPUT**

Is the register, or symbolic name of the fullword area, containing the address of the input string to convert.

**LENGTH**

Is the register, or symbolic name of the fullword area, containing the address of the length of the input string to convert.

**OUTPUT**

Is the register, or symbolic name of the fullword area, containing the address to which the conversion of the input string is to be moved.

**DSIC2T: Code Point Translation Service Reference**

You can use NetView Bridge with a problem-management database to open problem records when NetView alerts are received. When you use NetView Bridge in this manner, you should translate the numeric code points received in the alert into readable text. The NetView code point translation service is provided to perform this translation.
The code point translation service routine is available to the NetView program in REXX, PL/I, C, and assembler languages. The function performed is the same, regardless of the language you choose to use.

The following format is the syntax for macro DSIC2T:

**DSIC2T**

```
DSIC2T SWB= (register),TXTAREA= (register), TABLE= (register), CODE= (register), TXTLENG= (register)
```

*Where:*

**CODE**

Is the register containing the address of the area that contains the code point to be translated, or symbolic name of that area.

**MXTXTLN**

Is the register containing the address of the area that contains the maximum length of the text that can be returned, or symbolic name of that area. This is the maximum length of the *txtarea* field.

**SWB**

Is the register containing the address of a service work block, or symbolic name of that area.

**TABLE**

Is the register containing the address of the area that contains the 8-character name of the table to be used in the translation, or symbolic name of that area. The following tables are valid:

- **SNAALERT** SNA alert description code point
- **SNACAUSE** SNA probable cause
- **SNADDDATA** SNA detailed data
- **SNAFCAUS** SNA failure cause
- **SNAICAUS** SNA install cause
- **SNAREACT** SNA recommended actions
- **SNAUCAUS** SNA user cause

**TXTAREA**

Is the register containing the address of the area that receives the text for the translated code point, or symbolic name of that area.

**TXTLENG**

Is the register containing the address of the area that contains the length of the text returned for the specified code point, or symbolic name of that area.

**Note:** If the alert text is truncated, it is truncated using the MXTXTLN field value. Then the TXTLENG field contains the full alert text length.
Return Codes in Register 15

The following are return codes for macro DSIC2T:

0  The function is successful.
4  Alert text is truncated at specified MXTXTLN length. TXTLENG contains full alert text length.
8  Code point was not found. Length of TXTLENG is zero (0).
12  Table type is incorrect.

DSIDATIM: Date and Time

Macro DSIDATIM obtains and formats the date and time.

DSIDATIM places the date and time in an output area. You can use this macro to obtain the time for the HDRTSTMP field of a message.

MVT addressability is required.

The following format is the syntax for macro DSIDATIM:

```
DSIDATIM
```

Where:

AREA
Is the register containing the address of the area into which the date and time are returned, or symbolic name of that area. This area does not have a buffer header.

FORMAT
Specifies the format of the output.

BINARY
Returns the date and time in 8 bytes in packed decimal format as follows:

```
0CyydddFhhmmss0C
```

Where 0C indicates the century. In the years 1900 through 1999, the value of this field is 00. In the years 2000 through 2099, the value of this field is 01 yy is the last two digits of the year. ddd is the day. F is a 4-bit sign character that enables the data to be unpacked and printed. hh is the hour, mm is the minute, and ss is seconds. The ending X'0C' indicates the data is in packed decimal format.

EBCDIC
Returns the date and time in 17 bytes (including the space between date and time) formatted as follows:

```
mm/dd/yy hh:mm:ss
```

Where mm is the month, dd is the day, yy is the year, hh is the hours, mm is the minutes, and ss is the seconds. EBCDIC is the default.
**DSIDATIM**

*Note:* When using the BINARY form of DSIDATIM to initialize a HDRTSTMP, use an 8-byte work area for the AREA, and move the low-order 4 bytes to HDRTSTMP (X'hhmmss0C').

---

**DSIDEL: Delete User-Defined Module**

Macro DSIDEL deletes user-defined load modules. You specify the name of the module to be deleted. The following format is the syntax for macro DSIDEL:

```
DSIDEL
```

Where:

**EP**  Specifies the name of the module to be deleted.

**EPLOC**  Specifies the address of an 8-byte field that contains the module name to be deleted. The module name is left-justified and padded with blanks.

**Return Codes in Register 15**

The following return codes for macro DSIDEL are found in register 15:

- **0**  The module has been deleted.

- **Nonzero**  The attempt to delete the module was unsuccessful.

Refer to your operating system macro reference for the system macro return code description.

---

**DSIDKS: Disk Services**

You can use macro DSIDKS to connect to a DDNAME, locate a member, and read the records in that member. You can use this macro only to connect to data sets with the following DDNAMEs:

- BNJPNL1
- BNJPNL2
- CNMNPNL1
- DSIARPT
- DSIASRC
- DSICLD
- DSILIST
- DSIMSG
- DSIOOPEN
- DSIPARM
- DSIPRF
- DSIVTAM

NetView opens these data sets and keeps them open as long as NetView is operating. You can then use DSIDKS to find and read any member in any of the data sets concatenated under any of the DDNAMEs in your NetView startup procedure.
You must have a copy of the DSECT for the disk service block (DSIDSB) included in your program. DSIMVT addressability is also required.

The following format is the syntax for macro DSIDKS:

**DSIDKS**

```
label DSIDKS SWB= (register),DSBWORD= (register)
symbolic_name,TYPE= CONN
FIND
DISC
READ
symbolic_name

WHERE:

**DSBWORD**
Is the register containing the address of a user-provided fullword area on a fullword boundary, or symbolic name of that fullword area. When the macro completes processing for TYPE=CONN, this area contains the address of the DSB. For other disk service requests, this area specifies the DSB address previously obtained by TYPE=CONN.

**INCL**
Specifies whether %INCLUDE cards are to be processed. A %INCLUDE card is a record type that enables another member or file to be embedded into the member or file being read at the point that the %INCLUDE card is found.

INCL=YES specifies that any %INCLUDE cards found are to be processed and the members on files specified on the %INCLUDE cards found are to be embedded. INCL=NO specifies that any %INCLUDE cards found are not to be processed. The INCL default is NO.

For DD names listed previously, other than DSICLD, INCL=YES also enables the use of Data REXX in the member. Refer to [Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language](#) for details.

**NAME**
For TYPE=CONN and TYPE=DISC, a register containing the address of an 8-character user area with the caller’s definition name (DDNAME), or the symbolic name of that area. The area is left-justified and padded with blanks.

For TYPE=FIND, NAME is a register containing the address of an 8-character user area that contains the name of the member to be read, or the symbolic name of that area.

For TYPE=READ, NAME is not needed.

**SWB**
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**TYPE**
Specifies the type of processing the service routine is to perform:

**CONN**
Specifies that the service routine is to connect to or access the caller’s definition name (DDNAME). The address of the DSB is returned in the
area specified by the DSBWORD operand. You must issue DSIDKS with this option before you can choose any other options.

**DISC**
Specifies that the service routine is to disconnect from the DDNAME and release the DSB.

**FIND**
Specifies that the service routine is to find the member specified by the NAME operand. If the member is found, the first record is read. DSBBUFF addresses the buffer containing this record. Do not specify this option unless you also specify the CONN option.

**READ**
Specifies that the service routine is to read the next sequential record in the member. Do not specify this option unless you also specify the FIND option.

**Return Codes in Register 15**
The return codes and code meanings in register 15 are dependent on the “TYPE=” specification.

The following return codes are for TYPE=CONN:

- **0** The function is successful. Data control blocks and I/O buffer are obtained and initialized.
- **4** An incorrect data set name.
- **12** No storage was available for I/O buffer.

The following return codes are for TYPE=FIND:

- **0** The function is successful. The member or file is found and the first record is read.
- **4** The member or file is not found in the source statement library or in the specified library, or an empty member or file is found.
- **8** The member or file is found but an I/O error occurred on the first read.
- **12** The specified definition name or data set has not been opened.
- **20** The specified control block identifier is not valid; the member or file is not found.
- **28** There is a syntax error in the %INCLUDE card.
- **36** There is an incorrect member name on the %INCLUDE card.
- **40** There is an incorrect embed member, which can cause a deadlock condition. (This occurs when a member embeds itself.)
- **44** An unrecoverable system error occurred. An internal NetView service failed, because of a storage failure.
- **46** An I/O error is encountered while trying to include a member specified in a %INCLUDE statement.

**100 + xx**
An error occurred during CLOSE processing. The NetView program attempts to recover the data set after a failure during a previous FIND or READ. Refer to the description of the xx return code under the CLOSE macro in your operating system macro reference.
200 + xx
An error occurred during OPEN processing. The NetView program
attempts to recover the data set after a failure during a previous FIND or
READ. Refer to the description of the xx return code under the OPEN
macro in your operating system macro reference.

The following return codes are for TYPE=DISC:
0 The disconnect is successful; data and I/O buffers are freed successfully.
20 The specified control block identifier is not valid and no storage is freed.
46 An I/O error is encountered while trying to INCLUDE a member specified
in a %INCLUDE statement.

The following return codes are for TYPE=READ:
0 The function is successful; the record is read.
4 The end of data is reached.
8 An I/O error occurred during reading.
12 Reading of this record is prohibited; an I/O error may have occurred, the
end of data may have been reached, or the caller did not issue
TYPE=FIND first.
20 The specified control block identifier is not valid; the record is not read.
28 There is a syntax error in the %INCLUDE card.
36 A member name on the %INCLUDE card is not valid.
40 An embed member is not valid and can cause a deadlock condition. (This
occurs when a member embeds itself.)
44 An unrecoverable system error occurred. An internal NetView service
failed, because of a storage failure.
46 An I/O error is encountered while trying to include a member specified in
a %INCLUDE statement.

100 + xx
An error occurred during CLOSE processing. The NetView program
attempts to recover the data set after a failure during a previous FIND or
READ. Refer to the description of the xx return code under the CLOSE
macro in your operating system macro reference.

200 + xx
An error occurred during OPEN processing. The NetView program
attempts to recover the data set after a failure during a previous FIND or
READ. Refer to the description of the xx return code under the OPEN
macro in your operating system macro reference.

Refer to the Tivoli NetView for z/OS Administration Reference for information about
how to code a %INCLUDE card.

DSIFIND: Find Long-Running Command Storage

Macro DSIFIND retrieves a pointer to storage for a long-running command
processor and returns the storage address in register 1. A prior DSIPUSH
instruction names the storage pointer.
DSIFIND

If two storage pointers are given identical names, DSIFIND retrieves the most recent storage address with the specified name. You can issue DSIFIND in an immediate command. See "DSIPUSH: Establish Long-Running Command" on page 229 and "DSIPOP: Remove Long-Running Command" on page 217 for more information.

DSIMVT addressability is required.

The following format is the syntax for macro DSIFIND:

```
DSIFIND

(label) DSIFIND SWB= (register)
(symbolic_name), LIST= (register)
(symbolic_name)

, CORR= NONE

, CORR= CMD

NONE

LIST
```

Where:

**CORR**

Defines the correlation option for the DSIFIND request.

When a RESUME routine is specified for DSIPUSH, the RESUME routine is called with the correlation active at the time of the DSIPUSH regardless of the CORR option on either the DSIPUSH or the DSIFIND.

Options are:

**CMD**

Means that the correlation environment saved by DSIPUSH is retrieved and becomes the current command correlation environment. The correlation environment is saved if the CORR=CMD option was specified on DSIPUSH, and there was no associated RESUME routine specified on DSIPUSH, or if DSIPUSH specified an associated RESUME routine.

This function enables the DSIPSS output of a command using the DSIFIND CORR=CMD to be treated as if it was issued by the command that issued this DSIPUSH. The output will, in most cases be asynchronous when processed by a NetView pipeline, and a CORRWAIT stage should be coded in the pipeline.

**Note**: RESUME routines are recommended as a way to provide correlation, and as a general method of writing functions for NetView. The CORR=CMD option is intended for cases where a RESUME routine might be impractical.

**NONE**

Specifies that DSIFIND does not make the saved correlation environment the current command correlation environment. NONE is the default when the CORR keyword is omitted.

**LIST**

Is the register containing the address of the parameter list used by the service.
The routine, or symbolic name of that list. Do not specify this as register 1; register 1 contains the SWB address within DSIFIND. Do not put this list in the SWB that is to be passed to DSIFIND.

The parameter list is mapped by SWBLRCPL and contains the following fields:

<table>
<thead>
<tr>
<th>Hex Offset</th>
<th>Length</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>SWBLRCLN (length)</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>SWBLRCNM (name)</td>
</tr>
</tbody>
</table>

Where:

**SWBLRCLN**

Specifies the parameter list length. Set this halfword equal to SWBLRCFI (decimal 20).

**SWBLRCNM**

Specifies the name of the storage to be located. The storage address is returned in register 1. Specify this field exactly as you specified it in the corresponding macro DSIPUSH. See "DSIPUSH: Establish Long-Running Command" on page 228 for more information about specifying the name field.

**SWB**

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB). The task information block (DSITIB) address in SWBTIB must be correctly set.

**Return Codes in Register 15**

The following return codes for macro DSIFIND are found in register 15:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function is successful; the storage pointer is retrieved and the storage address is returned in register 1.</td>
</tr>
<tr>
<td>32</td>
<td>Macro invocation is not valid. Correct assembly errors before trying to run the program.</td>
</tr>
<tr>
<td>36</td>
<td>The specified NAME is not found.</td>
</tr>
</tbody>
</table>

**DSIFRE: Free Storage**

You must use macro DSIFRE to release storage that was obtained using macro DSIGET. Storage that was not obtained with DSIGET cannot be released using DSIFRE. Optionally, DSIFRE dequeues the storage from the user’s task vector block (DSITVB).

Registers 2–12 can be used for register notation. DSIFRE always generates reentrant code.

DSIMVT addressability is required.

The following format is the syntax for macro DSIFRE:

```dsc
DSIFRE
```
Where:

A  Is the register, or symbolic name of a fullword area on a fullword boundary, containing the address of the storage to be freed.

AQ  Indicates whether all queued storage is to be released. If you specify AQ, you cannot specify LV or A.

   NO  Specifies that queued storage is not to be released. This is the default.

   YES  Specifies that queued storage is to be released.

LV  Is the number of bytes, or a register that contains the number of bytes, of storage to be freed. This option is ignored if you specify Q=YES.

MAINTSK  Indicates whether the storage to be freed requires special handling to avoid storage accounting problems. MAINTSK=YES indicates that the storage should be treated as if it was owned by the NetView main task.

MAINTSK=YES and Q=NO should be specified when:

• The storage is obtained by one task.
• This storage is to be freed by a different task.
• The storage is not a buffer transferred by DSIMQS.

MAINTSK is ignored when Q=YES.

   NO  Specifies that the storage is not to be treated as if it was owned by the NetView main task. This is the default.

   YES  Specifies that the storage is to be treated as if it was owned by the NetView main task.

Q  Indicates whether the storage is to be released from the user’s TVB. This option must correspond to the option that was used to obtain storage using DSIGET.

   NO  Specifies that storage is not to be released. This is the default.
YES

Specifies that storage is to be released.

**Note:** Do not modify the two words immediately preceding the queued storage. An abend can result.

**SP**

Is the subpool number, or a register loaded with the subpool number, from which the storage is to be freed. Values in the range of 0–255 are acceptable; 0 is the default value.

**TASKA**

Is the register containing the address of the TVB, or symbolic name of the TVB for this task. If you do not specify this operand, the default is DSITVB, and addressability to the DSITVB is required.

**Return Codes in Register 15**

The following return codes for macro DSIFRE are found in register 15:

0  The function is successful; storage is freed and dequeued.

4  A possible storage overlay was detected. This return code will be accompanied by message DWO115W. Depending on the value you coded for STORDUMP, a dump may be attempted.

8  A storage mapping failure was detected. This means that the length or subpool on a piece of storage being freed does not match the length or subpool of the storage when it was gotten, or the storage may have already been freed. This error condition may result in an S378 abend.

12  The error conditions indicated by return codes 4 and 8 were both detected.

16  DSIFRE detected that a NetView storage management control block may have been overlaid, or it contains pointers that are not valid which may be the result of an overlay. Message DWO115W is issued, and depending on the value of STORDUMP, a dump may be attempted.

20  For storage subpools 0–15, of a size less than or equal to X'FE8' (decimal 4072), either DSIFRE Q=YES was issued for storage that was not obtained with DSIGET Q=YES, or the storage specified on the DSIFRE (Q=YES or Q=NO) has already been freed. Message DWO115W is issued with this return code, and depending on the value of STORDUMP, a dump may be attempted.

24  A zero length and zero address was specified on DSIFRE without the AQ option.

40, 42, 44, 46  A NetView internal error was detected. A storage overlay dump may be attempted, depending on the value of STORDUMP. Contact Tivoli Customer Support for programming assistance.

**Notes:**

1. For all return codes, if a storage overlay dump is taken, register 7 at the time of the dump will contain the return code. More information about the contents of the registers at the time of the dump can be found in the *Tivoli NetView for z/OS Diagnosis Guide*.

2. NetView-detectable DSIFRE Q=YES errors will result in NetView user abend ABENDU084. Most of these errors are a possible indication of a storage overlay. The user abend and associated dump are taken to facilitate problem determination.
DSIFRE

3. If an ABENDSA78 occurs at task termination and this DSIFRE error (return code 20 in register 15) occurs, a trace with OPT=STOR shows the DSIFRE trace entries with a return code of 20.

As an additional aid if this abend occurs, the return address of any issuer of DSIFRE Q=NO that results in a return code of 20 in the trace is stored in the DSITVB control block for the task under which the DSIFRE was issued. The saved address will be in field TVBFREM (TVB + X'1B4').

4. If you are a programmer developing new code to run under NetView, this return code indicates that you may have coded a DSIGET/DSIFRE pair incorrectly, or may have coded one without the other.

For example, if you get storage using a system macro such as GETMAIN or GETVIS, but release the storage with the DSIFRE macro, the issuer of DSIFRE will get a return code 20. In this case, the return code 20 is not really an error, but indicates that you did not use DSIGET to get the storage. DSIFRE will still use FREMAIN to free the storage.

DSIFREBS: Call DSIFREBS Service

The DSIFREBS macro is provided to simplify the call to the DSIFREBF assembler language called service routine. For an example of how to use it, see “Free NetView Buffers Service Routine (DSIFREBF)” on page 277.

DSIGET: Get Storage

You can use macro DSIGET to get storage. Optionally, you can use DSIGET to queue the obtained storage to your task vector block (DSITVB). Storage obtained with DSIGET is released using DSIFRE.

Registers 2–12 can be used for register notation. DSIGET enables you to queue storage on the TVB chain so that NetView can free the storage at logoff.

DSIMVT addressability is required.

The following format is the syntax for macro DSIGET:

DSIGET

\[ \text{DSIGET} \quad \text{LV} = \quad \text{(n)} \quad \text{A} = \quad \text{(register)} \quad \text{symbolic\_name} \]

\[ , \text{SP} = \quad \text{(register)} \quad \text{number} \]

\[ , \text{LOC} = \text{RES} \quad \text{LOC} = \text{ANY} \]

\[ , \text{BNDRY} = \text{DBLWD} \quad \text{BNDRY} = \text{DBLWD} \]

\[ , \text{CLEAR} = \text{YES} \quad \text{CLEAR} = \text{NO} \]

\[ , \text{Q} = \text{NO} \quad \text{Q} = \text{YES} \quad \text{TASKA} = \quad \text{(register)} \quad \text{symbolic\_name} \]
Where:

A Is the register containing the address of the fullword area on a fullword boundary into which the address of the obtained storage is returned, or symbolic name of that fullword area.

BNDRY
Specifies the alignment of obtained storage.

DBLWD
Specifies that obtained storage is to be aligned on a doubleword boundary.
This is the default.

PAGE
Specifies that obtained storage is to be aligned on a page boundary.

CLEAR
Specifies whether or not the allocated storage is to be initialized to 0.

NO
Specifies that storage is not to be initialized.

YES
Specifies that storage is to be initialized. This is the default.

LOC
This operand specifies where to allocate storage.

ANY
Allocates storage anywhere.

BELOW
Allocates storage below 16 MB.

RES
Allocates storage that is consistent with the residency of the caller. This is the default.

TEST
The caller of DSIGET has set the high-order bit of register 15 to indicate the type of storage desired. A 0 in the high-order bit means allocate storage below 16 MB; a 1 means allocate storage anywhere.

LV
Is the number of bytes, or a register containing the number of bytes, of storage to be obtained. The value must be positive.

MAINTSK
Indicates whether the storage to be obtained requires special handling to avoid storage accounting problems. MAINTSK=YES indicates that the storage should be treated as if it was owned by the NetView main task.

MAINTSK=YES and Q=NO should be specified when:
• The storage is obtained by one task.
• This storage is to be freed by a different task.
• The storage is not a buffer transferred by DSIMQS.
DSIGET

Multiple data buffers, such as those described by the DSIFR macro, are accounted for by NetView during DSIMQS. These buffers must be obtained with MAINTSK=NO.

MAINTSK is ignored when Q=YES.

NO
   Specifies that the storage is not to be treated as if it was owned by the NetView main task. This is the default.

YES
   Specifies that the storage is to be treated as if it was owned by the NetView main task.

Q
   Indicates whether or not the obtained storage is to be queued to your TVB. The option specified for this operand must correspond to the option specified for the Q operand in DSIFRE that is used to free the storage.

NO
   Specifies that storage is not to be queued. This is the default.

YES
   Specifies that storage is to be queued.

SP
   Is the subpool number, or register containing the subpool number, from which the storage is to be obtained. Values in the range of 0–255 are acceptable; 0 is the default value.

Note: Using a subpool in the range of 0–15 will enable NetView to use its storage management function for pooling storage. This will improve performance on individual requests of storage, but will cause a slight increase in the amount of storage used by NetView. Using a subpool in the range of 16–255 will cause NetView to issue a GETMAIN for each storage request. This will lessen the amount of storage used, but will slow performance.

TASKA
   Is the register containing the address of the TVB for this task, or symbolic name of that TVB. If you do not specify this operand, the default is DSITVB and addressability to the DSITVB is required.

Return Codes in Register 15

The following return codes for macro DSIGET are found in register 15:

0   The function is successful; storage was obtained.
4   No storage is obtained.
8   A negative or zero length was specified on the DSIGET call.

DSIGETDS: Data Queue Manipulation Service

You can use the DSIGETDS macro to retrieve single-line and multiline messages and MSUs in the initial data queue (pointed to by TVBAIIFR).

The following format is the syntax for macro DSIGETDS:

```
DSIGETDS
```

DSIGETDS: Data Queue Manipulation Service
Where:

**BTYPE**
Specifies a 1-byte area in which the buffer type (as in HDRMTYPE) is returned.

For an MSU, it is M. If the command processor is driven by automation of an alert, the second element is the hierarchy list and the buffer type is H.

MSU has a HDRMTYPE of X'10'. When the line type (HDRLNTYP) is B'00', M is returned. When the line type is B'01' (the hierarchy list), H is returned.

**DATALEN**
Specifies a 4-byte integer field where the length of the data (message text or MSU) is returned.

**DATAPTR**
Specifies the name of the 4-byte pointer field in which the address of the message text or MDS-MU data is returned. You can specify the name of the data pointer or a register containing a pointer to the data pointer. Do not free the storage pointed to by DATAPTR. If you need a copy, copy the data into your own storage area.

**DOMID**
Specifies an 8-byte area in which the origin domain ID (as in HDRDOMID) is returned. For an MDS-MU, this is the current NetView domain ID.

**INDEX**
Specifies a 4-byte integer field containing the number (index) of the line of the message to be returned, or of the MDS-MU within a chain of reply MDS-MUs to be returned. If you do not specify index, or if you specify it with a value of zero, the next available line or MDS-MU is returned.

**LTYPE**
Specifies a 1-byte area in which the line type is returned. For an MDS-MU or a single-line message, it is blank. For an MLWTO, it can be set to the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Control line</td>
</tr>
<tr>
<td>L</td>
<td>Label line</td>
</tr>
<tr>
<td>D</td>
<td>Data line</td>
</tr>
<tr>
<td>E</td>
<td>End line without data</td>
</tr>
<tr>
<td>F</td>
<td>End line with data</td>
</tr>
</tbody>
</table>
DSIHREGS

ORIGAPP  
An 8-byte character field in which the origin application name from the MDS header is returned if the data is an MDS-MU. If the data is not an MDS-MU, blanks are returned.

ORIGLU  
An 8-byte character field in which the origin LU or VTAM CP name from the MDS header is returned if the data is an MDS-MU. If the data is not an MDS-MU, blanks are returned.

ORIGNET  
An 8-byte character field in which the origin network ID from the MDS header is returned if the data is an MDS-MU. If the data is not an MDS-MU, blanks are returned.

SWB  
Specifies a 4-byte pointer field containing the address of the service work block (DSISWB) control block. You can specify either the name of the SWB or a register containing the pointer to the SWB.

SWBTIB must have the address of the task information block (DSITIB).

TASKID  
Specifies an 8-byte area in which the origin task ID (as in HDRSENDR) is returned. For an MDS-MU, this is the task ID of the MS transport DST.

TOTLINE  
A 4-byte integer field in which the total number of lines (for example, total number of lines in a MLWTO) is returned. For an MDS-MU, this is 1 or the number of replies in the chain of MDS-MUs. For an automated alert, this is 2.

Return Codes in Register 15

The return codes for macro DSIHREGS, found in register 15, are:

0 The requested function is performed (CNM_GOOD).
12 TIB specified is not valid.
80 Queue empty (CNM_QUEUE_EMPTY).

Note: Most of the return codes are declared in DSIRTN.

DSIHREGS: High-Performance Transport Application Registration

Macro DSIHREGS registers any application that wants to send data to or receive data from another application through the high-performance transport application program interface (API). Register the application before attempting to send or receive data.

DSIHREGS also deregisters applications, which results in the termination of the high-performance transport’s awareness of the application. After deregistration, the application cannot send or receive any further data.

In addition to registering the application, the registration includes the command name of the command processor to be invoked when asynchronous unsolicited data is routed to the application. This command processor runs under the task that invoked the registration macro. Also, you can specify the logmode that this application uses.
The service program you invoke keeps track of the registered applications to provide routing. NetView verifies the command you specify as the command to be driven when unsolicited data is received.

You can specify whether or not the current registration is replaced. When you replace prior registrations, the later registration can replace the task or the command processor. The logmode for the second registration request must be the same as the first and cannot be changed by a later registration request. To change the logmode, deregister the application and reregister it with the new logmode.

The following format is the syntax for macro DSIHREGS:

**DSIHREGS**

\[
\text{DSIHREGS} \quad \text{SWB=} \quad \text{swbname} \quad \text{TYPE=} \quad \text{REGAPPL} \\
\text{DREGAPPL} \\
\text{APPL=} \quad \text{applname} \quad (\text{register}) \\
\text{COMMAND=} \quad \text{cmdname} \quad (\text{register}) \\
\text{LOGMODE=} \quad \text{logmode} \quad (\text{register}) \\
\text{REPLACE=} \quad \text{YES} \quad \text{NO} \\
\text{NOTIFY=} \quad \text{NONE} \quad \text{ALL} \quad \text{ERROR} \quad \text{NONE} \\
\text{PRI=} \quad \text{LOW} \quad \text{HIGH} \quad \text{LOW} \quad \text{NORMAL} \quad \text{TEST}
\]

Where:

**APPL**

Is an 8-byte character field that specifies the application name that is being registered or deregistered.

The identifier name can be one of the following:

- An architecturally defined 4-byte value (padded with blanks to 8 bytes) for management services (MS) application programs.
- A 1–8 character installation-defined name (padded with blanks). Use the EBCDIC characters 0–9 and A–Z (capitals only).

The application names and their hexadecimal equivalents, for the NetView program that reserves application name categories are as follows:

- **ALERT** X’23F0F3F1’
- **EP_OPS** X’23F0F1F6’
- **EP_SPCS** X’23F0F1F4’
- **LINKSERV** X’23F0F3F5’
- **MS_CAPS** X’23F0F1F1’
DSIHREGS

OPS_MGMT X'23F0F1F7'
R_BRIDGE X'30F0F5F9'
RMTCMD_O X'30F0F7F2'
RMTCMD_R X'30F0F5F5'
RMTCMD_S X'30F0F7F0'
SPCS X'23F0F1F5'
STATUS No hexadecimal equivalent
No character equivalent X'23F0F1F0'
No character equivalent X'23F0F0F1'
No character equivalent X'30F0F7F3'

Note: If you use an architected name, you must supply a field name because hexadecimal data cannot be specified in a literal string. The registration service routine ignores any trailing blanks.

COMMAND
Is an 8-byte character field that specifies the command name of the command processor that should be driven with any data, received as a message unit, that has a destination application name equal to the one in the applname operand. COMMAND is either the command name or a register containing a pointer to it. This field is required for registration requests but is not valid for deregistration requests.

LOGMODE
Is an 8-byte character field that specifies the logmode that is used for sending the application data. This name is a logmode that is defined to the local VTAM and the receiving LU or control point (CP) with which this application communicates.

If you specify a LOGMODE that is not defined in the local VTAM logmode table, VTAM defaults this value to the first LOGMODE defined in the local logmode table.

Note: Use LOGMODE only on the first registration request for an application. If another command processor registers for an application that is already registered, LOGMODE is ignored.

NOTIFY
Is a literal value that specifies whether or not this registration is to supply an MDS error message if connectivity to other nodes is lost. NONE is the default.

ALL Specifies that this registration receives all the notifications that ERROR provides, and in cases when the high-performance transport classifies a session outage as normal and does not attempt to reestablish connectivity.

NONE Specifies that this registration does not receive any error notification. NONE is the default.

ERROR Specifies that this registration receives notification if there is a normal or abnormal loss of connectivity to another node. A normal loss can occur if you have outstanding transactions with the other node. An abnormal loss can occur if you have a session outage and connectivity cannot be reestablished.

Note: If two nodes in two networks have the same LU name, VTAM can locate either one, depending on the active configuration.
PRI
Supplies the MQS priority for incoming requests. The MQS priority is used when the MS or high performance transport uses the MQS for processing of any received unsolicited MDS-MUs. The value is one of the four strings HIGH, NORMAL, LOW, or TEST. The priority value must be entered as 8-bytes, including blanks. For example:

`'LOW '`
`'HIGH '`
`'NORMAL '`
`'TEST '`

Note: The single quotes are shown only to demonstrate the 8-byte field. Do not include them as part of the priority specification.

Priorities are:

**HIGH**
Processing begins after any NORMAL requests currently in progress, but before queued NORMAL or LOW requests.

**LOW**
Processing is preempted by HIGH and NORMAL priority requests. This is the default.

**NORMAL**
Processing priority preempts a queue of LOW priority requests.

**TEST**
DSIMQS queues the request at either HIGH or LOW priority, according to the destination task’s command priority. Refer to the OVERRIDE command in the NetView online help for an explanation of command priorities.

**REPLACE**
Is a 4-byte character field that specifies whether this registration is to supersede any previous registration for this application.

**YES**
Specifies that this registration replaces the current registration for this application. YES is the default. If you specify YES, an application can change the name of the command driven to process received data and the task where the command is driven with unsolicited data.

**NO**
Specifies that this registration does not replace the current registration for this application.

**SWB**
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**TYPE**
Is a 10-byte character field that specifies the type of request:

**REGAPPL**
Registers an application to the high-performance transport.

**DEREGAPPL**
Deregisters an application from the high-performance transport.

The NetView task where an application receives an MDS-MU is determined as follows:

- For an MDS reply, the receiving task is the task under which the requesting application was running.
DSIHREGS

- For an MDS request, the receiving task is the task from which DSIHREGS is invoked for the receiving application.
- For an MDS error message:
  - If the agent unit of work correlator (AUOWC) matches an active AUOWC in the active transaction list:
    - For an outgoing request, the receiving task is the task under which the requesting application is running.
    - For an incoming request, the receiving task is the task under which the receiving application is running.
  - If the AUOWC does not match an active AUOWC, the receiving task is the task from which DSIHREGS is invoked for the receiving application.

You can change the task under which DSIHREGS was invoked by registering the application from the desired task and specifying REPLACE=YES.

**Return Codes in Register 15**

The following return codes for macro DSIHREGS are found in register 15:

- **0** The function is successful. The application (APPL) was registered or deregistered. LOGMODE cannot be changed on a subsequent registration request unless the application is first deregistered.
- **4** The attempt made to change the LOGMODE for an application that is already registered is not valid.
- **16** LOGMODE is not specified, or the restricted logmode SNASVCMG is used.
- **20** Deregistration unsuccessful; APPL is not registered.
- **24** Registration unsuccessful; no storage available.
- **44** Deregistration unsuccessful; issued from an installation exit.
- **104** Registration unsuccessful; APPL is already registered.
- **472** Registration/deregistration unsuccessful. The resource user queue is full.
- **476** Registration/deregistration unsuccessful; APPL name syntax is not correct.
- **480** Registration/deregistration unsuccessful; APPL is restricted.
- **580** Registration unsuccessful. LOGMODE does not match the existing LOGMODE for APPL.
- **584** Registration unsuccessful. A value for LOGMODE is required.
- **500 + xx** Registration unsuccessful; failure in the internal NetView resource manager.
- **9012** Registration unsuccessful; COMMAND not valid.
- **9016** Registration unsuccessful. The current task is not authorized for COMMAND.
- **9020** Registration unsuccessful. The current task and COMMAND are not compatible.
DSIHSNDS: Send High-Performance Message Unit

Macro DSIHSNDS enables NetView applications to send data to a specified target through the high-performance transport. The high-performance transport uses an LU 6.2 conversation, and VTAM selects the appropriate session for the actual transmission. You can invoke DSIHSNDS only in applications registered through DSIHREGS.

The data is sent in the form of an MDS-MU. You can supply the following information:

- A completely built MDS-MU
- An MDS-MU that is missing one or more of the following:
  - A unit of work correlator (UOWC)
  - An origin NETID
  - An origin LUNAME

These are added by the service routine.
- A GDS variable that can be contained in an MDS-MU, and can supply sufficient other fields for the service routine to build an MDS-MU header.

Refer to the SNA library for more information about MDS-MUs and GDS variables.

The DSIHSNDS macro builds the necessary NetView MQS buffer with the specified data and queues it to the high-performance transport.

The following format is the syntax for macro DSIHSNDS:

```
DSIHSNDS

(label)

DSIHSNDS SWB=(swbname)

(register),DATATYP=MDSMU

(DATA=(dataarea)

(register)),SUPCORR=(dataarea)

(register),CORAREA=(correlarea)

(register),SECONDS=(timeout) constant

(REPCMD=(replcmd)

(register)),ORIGAPP=(origappl)

(register),DESTNET=(destnet)

(register),DESTLU=(destlu)

(register)
```

Chapter 8. Macros 189
**DSIHSNDS**

Where:

**CORAREA**

Is a 52-byte varying length character field in which a new unit of work correlator (X'1549') GDS variable is created and returned by the DSIHSNDS macro. The length subfield (the first 2 bytes) indicates the length of the correlator. The correlator length is always 52 bytes for this release of NetView.

If you specify CORAREA for an MDS-MU, NetView creates the unit of work correlator in this area and inserts it into the specified MDS-MU while copying it into the buffer for the high-performance transport. In this case, NETID and LUNAME are inserted into the origin location (X'81') subvector at the same time if they are not already present (if there are no X'01' and X'02' subfields within the subvector). If you omit CORAREA, the MDS-MU must be complete and ready to be transmitted as supplied.

For NONMDSMU specify either CORAREA or SUPCORR. If you specify CORAREA, DSIHSNDS creates the unit of work correlator GDS variable in this area and uses it in building the MDS header. The service routine uses the supplied value in building the MDS header if you specify SUPCORR. No validity checking is done for a correlator the invoker supplies.

If this is an MDS reply or an MDS error message, do not use CORAREA, because an MDS reply or error message returns the correlator sent with the request. The invoking application supplies the original correlator either in the MDS-MU or with the SUPCORR keyword.

CORAREA is mutually exclusive with the SUPCORR keyword.

You can specify either the name of the area or a register containing a pointer to the area.

**DATA**

Is a varying length character field containing the data being sent. For either MDSMU or NONMDSMU the first 2 bytes contains the entire length of the data and the next 2 bytes contain the key. The maximum length of the data is as defined in the SNA architecture for the data object being sent.

For an MDS-MU, all fields within the MDS-MU header must be properly prepared before invocation (with the possible exception of the correlator and the origin NETID and LUNAME). If the correlator is not contained in the data, you need to specify corarelarea.

You can specify either the name of the data or a register containing a pointer to the data.

**DATATYP**

Is an 8-byte character field indicating whether the data item specified with the DATA keyword is an MDS-MU or a non-MDS-MU.
MDSMU
Indicates that the DATA keyword is an MDS-MU. MDSMU is the default.

NONMDSMU
Indicates that the DATA keyword is not a complete MDS-MU because it does not contain an MDS-MU header. The DSIHSNDS macro envelopes this data in an MDS-MU header before sending it.

DESTAPP
Is an 8-byte character field that specifies the destination high-performance application name.

The application name can be one of the following:
- An architecturally defined 4-byte value (padded with blanks to 8 bytes) for MS application programs.
- A 1–8 character installation-defined name (padded with blanks). Use the EBCDIC characters 0–9 and A-Z (capitals only).

The DSIHSNDS macro truncates any trailing blanks before putting this value in the MDS-MU header.

You can specify either the destination high-performance application name or a register containing a pointer to the high-performance application.

This field is required for NONMDSMU.

DESTLU
Is an 8-byte character field that specifies the LU name of the destination LU. Specify the 1– to 8–character LU name (padded with blanks to 8 characters) using only the EBCDIC characters 0–9 and A-Z (capitals only).

The DSIHSNDS macro truncates any trailing blanks before putting this value in the MDS-MU header.

This field is required for NONMDSMU.

DESTNET
Is an 8-byte character field that specifies the ID of the network of the destination LU. Specify the 1– to 8–character NETID (padded with blanks to 8 characters) using only the EBCDIC characters 0–9 and A-Z (capitals only).

The DSIHSNDS macro truncates any trailing blanks before putting this value in the MDS-MU header.

You can specify either the destination NETID name or a register containing a pointer to the destination NETID.

If you do not specify the DESTNET keyword when using DSIHSNDS for non-MDS-MU data types, the value of this field defaults to the NETID of the local NetView program. Otherwise, this field is required for non-MDS-MU data types.

MUTYPE
Is a 4-byte integer field that specifies the index number that identifies the type of MDS-MU to build. The type identifies the MDS-MU as a request, a reply, or an error message, and if additional messages are expected.

The following types are defined as constants:
1 REQUEST_WITH_REPLY
2 REQUEST_WITHOUT_REPLY
The DSIHSNDS macro uses the MUTYPE value to determine the settings for
the MDS message type and first and last message indicator bits in the flags
(X'90') MDS routing information subvector.

This is a required keyword for NONMDSMU.

ORIGAPP
Is an 8-byte character field that specifies the origin high-performance
application name.

The application name can be one of the following:
• An architecturally defined 4-byte value (padded with blanks to 8 bytes) for
  MS application programs.
• A 1- to 8-character installation-defined name (padded with blanks). You
  must use the EBCDIC characters 0–9 and A–Z (capitals only).

The DSIHSNDS macro truncates any trailing blanks before putting this value
in the MDS-MU header.

You can specify either the origin application name or a register containing a
pointer to the origin application.

This field is required for NONMDSMU.

PRI
Supplies the MQS priority for incoming reply or MDS error message resulting
from any outgoing MDS-MU. It must be the name or address of an 8-byte
(including blanks) character field that contains the desired priority. Its value,
including blanks, can be:

'LOW'
'HIGH'
'NORMAL'
'TEST'

REPCMD
Is an 8-byte character field containing the name of the command to be driven
with the reply. You can specify REPCMD only in an application that is sending
REQUEST_WITH_REPLY, with the reply being received asynchronously.

If REPCMD is not specified when a high-performance application sends a
REQUEST_WITH_REPLY, the value defaults to the registered command of the
high-performance application at sending time. If the high-performance
application issues the DSIHREGS macro with the REPLACE(YES) option before
the reply is received, the reply is sent to the original registered command when
it comes in.

You can specify either the reply command processor name or a register
containing a pointer to the reply command processor.

This is an optional field. The default is the registered command for the
invoking application.

SECONDS
Is a 4-byte integer field that specifies the number of seconds to wait for the
reply of an outstanding REQUEST_WITH_REPLY. For a REQUEST_WITH_REPLY that generates multiple replies, the timeout value applies only to the last reply.

NetView initializes default and maximum timeout values for the LU 6.2 transport send services. The initial default and maximum timeout values are 120 and 86400 seconds, respectively. If you specify a value of X'FFFFFFF' (-1), the maximum timeout value is used. The maximum value is initialized to 86000 (24 hours). You can change these values with the DEFAULTS command.

The following values are valid for SECONDS:

- **1 ... X** Where X is the maximum timeout value
- **0** Indicates the default timeout value
- **-1** Indicates the maximum timeout value

If you do not specify the SECONDS keyword when using the DSIHSNDS macro for a REQUEST_WITH_REPLY, the default timeout value is used. Otherwise, this field is required for REQUEST_WITH_REPLY.

The parameter specified by SECONDS is either the variable name (`timeout`) containing the time interval value, or the value itself.

**SUPCORR**

Is a varying length character field containing a complete unit of work correlator (X'1549') GDS variable. The SUPCORR field must contain a 2-byte length, a 2-byte key, and at least 1 byte of correlator data. Refer to the SNA library for more information about defining the correlator.

SUPCORR is not valid for an MDS-MU. If you use an existing unit of work or create a new one, the MDS header contains the unit of work and the MDS-MU must be ready to be transmitted as supplied. An MNOTE is returned if you specify this keyword for an MDS-MU.

SUPCORR is optional for NONMDSMU. For NONMDSMU, specify either SUPCORR or CORAREA. The supplied value is used to build the MDS header if you specify SUPCORR. No validity checking is done for a correlator supplied by the invoker. If an MDS reply or an MDS error message is sent, you must specify SUPCORR because an MDS reply or error message returns the correlator sent with the request. The invoking application supplies the original correlator either in the MDS-MU or with the SUPCORR keyword.

You can specify the correlator name or a register containing a pointer to the data.

**SWB**

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**SYNCH**

Specifies the buffering options available on requests_with_replies:

- **NO_BUF**
  Specifies to buffer replies until reply_last is received. This is the default.

- **NO_UNBUF**
  Specifies to send each reply to the application as it is received.

**Notes:**

1. Control is returned to the invoking program after DSIHSNDS successfully queues the request to the high-performance transport.
DSIHSNDS

2. For MDSMU, all fields within the MDS-MU header must be correct except for origin NETID and LUNAME. The macro can determine and set these fields. If the data does not contain the correlator, you must specify CORAREA.

3. For REPLY_ONLY, REPLY_NOTLAST, REPLY_LAST, and ERROR_MESSAGE, you must specify SUPCORR to return the correlator sent with the request.

4. The high-performance transport implements a timeout value for the application receiving the data. If the invocation of DSIHSNDS specifies a timeout value greater than the timeout value set by the transport at the receiving node, the sending application might time out in less than the specified interval.

5. A request can time out in less than the specified interval if the receiving partner has implemented a maximum timeout value of less than 24 hours, or the one specified with the DEFAULTS command.

6. When VTAM is active, you can use DSIHSNDS to send data to another application in the same domain.

7. If DESTNET is not the NETID determined by VTAM for the LU specified in DESTLU, the send fails.

8. A high-performance application cannot send data to itself within the same NetView program.

Return Codes in Register 15

The following return codes for macro DSIHSNDS are found in register 15:

0  Requested function was performed.
4  Task is terminating (TVBABEND/TVBLOGOF is on), and a request-with-reply is sent.
24  No storage.
44  Send MU service is invoked in an asynchronous exit.
56  Timeout value is not valid.
88  MDS-MU length is not valid.
220  DSIHPDST task inactive.
400  Data type is not valid.
404  DATA missing or is not valid.
408  High-performance application cannot send data to the same high-performance application within the same NetView program.
416  MS application not registered.
424  UOW missing or is not valid.
444  Destination network ID is missing or is not valid.
448  Destination LU name missing or is not valid.
452  Destination application name (DAN) missing or is not valid.
460  Reply is not valid.
464  Incorrect MUTYPE given.
472  User list is full.
556  Task does not have authorization to execute the registered command associated with the origin application name (OAN). Ensure the task has...
command authorization for the command specified on the COMMAND operand of the registration macro DSIHREGS.

MDS error message does not have SNA condition report.

NETID is not available. The probable cause is that VTAM is not active.

The variable $x$ is the return code from DSIMQS.

The variable $x$ is the return code from DSIPUSH.

A return code of 24 or 28 from DSIPUSH indicates that DSIOLGF is not defined or is not defined properly in DSICMD.

Reply command is not valid.

The variable $Y$ is the return code from DSICES. Ensure that DSI6SNDP is defined correctly in DSICMD. If you specify a reply command processor, ensure that it is also defined properly in DSICMD. If the sending application is an operations management served application, make sure DSIOARCP is defined properly in DSICMD.

**DSIID: Store SYSMOD Level in CSECT**

You can use macro DSIID to store the SYSMOD level in an assembler module. The DSIID macro expects the &SYSPARM field to be 8 characters or less and contain the SYSMOD level. The DISPMOD command can be used to display the module compile date and SYSMOD level.

The following format is the syntax for macro DSIID:

```assembly
DSIID
```

Where:

**ONLYDC**

Specifies that only the DC statement will be generated. If ONLYDC is not specified, then the code to branch around the DC statement will be generated.

At the beginning of the CSECT, code the DSIID macro as follows to use DISPMOD to display the information:

```assembly
B MODENTRY
DC CL8'&SYSDATE'
DC CL8'&SYSTIME'
DSIID ONLYDC
MODENTRY DS 0H
```

**DSIKVS: Keyword/Value Services**

You can use macro DSIKVS in a command processor to determine whether or not an operator is authorized to use a given keyword or keyword and value pair.
DSIKVS

The return code shown in register 15 indicates whether the operator who issued
the command has been authorized to issue it with the particular keyword, value,
or both.

MVT addressability is required.

The following format is the syntax for macro DSIKVS:

```
DSIKVS

<table>
<thead>
<tr>
<th>label</th>
<th>DSIKVS SWB= (register)</th>
</tr>
</thead>
<tbody>
<tr>
<td>,CMD= (register)</td>
<td></td>
</tr>
<tr>
<td>symbolic_name</td>
<td></td>
</tr>
<tr>
<td>,SCTADDR= (register)</td>
<td></td>
</tr>
<tr>
<td>symbolic_name</td>
<td></td>
</tr>
<tr>
<td>,KEYWORD= (register)</td>
<td></td>
</tr>
<tr>
<td>symbolic_name</td>
<td></td>
</tr>
<tr>
<td>,VALUE= (register)</td>
<td></td>
</tr>
<tr>
<td>symbolic_name</td>
<td></td>
</tr>
</tbody>
</table>
```

Where:

**CMD**
Is the register containing the address of an 8-byte field with the command
name left-justified and padded with blanks, or symbolic name of that field.

**KEYWORD**
Is the register containing the address of an 8-byte field, or the symbolic name
of an 8-byte field, that contains the keyword, left-justified and padded with
blanks.

**SCTADDR**
Is the register, or symbolic name of a fullword area, containing the address of
the SCT entry for the command that is to be checked.

**SWB**
Is the register, or symbolic name of a fullword area, containing the address of a
service work block (DSISWB).

**VALUE**
Is the register containing the address of an 8-byte field, or the symbolic name
of an 8-byte field that contains a value, left-justified and padded with blanks.
Specify VALUE when you want to check the value of a keyword.

**Return Codes in Register 15**
The following return codes for macro DSIKVS are found in register 15:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified keyword and value, if given, are valid for the operator.</td>
</tr>
<tr>
<td>4</td>
<td>The operator is not authorized to use the keyword.</td>
</tr>
<tr>
<td>8</td>
<td>The specified value is not authorized for this operator.</td>
</tr>
<tr>
<td>12</td>
<td>A required parameter is missing, or a parameter specified in DSIKVS is not valid.</td>
</tr>
<tr>
<td>16</td>
<td>Working storage could not be obtained. No storage is available.</td>
</tr>
<tr>
<td>20</td>
<td>The operator is not authorized to use this keyword or this keyword and...</td>
</tr>
</tbody>
</table>
value combination. This is issued when a command authorization table or an SAF product is being used for command authorization.

24 An unexpected return code was received from the security authorization facility (SAF). Message BNH238E is issued with the SAF return code inserted.

28 Authorization to issue this keyword or keyword and value combination is not granted because the security environment for the operator cannot be established. Message BNH239E is issued when this condition is first encountered to provide the security product return code information. Message BNH273I is issued when the condition has been corrected.

32 Authorization to issue this keyword or keyword and value combination is not granted because an unexpected return code was received from the command authorization table. Message BNH199E is issued indicating the command identifier and the operator ID being checked.

36 Authorization to issue this keyword or keyword and value combination is not granted because the NetView internal security information containing the source ID of the command could not be found. Message BNH277E is issued identifying the command being checked.

40 Authorization to issue this keyword or keyword and value combination is not granted because the source ID is blank in the NetView internal security information. Message BNH277E is issued identifying the command being checked.

Note: If both KEYWORD and VALUE are specified, the combination is checked. Return code 20 indicates a failure of the combination.

DSILCS: Obtain/Release Control Blocks

Macro DSILCS performs one of the following actions:

- Obtains a service work block (SWB) for the caller and places the address of that SWB in a fullword area specified by the CBADDR parameter
- Releases an SWB
- Obtains a command work block (CWB) for the caller and places the address of that CWB in a fullword area specified by the CBADDR operand
- Releases a CWB
- Locates a task vector block (TVB) by operator identification or by LU name
- Locates, from a specified starting position, the next active TVB for a NetView-NetView task (NNT), a hardcopy task (HCT), an operator station task (OST), or an optional task
- Locates a TVB for an operator designated as a receiver of authorization messages by the AUTH statement of a profile definition

Main vector table (MVT) addressability is required.

The following format is the syntax for macro DSILCS:

DSILCS

```
(label) DSILCS CBADDR=(register) symbolic_name
```
Where:

**AUTHRCV**
- Specifies that the routine is to search for the first TVB to locate an operator authorized to receive messages related to successful and unsuccessful logons and lost station messages. AUTHRCV can be explicitly set to NO (the default) only when SWB, CWB, or TVB are also specified.

**CBADDR**
- For the GET and TVB options, this is a register containing the address of a user-provided fullword area on a fullword boundary, or symbolic name of that area. The specified SWB, CWB, or TVB address is returned to this area.
- DSILCS gets global storage (non-queued) for CWB or SWB by issuing a DSIGET with the Q=NO option.
- For the FREE option, CBADDR contains the control block address as either a register that contains the address, or a symbolic name of the control block.

**CWB**
- Specifies the type of operation to be performed on the CWB.

**FREE**
- Specifies that the caller wishes to release the CWB whose address is found in the area specified by the CBADDR operand.

**GET**
- Specifies that the caller needs a CWB. The address of the CWB is returned to the area specified by the CBADDR operand. Before you request NetView services, initialize the CWBTIB field with the address of your TIB.

**LOC**
- Used with CWB=GET or SWB=GET, determines the residency of the work block you are requesting.

**ANY**
- Obtains a work block anywhere in storage.

**BELOW**
- Obtains a work block below 16 MB.
RES
Obtains a work block in storage consistent with the residency of the caller. This is the default.

TEST
The caller of DSILCS sets the high-order bit of register 15 to indicate the type of storage desired. A 0 in the high-order bit means allocate storage below 16 MB, and a 1 means allocate storage anywhere.

LU
Used with TVB, is the register containing the address of an 8-byte LU name field, or symbolic name of that field. This name locates a TVB with a matching LU name.

NEXT
Used with TVB, specifies the TVB to be located for the next task.

HCT
Specifies that the TVB associated with the next active hardcopy task is to be located.

NNT
Specifies that the TVB associated with the next active cross-domain task is to be located.

OPT
Specifies that the TVB associated with the next optional task is to be located.

OST
Specifies that the TVB associated with the next active operator station task is to be located.

PPT
Specifies that the TVB associated with the next active primary program operator interface task is to be located.

OPID
Used with TVB, is the register containing the address of an 8-byte operator identification field, or symbolic name of that field. This name locates a TVB with a matching operator identification.

SWB
Specifies the type of operation to be performed on the SWB.

FREE
Specifies that the caller wishes to release the SWB whose address is found in the area specified by the CBADDR operand.

GET
Specifies that the caller needs an SWB. The address of an SWB is returned to the area specified by the CBADDR operand. Before you request NetView services, initialize the SWBTIB field with the address of your TIB.

TVB
Is the register containing the address of the TVB where the routine begins the search. The routine searches for the TVB specified by LU, OPID, or NEXT, or by the symbolic name of an area containing the address of this TVB.

TVB must be used with LU, OPID, or NEXT; and TVB must not be used with SWB, CWB, or AUTHORCV=YES.
The address of the beginning of this TVB chain is found in the MVTTVB. The TVB address that is found is placed in the area specified by CBADDR after the routine has completed processing.

**Return Codes in Register 15**

The following return codes for macro DSILCS are found in register 15:

- **0**  
The function was successful. The address was returned, or the control block was released.

- **4**  
No active TVBs of the type specified were found.

- **8**  
If TVB was specified, the end of the TVB chain was reached, or an OPID provided is not valid.
  
  If SWB=GET or CWB=GET was specified, no storage was available.
  
  If SWB=FREE or CWB=FREE was specified, a defective control block was detected.

- **12**  
Incorrect parameters were passed to DSILCS.

**Notes:**

1. When a GET request is made, NetView reuses an existing free CWB or SWB rather than issuing a GETMAIN. Even if the user specifies LOC=RES from a command processor above the line, if the only free control blocks are below the line, NetView uses those control blocks.

2. The routine searches the address once to the end of the TVB chain. It does not loop to the beginning of the TVB chain.

For more information about the AUTH statement and unsolicited message routing, refer to the [Tivoli NetView for z/OS Administration Reference](#).

---

**DSILOD: Load User-Defined Module**

Macro DSILOD loads a user-defined module. You specify the name of the module to be loaded.

The following format is the syntax for macro DSILOD:

```
DSILOD
```

```
DSILOD label
        EP=modulename
        EPLoc=(register) symbolic_name
        ,DCB=(register) symbolic_name
```

**Where:**

- **DCB**  
  Is the register, or symbolic name of an area, containing the address of the data control block (DCB) for a partitioned data set to be searched for the module. The DCB must reside above 16 MB.

- **EP**  
  Specifies the name of the module to be loaded.
EPLOC
Is the register, or symbolic name of an 8-byte field, containing the modulename to be loaded. The modulename should be left-justified and padded with blanks.

Return Codes in Register 15
The following return codes for macro DSILOD are found in register 15:

0  The module has been loaded.

Nonzero  The module has not been loaded. Refer to your operating system macro reference for information about system macro return codes.

Notes:
1. You must specify EP or EPLOC, but not both.
2. If the module is successfully loaded, register 0 contains the load-point address of the module. Register 1 contains the authority code in the high-order byte and the module length in doublewords in the low-order 3 bytes.
3. If the module has not been loaded, register 15 contains the return code returned by the system load facility. Register 1 contains the abend code and register 0 contains the reason code for the abend.
4. The module is loaded in virtual storage consistent with the linkage editor attribute RMODE.
5. If AMODE=31, the module is called in 31-bit mode; otherwise it is called in 24-bit mode.

DSIMBS: Message Build Services
Macro DSIMBS puts variable text combined with message text into a buffer that you provide. DSIMBS can determine the size of the buffer required to accommodate the message.

You can supply variable fields to be inserted into NetView messages or unique messages of your own with up to nine varying positional fields.

MVT addressability is required.

The following format is the syntax for macro DSIMBS:

DSIMBS

```
<table>
<thead>
<tr>
<th>label</th>
<th>DSIMBS</th>
<th>SWB= (register)</th>
<th>symbolic_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID= (register)</td>
<td>symbolic_name</td>
<td>equate_name</td>
<td></td>
</tr>
<tr>
<td>MSGA= (pdb1addr, pdb2addr)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Chapter 8. Macros  201
DSIMBS

\[
\text{\texttt{\textbackslash{}MSGTBL= (register)\textbackslash{}symbolic\_name\textbackslash{}OPT=CONCAT} }
\]

PosFields:

\[
\text{\texttt{\textbackslash{}Pn=(text,length,\textbackslash{}padding,side,fill} }
\]

Where:

BFR
Is the register, or symbolic name of a fullword area, containing the address of the buffer in which the edited message is to be returned. BFR must have an initialized BUFHDR. Macro DSIMBS initializes the HDRMLENG, HDRDOMID, and HDRTSTMP fields.

MID
Identifies the message to be edited for the user. You can specify the message:
- By the message number (\textit{nnn})
- In a register
- In a user area specified by a symbolic name
- By the equate name preceded by an asterisk

For example, for MSG999 EQU 999, you could specify MID=*MSG999.

MSGA
Specifies the registers used for variable field substitution in message texts:
\begin{itemize}
  \item \textbf{\texttt{pdb1addr}}
    \begin{itemize}
      \item The address of the parse descriptor block (DSIPDB) or the symbolic name of a fullword area with the address of the PDB. This address contains the addresses and lengths of the variable fields to be substituted into the message text.
    \end{itemize}
  \item \textbf{\texttt{pdb2addr}}
    \begin{itemize}
      \item The address of the PDB or the symbolic name of a fullword area with the address of the PDB. The message skeleton to be edited is located at this address. This is not a NetView message; you supply the message.
    \end{itemize}
\end{itemize}

MSGSIZE
Is the register containing the address of a user-provided fullword area, or symbolic name of that area. Use MSGSIZE only to request the service routine for determining the size of the buffer needed for the message to be edited. When the routine completes processing, the required size is returned in this area.

MSGTBL
Is the register, or symbolic name of a fullword, containing the address of a user-defined NetView message definition module. Use macro DSIMDS to generate this table.

OPT
Specifies that NetView is to search DSIMDM for a specified message identifier
that cannot be found in the specified user-defined NetView message definition module. If you omit OPT=CONCAT, the NetView program searches only the user-defined NetView message definition module for the message identifier.

\textbf{Pn} Used only in combination with the MID operand, specifies the positional fields in a message that are to be replaced by user-supplied text. \textit{n} is a number from 1 to 9. You must specify the first two values, \textit{text} and \textit{length}; the others are optional.

\textit{fill} Is the fill character for the area to be padded. The default fill character is a blank (X'40').

\textit{length} Is the length of the variable text to be substituted into the edited message. The maximum length is 255 characters, specified in character format. It can be a binary value in a register or in a user area specified by a symbolic name. The user area must be a 4-byte fullword.

\textit{padding} Is the length of the variable field to be padded with fill characters. This length must be equal to or less than the length specified by the \textit{length} operand. The maximum length is 255 characters, specified in character format. It can be a binary value in a register or in a user area specified by a symbolic name. The user area must be a 4-byte fullword.

\textit{side} Specifies whether the fill characters are added to the left or right of the data in the field. You can specify this value as L, for left-fill, or R, for right-fill. The default is R.

\textit{text} Is the address of the variable text or the symbolic name of the area with the text to be substituted into the edited message.

\textbf{SWB} Is the register or symbolic name of a fullword area containing the address of a service work block (DSISWB).

\section*{Return Codes in Register 15}

The following return codes for macro DSIMBS are found in register 15:

- \textbf{0} The function was successful. The edited message is in the provided buffer and the length of the message is stored in the message length field of the buffer header; or the size of the message buffer required has been calculated and stored in the area specified by MSGSIZE.

- \textbf{4} The edited message is in the provided buffer, but the message skeleton contained a parameter for which the caller did not supply text. The message contains the characters &n where \textit{n} can be any value 1–9.

- \textbf{8} Unsuccessful. The buffer overflowed, and the message has been truncated. The size of the truncated message has been stored in the message length field of the buffer header.

- \textbf{12} The message number specified could not be found in the NetView or user-specified NetView message definition module. The message \textit{000I} was edited into the caller’s buffer. If only the buffer size is requested, the size of message \textit{000I} is returned.

- \textbf{16} The caller did not supply a buffer address.

- \textbf{20} Combined conditions 4 and 8 occurred.
DSIMBS

24 Combined conditions 8 and 12 occurred.
28 A validity check failed on the user message definition module. The address passed in the MSGTBL operand does not point to a message definition module that was created with macro DSIMDS.
32 The storage request failed.
36 I/O error.
40 Unexpected end of file found.

Notes:
1. Because the variable field information is contained in pdb1addr, you cannot use the P1...P9 operands with MSGA.
2. MID and MSGA are mutually exclusive.
3. BFR and MSGSIZE are mutually exclusive.

DSIMDS: Message Definition Services

Macro DSIMDS generates a message definition module to be used by macro DSIMBS to display messages in installation exits, command processors and subtasks.

After you have coded a message definition module, you must assemble it and link-edit it into a NetView load library. DSIMDS has no return codes.

DSIMVT addressability is not required.

Three forms of DSIMDS are required to generate a message definition module. The following three formats describe these forms:
- Format 1: Start Message Definition Module Statement
- Format 2: Define Individual Messages Statement
- Format 3: End Message Definition Statement

Each format must be coded in the sequence shown.

Format 1: Start Message Definition Module Statement

The syntax for the start message definition module statement follows:

```
DSIMDS
```

Where:
MAXLEN
Defines the maximum message length. MAXLEN is determined by calculating the length of \textit{pppxxxt msgtext} (message number, type, a blank, and text). MAXLEN is a multiple of 71. The valid maximum message size is 213 characters. If you do not specify MAXLEN, the message size defaults to 142 characters.

\textit{prefix}
Is the required positional parameter that becomes the 3-character prefix for the messages in the module. The prefix cannot conflict with the NetView message prefixes:

\begin{verbatim}
AAU  BNT  EGV  EZL  FLB
BNH  CNM  EKG  FKB  FLC
BNI  DS1  EUY  FKV  FMB
BNJ  DJ1  EXQ  FKW  FNA
BNK  DWO  EYV  FKX  IHS
\end{verbatim}

SEARCH
Indicates where the messages can be found: in the message definition module, on disk, or both. SEARCH causes a message definition module to be built. This indicator becomes part of the message definition table.

\begin{itemize}
\item \textbf{B} Indicates that some of the messages can be found in the message definition module, and the others can be found on disk. Code individual message statements using Format 2 of DSIMDS only for those messages that are not defined on disk. See \textit{“Defining Messages on Disk” on page 206}.
\item \textbf{D} Indicates that the messages can be found only on disk. Do not code individual message statements using Format 2 of DSIMDS for the messages. The individual messages are coded on disk. See \textit{“Defining Messages on Disk” on page 206}.
\item \textbf{T} Indicates that the messages can be found only in the message definition module. T is the default.
\end{itemize}

\textbf{TYPE}
Specifies the beginning of generation for the message definition module.

\textbf{Format 2: Define Individual Messages Statement}

The syntax for the define individual messages statement follows:

\textbf{DSIMDS}

\begin{verbatim}
label DSIMDS xxx,‘message_text’,$‘\&\&n’(TYPE= A I D E W)
\end{verbatim}

\textbf{Where:}

\begin{itemize}
\item \textit{label}
  Is an optional label.
\item \textit{message text}
  Is the text of the message added or changed.
\end{itemize}
Is a variable field for text insertion. The positional fields are specified by the $Pn$ parameter in the DSIMBS macro. You can specify $&&1–&&9$.

**TYPE**

Specifies the message type.

- **A** Specifies an action message, one for which appropriate action must be taken.
- **I** Specifies that the message is for information only. No specific action is required.
- **D** Specifies that the action requires a choice or alternative.
- **E** Specifies that the message indicates an error has occurred.
- **W** Specifies that the message is a warning.

**xxx**

Is the message number. It can be any number from 000–999. When DSIMBS is issued to build the message, it creates the message identifier by concatenating your 3-character prefix, the message number, and the type.

When coding your message CSECT, code a message 000 statement to be issued when an incorrect message number is specified. Message 000 has one insert ($&&1$), which contains the incorrect message number. Use wording similar to the NetView message DSI000I:

```
MSG000 DSIMDS 000,'MESSAGE &&1 ISSUED BUT DOES NOT EXIST IN MESSAGE TABLE DSIMDM - CALL IGNORED', TYPE=I
```

You should replace DSIMDM with the name of your NetView message definition module; that is, the name specified on the DSIMDS TYPE=START statement. The DSIMBS service routine substitutes the message number of the incorrect message in place of $&&1$.

**Note:** When coding, be sure that the buffer passed to DSIMBS is large enough to hold the message with all the inserts substituted. Otherwise, the message is truncated.

**Format 3: End Message Definition Statement**

The syntax for the end message definition statement follows:

```
DSIMDS (example 2)
```

```
label DSIMDS TYPE=END
```

**Where:**

- **TYPE** Specifies the end of the message definition module. This is the last statement specified.

The TYPE=END statement is followed by an assembler END statement.

**Defining Messages on Disk**

Messages can be defined on disk instead of, or in addition to, defining them in the message definition module. The benefits of defining messages on disk are:
The message coding is simpler and does not require assembling and link-editing the message definition module when messages are added or changed.

NetView reads in the messages when the DSIMBS macro is issued. Messages can be changed while Netview is running, and the changes take effect immediately.

Messages are coded in members in NetView DSIMSG data sets. The member names are in the format of DSIpppxx, where:

- ppp is the message prefix that was defined on the DSIMDS start message definition module statement.
- xx is the first 2 digits of the 3-digit message number. Within this member, you can define up to 10 messages by incrementing the third digit of the message from 0–9 (xx0–xx9).

The syntax for the user messages follows:

**DSIMDS (example 3)**

```
xxt message_text &n
```

Where:

- `message_text` is the text of the message added or changed. To continue a message on a second line, type an asterisk in column 72, as shown at the end of the second line in Figure 23 on page 208.
- `&n` is an operand for optional information. The positional fields are specified by the Pn parameter in the DSIMBS macro. You can specify &1–&9.
- `t` is the message type:
  - A specifies an action message, one for which appropriate action must be taken.
  - I specifies that the message is only for information. No specific action is required.
- `xxx` is the message number. It can be any number from 001–999.

**Note:** Messages issued from code running with TVBINXIT on cannot be read from disk. You must define these messages in your message definition module with Format 2 DSIMDS statements.

### User Message Definition Module Examples

The following two examples define five user messages (USR001–USR005). The first example, shown in Figure 23 on page 208, defines message USR001, which resides in a message definition module called USRTABLE. The second example, shown in Figure 24 on page 208, defines messages USR002–USR005, which reside in a NetView DSIMSG data set under the member name DSIUSR00.
DSIMMDBS: Call DSIMMDB Service

The DSIMMDBS macro is provided to simplify the call to the DSIMMDB assembler language called service routine. For an example of usage, see "Process Message Data Block Routine (DSIMMDB)" on page 278.

DSIMQS: Message Queuing Services

Macro DSIMQS sends a user-supplied message or command to the message queue of a task vector block (DSITVB).

This message or command appears on the operator’s screen or hardcopy log, depending upon which identification is specified. Buffers that are formatted as internal function requests (IFRs) are not displayed. Instead, they cause the receiving subtask to take the action requested by the IFR. Buffers that are formatted as IFRs are not sent to the authorized receiver (see the description for the AUTHRCV operand). An IFR must have a BUFHDR extension regardless of the BFRFLG value specified on the DSIMQS. The HDRTDISP in the BUFHDR must be X’24’.

When DSIMQS is used to queue a command, the operator ID of the command issuer (source) will be queued along with the command. If DSIMQS is invoked from within an installation exit, the SOURCEID is the identity of the task under which the installation exit is running. If DSIMQS is invoked in a command processor, the SOURCEID is the identity (TVBOPID) of the task that originated the command or the existing SOURCEID at the time the command was issued.

DSIMVT addressability is required.

The syntax for macro DSIMQS follows:

```
DSIMQS label SWB= (register) symbolic_name, BFR= (register) symbolic_name
```
Where:

**AUTHRCV**

Specifies that the first operator designated as the receiver of authorized messages (by the AUTH statement of the profile definition) is to receive the message. All messages sent to the authorized receiver are routed first to the PPT to test for automation and message routing (using the ASSIGN command). If not suppressed by automation or handled by routing, the messages are sent to the authorized receiver, if one is logged on, or to the system console.

The AUTHRCV option is mutually exclusive from LIST and TASKID.

The AUTHRCV option is used only for messages. If the buffer to be sent is formatted as an internal function request (DSIIFR) and it is not an automation IFR containing a message, the DSIMQS macro fails with a return code 4 (incorrect buffer format) in register 15. NO is the default for this operand.

**BFR**

Is the register, or symbolic name of a fullword area, containing the address of a buffer. (If DSIGET obtained this buffer, DSIFRE frees it after use. Use the same option for the Q parameter for both DSIGET and DSIFRE.) BFR requires an initialized BUFHDR.

**BFRFLG**

Specifies whether the subtask that sends the buffer has released control and responsibility for it (BFRFLG=YES). With BFRFLG=YES, the buffer must include HDRMCEXT with HDRSENDR initialized. BFRFLG=NO indicates that the buffer is returned to the issuer of DSIMQS. The issuer must dispose of the buffer.

**Recommendations:**

 Except

```
Options)
```

`register` symbolic_name

`register` symbolic_name

**AUTHRCV**

Specifies that the first operator designated as the receiver of authorized messages (by the AUTH statement of the profile definition) is to receive the message. All messages sent to the authorized receiver are routed first to the PPT to test for automation and message routing (using the ASSIGN command). If not suppressed by automation or handled by routing, the messages are sent to the authorized receiver, if one is logged on, or to the system console.

The AUTHRCV option is mutually exclusive from LIST and TASKID.

The AUTHRCV option is used only for messages. If the buffer to be sent is formatted as an internal function request (DSIIFR) and it is not an automation IFR containing a message, the DSIMQS macro fails with a return code 4 (incorrect buffer format) in register 15. NO is the default for this operand.

**BFR**

Is the register, or symbolic name of a fullword area, containing the address of a buffer. (If DSIGET obtained this buffer, DSIFRE frees it after use. Use the same option for the Q parameter for both DSIGET and DSIFRE.) BFR requires an initialized BUFHDR.

**BFRFLG**

Specifies whether the subtask that sends the buffer has released control and responsibility for it (BFRFLG=YES). With BFRFLG=YES, the buffer must include HDRMCEXT with HDRSENDR initialized. BFRFLG=NO indicates that the buffer is returned to the issuer of DSIMQS. The issuer must dispose of the buffer.

**Recommendations:**
When using DSIMQS BFRFLG=YES, the following are recommended practices:

- All storage should be obtained using DSIGET,Q=NO,SUBPOOL=0.
- Buffers should not contain addresses of nonbuffered data that will be freed by other tasks. This practice causes inaccurate storage accounting.

**Note:** Assembler programs containing addresses of data should specify MAINTSK=YES on the DSIGET request for the data pointed to by the buffers. The buffers themselves should not have MAINTSK=YES specified on DSIGET. Only data contained in the buffer should be sent using DSIMQS BFRFLG=YES.

DSIGET MAINTSK=YES is intended to keep storage accounting accurate where normal storage management cannot be used, for example, when DSIGET is used to obtain storage in one task, free the same storage using DSIFRE in another task, and not transfer the buffer using DSIMQS.

- You should use the Automation Internal Function Request (AIFR) buffer structure, IFRAUTO in DSIIFR, to send multiple buffers. All buffers in an AIFR must be separately obtained using DSIGET,Q=NO,SUBPOOL=0. For more information about DSIIFR, see "DSIIFR: Internal Function Request" on page 125.

- When using DSIMQS BFRFLG=YES, specify STGACCT=YES if at all possible.

**CORR**

When CORR=None is coded, no correlation is provided for the buffer being sent. None is the default.

When CORR=CMD is coded, the data being sent will be treated as a command by the receiving task. Any output (for example, DSIPSS TYPE=OUTPUT) from that command will be returned to the environment from which the message queuing service (MQS) is issued. There are two environments:

1. If the MQS occurred within a pipeline (within a command running under the NetView stage), the pipeline is the environment. The output (from the other task) is introduced into the pipeline through CORRWAIT.
2. If not in a pipeline, then the environment is the task from which the MQS was issued.

Under DST, execution of DSRB-based functions DSIZCMSMS and DSIZVSMS preserves environment information. Output produced from the command that is notified of completion of a DSIZCMSMS or DSIZVSMS function will be processed using the same rules as the command that issued the DSIZCMSMS or DSIZVSMS.

**Example Scenario 1:**

OPER1 executes CMD1 outside of a pipeline. CMD1 sends (MQS CORR=CMD) another command, CMD2, to another task such as DST1. CMD2 then issues a message through DSIPSS TYPE=OUTPUT. This output will be displayed at OPER1.

**Example Scenario 2:**

Using the same commands as in Example Scenario 1, OPER1 executes (from within a command list):
The first two outputs from CMD2 will be found in the stem variables. Any further output from CMD2 is discarded.

EXCEPT
Specifies an 8-character field that contains an operator or group ID, left-justified and padded with blanks, that does not receive the message. You can specify this operand only if you specify LIST.

LIST
Is the pointer to, or symbolic name of, a fullword area containing the address of a list of operator IDs or group IDs to receive the message. Operators are assigned to groups using the ASSIGN command. Refer to the NetView online help for more information about the ASSIGN command.

The LIST option is mutually exclusive from AUTHRCV and TASKID. For this operand:
- If you specify 1ST as the LIST type, the first logged-on operator in the list receives the message. (The first logged-on operator can be in a group.) The receiving operator ID is returned in the MQSENTTO field in the SWB.
- If you specify ALL as the LIST type, and a return code of 0 is received, the message is sent to all specified operators and groups of operators in the list who are logged on. While the message is sent to all specified logged-on operators, it is not necessarily received.
- If you specify multiple operator or group IDs, the last two fields (shown in the ID list following) must be repeated for each operator or group listed.

The following ID list contains hexadecimal offsets:
0 1ST or ALL (3 bytes)
3 Number of IDs in list (1 byte)
4 Unused (8 bytes)
C Operator or group ID (8 bytes)

See "Return Codes in Register 0" on page 212 for the return codes in register 0 when the LIST option is specified.

PRI
Specifies a priority for message processing by the destination task. The value is one of the four strings HIGH, NORMAL, LOW, OR TEST, or a register or name of a fullword area containing one of the values defined in DSISWB: MQSHI, MQSNORM, MQSLO, or MQSTEST. The default value is NORMAL. A message or command sent at HIGH priority would begin processing after any normal message currently in progress, but before other queued NORMAL messages. A message or command sent at NORMAL priority would similarly preempt a queue of LOW priority messages. When you specify TEST, DSIMQS queues the message at either high or low priority according to the destination task’s command priority. Refer to the OVERRIDE command in the NetView online help for an explanation of command priorities.

Any command that is currently running can be interrupted at that point in processing where control is returned to Netview. For example:
- A running command processor is not interrupted. The newly queued command (no matter which priority queue) runs when the command processor returns to the command facility after issuing a DSIPUSH macro (if the command is a compound command processor) or when the processor finishes.
A running command list (REXX or command list language) can be interrupted under several conditions. For example, an interruption can occur just after a command list begins processing and is scheduled by the NetView program but is not yet running. Other examples are when a command list issues a command or performs an &WAIT or WAIT.

Of the NetView-supplied tasks, only destination task types OST, PPT, and NNT recognize priority. For destination tasks that have not indicated support for multiple priorities, DSIMQS automatically converts all messages to NORMAL priority.

**STGACCT**
Specifies whether the storage specified by BFR was obtained using DSIGET. STGACCT is only valid when BFRFLG=YES.

**YES**
Indicates that the storage specified by BFR was obtained using DSIGET.

**TEST**
Indicates that the origin of the storage specified by BFR is unknown. Specify STGACCT=TEST when you cannot determine if the buffers were obtained using DSIGET. TEST is the default.

**SWB**
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**TASKID**
Is the register containing the address of a user-provided 8-byte area, or symbolic name of that area, or PPT for the primary POI task. The area contains the 8-character operator identification (TVBOPID) of the task for which the message is to be queued.

The TASKID option is mutually exclusive from AUTHRCV and LIST.

**Return Codes in Register 0**
The following return codes for macro DSIMQS can appear in register 0 if the LIST option is used:

- **0** At least one operator was active and all active operators received the buffer.
- **12** The attempt to obtain buffer storage failed and one or more active operators did not receive the buffer.
- **23** The message was routed to the first 255 active operators or groups, or both, in the list.
- **26** One or more tasks in the list do not support MQS with Receipt buffer. The buffer has been sent to all tasks in the list capable of receiving it.

**Return Codes in Register 15**
The following return codes for macro DSIMQS are found in register 15:

- **0** The function was successful; the message is queued.
- **4** The buffer length was either:
  - Not greater than 0
  - Less than the combined length of HDRBLEN plus HDRTDISP
  - Greater than 32000
The operator ID designated as the receiver of authorized messages was not found.

A buffer could not be obtained or dynamic resource control failed.

NetView is terminating; the external request cannot be completed.

The SWB address is not valid.

The list specified with the LIST option contained no operator IDs. It contained only unassigned group IDs.

Messages were routed to the first 255 operators or groups, or both.

The value specified for priority was not valid.

The internal function request for the command to be run contains the IFRAUTBC or IFRAUTBN fields. The task that receives this command has no MQS receipt support and cannot process these fields.

A message stack enquiry failed.

NetView internal error.

Note: When a command procedure written in REXX or NetView command list language is executing, NetView services all message queues, except the low-priority queue, at three points:
- Initially, before the first instruction
- After the execution of any NetView command
- Throughout the period of any wait state (for &WAIT, &PAUSE, or WAIT)

Because of this, two command lists queued at the same time to the high- or normal-priority queues appear to run in reverse order. The first one is initiated, then before it executes its first instruction, it is preempted, and the second command executes. To have command lists execute in the order queued, always queue them at low priority.

For more information about the AUTH statement and unsolicited message routing, refer to the Tivoli NetView for z/OS Administration Reference.

---

**DSINOR: Resource Object Data Manager**

The DSINOR service routine enables interaction with a specified resource object data manager (RODM). All RODM application programming interface (API) functions are supported through this interface, including querying for data, changing data, and triggering methods.

The syntax for macro DSINOR follows:

```
DSINOR

label,DSINOR ACB=(register),TIF=(register),symbolic_name,RESP=(register),FUNC=(register),SWB=(register),symbolic_name
```

---

Chapter 8. Macros 213
Where:

ACB

Is a RODM access control block following the format of the RODM API.

The access block contains the following fields:

orname

This field specifies the name of the RODM that the caller wants to access. If this field is blank (X'40'), the current run-time RODM is used. The current run-time RODM is defined in the DSIQTSKI initialization member in DSIPARM with the AO parameter on the REP keyword. The name is left-justified and must be padded with blanks (X'40') to 8 characters.

signon_token

Specifies the RODM sign-on token to be used within the call.

DSINOR ignores this field and fills it with the sign-on token received by DSIQTSK when it initially connects to the RODM being accessed. Because DSINOR overrides this field, you can ignore it.

user_appl_id

This field specifies the application name of the caller.

DSINOR sets this field to the user application specified with the ID parameter of the REP keyword (of DSIQTSKI) for the RODM being accessed by this call.

FUNC

Specifies a varying length function block, following the format of the RODM API function block, that describes the function requested and all required parameters. The actual function block format depends on the function being requested.

RESP

Specifies a response block following the format of the RODM API response block control structure.
SWB
Specifies a register, or the symbolic name of a fullword area, containing the address of a caller’s NetView service work block (SWB) control block.

TIF
Specifies a transaction information block following the format of the RODM API transaction information block and contains the following fields.

WAITF
Specifies whether the request should wait when a checkpoint is detected. If a checkpoint is in progress for the specified RODM, the request is placed on a queue until the checkpoint is complete. Upon checkpoint completion, the request is processed. Recognized values are:
N: Do not wait for checkpoint completion. This is the default.
Y: Wait for checkpoint completion.

WAITT
Is a 2-byte field that specifies the maximum time in seconds for which the call should be suspended if a checkpoint wait is to be invoked. The expected value range is 10 to 3600 seconds (1 hour). If you specify a time greater than 3600, 3600 is used. If you do not specify this field, the default specified with the T keyword of the DSIQTSKI initialization member for the DSIQTSK task is used.

Return Codes in Register 15
The following return codes for macro DSINOR are found in register 15:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The function was successful</td>
</tr>
<tr>
<td>4</td>
<td>RODM not under control of RODM access and control component</td>
</tr>
<tr>
<td>8</td>
<td>DSINOR or DSIGET failure. Internal macro call failure; possible storage problem.</td>
</tr>
<tr>
<td>12</td>
<td>Incorrect parameters received</td>
</tr>
<tr>
<td>20</td>
<td>Checkpoint in progress</td>
</tr>
</tbody>
</table>

Notes:
1. DSINOR applies only to those RODMs under the control of the DSIQTSK task. Refer to the [Tivoli NetView for z/OS Automation Guide](#) for an example of managing your RODMs with DSIQTSK in a NetView automation scenario that uses RODM.
2. For an application that uses DSINOR in a command processor, provide a keyword (which can be authority-checked) for controlling the level of access to RODM.
3. An application can connect to RODM with an option specifying that RODM can truncate its responses if the application’s response block is smaller than RODM’s response. If this option is used, RODM truncates the response, does not save the overflow data, and informs the application of the condition. This information is sent in the return code and reason code in the transaction information block. DSIQTSK connects using this option, saving DSINOR from having to deal with overflow cleanup.
4. DSINOR returns two sets of return codes. The caller can use either set. The first set is in the transaction information block provided by the user upon invocation. These return codes are RODM return codes and are documented for each possible function in the [Tivoli NetView for z/OS Resource Object Data Manager](#)
DSIPAS: Parameter Alias Services

Macro DSIPAS receives a command parameter as input and searches the system command table (DSISCT) to determine whether the entered parameter is an alias for the actual parameter.

If the parameter is an alias, the regular value is returned to a user-provided area. If it is not an alias, the input value is returned to the user area. If the value is not valid, blanks are returned to the input area.

DSIMVT addressability is required.

The syntax for macro DSIPAS follows:

```
DSIPAS
```

Where:

OUT

Is the register containing the address of a user-provided 8-byte area to which the NetView equivalent of the input operand is returned if found; or the symbolic name of that user area.

PDB

Specifies two values. One value is the address of a PDB and the other value is the entry number of the field in the PDB to be examined.

```
combo
```

The value for combo can be specified using any of the following forms:

- `pdbname, entname`
- `pdbname, (entreg)`
- `pdbname, 'ENTRY'`
- `(pdbname), entname`
- `(pdbname), (entreg)`
- `(pdbname), 'ENTRY'`

entname

Is the symbolic name of a fullword that contains the entry number, right-justified, and padded with binary zero.

entreg

Is the register containing the entry number and padded with binary zero.
ENTRY
Is a constant that specifies the entry number.

pdbname
Is the symbolic name of a fullword that contains the address of the PDB.

pdbreg
Is the register that contains the address of the PDB.

SWB
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

Note: PDBCMDA must contain the address or pointer to an entry in the SCT.

Return Codes in Register 15
The following return codes for macro DSIPAS are found in register 15:
0 A regular operand value was returned.
4 No equivalent was found; the same operand is returned.
8 Incorrect operand; blanks are returned.

DSIPOP: Remove Long-Running Command
Macro DSIPOP removes a long-running command element that DSIPUSH placed on the stack.

The element canceled is the one nearest the top of the stack with the name specified in the parameter list. If a calling command procedure is suspended by DSIPUSH, use DSIPOP to let the command procedure continue at the next instruction. The command procedure continues when the RESUME routine or currently running command returns control to the OST or PPT.

Do not use DSIPOP while in a LOGOFF routine, an abend reinstate routine, or an immediate command.

See "DSIFIND: Find Long-Running Command Storage" on page 173 and "DSIPUSH: Establish Long-Running Command" on page 229 for more information.

DSIMVT addressability is required.

The syntax for macro DSIPOP is:

```
DSIPOP
```

```
label DSIPOP SWB= (register) symbolic_name
```

```
LIST= (register) symbolic_name COMPCDE
```

COMPCDE

\[ \text{COMPCDE} = (\text{register}) \text{ or } \text{symbolic \_name} \]

Where:

COMPCDE

Specifies the value of the completion code for the long-running command being removed. You can specify the value as a register, symbolic name of a fullword area, or a fullword literal. The value is meaningful only if the long-running command being removed specified a RESUME routine and only if the process that created the long-running command element (using DSIPUSH) was directly invoked from another long-running command. If you do not specify COMPCDE, the default values are:

- **0**: If the long-running command being removed is in control on top of the stack at the time DSIPOP is invoked. This is the usual (and recommended) case.
- **−5**: If the long-running command being removed is not at the top of the stack. NetView command procedures and certain related commands treat negative 5 (−5) as a CANCEL request by NetView command procedures and certain related commands.

You can be certain that your long-running command is at the top of the stack when it is resumed.

Note: NetView command procedures are long-running commands. If you have written a long-running command to be called from a command procedure, you can pass a return code to it using the COMPCDE keyword. If a long-running command makes a direct call to schedule a command procedure, you can obtain its return code upon resumption from CWBRCODE.

LIST

Is the register containing the address of a parameter list used by the service routine, or symbolic name of that list. Do not specify this as register 1; register 1 contains the SWB address within DSIPOP. Do not put this list in the SWB that is to be passed to DSIPOP.

The parameter list contains the following fields:

<table>
<thead>
<tr>
<th>Hex Offset</th>
<th>Length</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>SWBLRCLN (Length)</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>SWBLRCNM (Name)</td>
</tr>
</tbody>
</table>

Where:

SWBLRCLN

Specifies the parameter list length. Set SWBLRCLN equal to SWBLRCP0 (decimal 20).

SWBLRCNM

Specifies the name of the storage to be de-queued and freed. Specify this
field exactly as you specified it in the corresponding macro DSIPUSH. This 16-byte field is used as is. Instructions for specifying the name field are under “DSIPUSH: Establish Long-Running Command” on page 229.

**SWB**

The register, or symbolic name of a fullword area, that contains the address of a service work block (DSISWB). The TIB address in SWBTIB must be correctly set.

**Return Codes in Register 15**

The following return codes for macro DSIPOP are found in register 15:

- **0** The function was successful; the long-running command processor element is dequeued.
- **16** The request was issued while in an immediate command, or while the NetView program was in an exit, a LOGOFF, or an ABEND reinstate routine.
- **32** Incorrect macro call. Fix assembly errors before trying to run the program.
- **36** The specified NAME was not found.

**DSIPOS: ECB Post Services**

Macro DSIPOS indicates the completion of an event by posting an event control block (ECB).

DSIMVT addressability is required.

The syntax for macro DSIPOS is:

```
DSIPOS
```

```
label DSIPOS ecbaddress, compcode
```

Where:

- **compcde**
  
  Is the value of the completion code to be placed in the ECB (0–16777215) or in a register (0, 2–12) that contains the value. If you specify a register, code it in parentheses. If you do not specify a value, 0 is assumed.

- **ecbaddress**
  
  Is the symbolic name of an ECB or register (1–12) that contains the address of the ECB. If you specify a register, enclose it in parentheses.

**Note:** These parameters are positional; they must be specified in the indicated order.
Macro DSIPRS parses commands using specified or assumed delimiters, or determines the size of the parse table required to parse the input buffer. DSIPRS must be issued in pairs. Specify the PDBSIZE in the first issuance, to determine the parse descriptor block (DSIPDB) length necessary to parse the command. The second issuance actually passes the command into a buffer long enough to hold the PDB.

The parse table describes the data contained in the buffer. DSIPRS finds delimiters in the data and formats the PDB to indicate data segments separated by the delimiters. You can call DSIPRS again to build the parse table in a user-provided area.

DSIMVT addressability is required.

The syntax for macro DSIPRS is:

```
DSIPRS

(label)

DSIPRS SWB=

/register /

(symbolic_name)

BFR=

/register /

(symbolic_name)

,POBSIZE=

/register /

(symbolic_name)

,PDB=

/register /

(symbolic_name)

,FIRST=YES

,FIRST=NO

,YES

,NO

,SUB=NO

,YES

,NO

DELIM

,DELIM=(

'delimiter'

)
```

Where:

**BFR**

Is the register, or symbolic name of a fullword area, containing the address of the buffer to be used for input. BFR must have an initialized BUFHDR.

**DELIM**

Enables you to specify delimiters instead of NetView defaults. NetView default delimiters are blank, comma, period, and equal sign. Blank is always considered a delimiter, even if you specify your own delimiters.

**FIRST**

Indicates whether the first word of the input buffer can be delimited only by a blank (YES) or by any delimiter (NO). YES is the default.
PDB
Is the register containing the address of a fullword pointing to the area where
the parse table is to be built, or symbolic name of that area. The parse table
must include a user-initialized DSICBH header that contains the control block
identification and length before the data can be parsed. The name of the
constant set to PDBCBH is CBHPDB.

PDBSIZE
Is the register containing the address of a fullword area to which the size of
the parse table is to be returned, or symbolic name of that area.

SUB
Indicates whether all text within single quotes is to be parsed as one element
(YES) or not (NO). This option treats everything between single quotes as one
element, provided the first quote is preceded by a delimiter and the last quote
is followed by either a blank or comma. NO is the default.

SWB
Is the register, or symbolic name of a fullword area, containing the address of a
service work block (DSISWB).

Return Codes in Register 15
The following return codes for macro DSIPRS are found in register 15:

0  The function was successful. The required size of the table was returned in
    PDBSIZE, or the command was parsed and the parse table was built.

4  The input buffer was parsed, but there was no data in the input buffer (0
    length data) or the data in the input buffer was all blanks. Only the buffer
    address and number of entries (0) could be returned in the parse table.

8  The parse table was too small for the input buffer; a partial parse table was
    built, and the number of entries was set to the number that the parse table
    could hold. The size of the parse table should be increased.

12  Unbalanced quotes. Returned only if SUB=YES is specified.

16  The number of characters between two consecutive delimiters in the input
    buffer was greater than 255.

20  An unpaired double-byte character set (DBCS) delimiter of DBCS data
    bytes was found in the input buffer. For example, one of the following may
    have occurred:
    • The end of the input buffer was found before the DBCS data-ending
      delimiter shift-in (X'0F').
    • A second DBCS data-beginning delimiter shift-out (X'0E') was found
      before the DBCS data-ending delimiter shift-in (X'0F').
    • An odd number of DBCS data bytes were found between DBCS data
      delimiters.

100 No PDB or an incorrect PDB was passed; or no PDBSIZE or an incorrect
    PDBSIZE was passed.

Note: You must specify the operands DELIM, FIRST, and SUB, identically, in the
pair of DSIPRS parse commands issued. Otherwise, the second parse can fail
or the storage can be overlaid.
DSIPRS

Parsing Services Module Examples

In the following examples, DSIPRS is issued to pass the given character string with
the default delimiters and SUB=YES.

Example 1
RETURN CODE IS 'NONZERO'!

DSIPRS returns the UNBALANCED QUOTES return code, if you specify
SUB=YES.

Example 2
RETURN CODE IS 'GOOD', CONTINUE.

The following PDB table is built:

<table>
<thead>
<tr>
<th>PDBTYPE</th>
<th>PDBLENG</th>
<th>PBDISP</th>
<th>TOKEN VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>6</td>
<td>24</td>
<td>RETURN</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>2B</td>
<td>CODE</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>30</td>
<td>IS</td>
</tr>
<tr>
<td>,</td>
<td>4</td>
<td>34</td>
<td>GOOD</td>
</tr>
<tr>
<td>,</td>
<td>8</td>
<td>3C</td>
<td>CONTINUE</td>
</tr>
</tbody>
</table>

Example 3
RETURN CODE IS (X'00'), CONTINUE.

The following PDB table is built:

<table>
<thead>
<tr>
<th>PDBTYPE</th>
<th>PDBLENG</th>
<th>PBDISP</th>
<th>TOKEN VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>6</td>
<td>24</td>
<td>RETURN</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>2B</td>
<td>CODE</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>30</td>
<td>IS</td>
</tr>
<tr>
<td>'</td>
<td>2</td>
<td>33</td>
<td>X</td>
</tr>
<tr>
<td>'</td>
<td>2</td>
<td>36</td>
<td>00</td>
</tr>
<tr>
<td>'</td>
<td>2</td>
<td>39</td>
<td>)</td>
</tr>
<tr>
<td>,</td>
<td>8</td>
<td>3C</td>
<td>CONTINUE</td>
</tr>
</tbody>
</table>

DSIPSS: Presentation Services

Macro DSIPSS writes a message to an operator’s screen or sends messages to
another NetView program. The presentation service routines called by DSIPSS
control screen formats, organize data into a specific form for each device, and send
the data.

DSIMVT addressability is required.

The syntax for macro DSIPSS is:
Notes:

1. Values of SEG, FIRST, MIDDLE, LAST, and ONLY are not fully supported in NetView.

Where:

**APPLID**
Is the register containing the address of an 8-byte area that contains the name (left-justified and padded with blanks) of the application program to which the data is to be sent, or symbolic name of that 8-byte area. This name should be the same as the name specified on the START command when a session is started. Specify APPLID only when you specify TYPE=XSSEND.

**BFR**
Is the register, or symbolic name of a fullword area, containing the address of a user-provided buffer. This buffer should contain the data to be processed. Use BFR only for TYPE=FLASH, TYPE=OUTPUT, TYPE=IMMED, and TYPE=XSSEND. BFR must have an initialized BUFHDR.

**ECBLIST**
For TYPE=PSWAIT, this is the register, or symbolic name of a fullword area, containing the address of an ECB list. An ECB list is a list of addresses of user-defined ECBs that is copied and combined with another ECB list. The NetView program waits for this combined list. When one of the events
associated with this list is posted, control is returned to the next sequential
instruction. The input ECB list is made up of fullword ECB addresses. The last
address in the list must have the first bit set on to specify that this is the last
entry.

PANEL
For TYPE=ASYPANEL, a register containing the address of a parameter list, or
the symbolic name of that list. You can use one the following types of
parameter lists:

- Settable PF keys enabled; NetView manages the PF keys on the panel. Use
  the DSIASYPN macro to create the parameter list. The macro comments
  explain how to code your parameters.

- Settable PF keys not enabled; your code manages hardcoded PF keys. Use
  the parameters shown in Table 48.

The parameter list for panels without settable PF keys is formatted as shown in
Table 48.

Table 48. PANEL Parameters for the DSIPSS Command

<table>
<thead>
<tr>
<th>Bytes (Decimal)</th>
<th>Bytes (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0)</td>
<td>ECB address</td>
</tr>
<tr>
<td>4</td>
<td>(4)</td>
<td>Output data stream address</td>
</tr>
<tr>
<td>8</td>
<td>(8)</td>
<td>User input area address</td>
</tr>
<tr>
<td>12</td>
<td>(C)</td>
<td>Output length</td>
</tr>
<tr>
<td>16</td>
<td>(10)</td>
<td>Data length address</td>
</tr>
</tbody>
</table>

If you request asynchronous full-screen output, the output data stream address
field contains the address of a 3270 data stream, including a 3270 command,
write control character (WCC), and orders to be written to the terminal. Code
the command using remote EBCDIC values. The output length field indicates
the length, in bytes, of the 3270 data stream (32767 bytes maximum). If output
is requested, the ECB address, input area length, user input area address,
and data length address fields are not used.

To read asynchronous full-screen input from a terminal, the ECB address area
contains the address of an ECB to be posted when the asynchronous input is
received. The user input area address contains the address of a user area into
which the full-screen panel data is read. If the length of the data being read is
greater than the user input area, the data is truncated in that area. The input
area field indicates the length of the input data area in bytes (32767 bytes
maximum). The data length address field contains the address of a halfword
field set to the amount of data read when the ECB is posted.

OPTIONS
Provided only for downward compatibility to previous NetView releases,
specifies the type of message to be sent. Use OPTIONS only for
TYPE=OUTPUT. The default is MSG, which specifies that the data to be sent is
a complete message.

In general, NetView does not support OPTIONS for DSIPSS, but attempts to
display messages with results similar to prior releases. Use title-line output
instead of the other values for the OPTIONS operand you might have used in
previous releases.
SIZE

For TYPE=SCRSIZE, this is the register, or symbolic name of a fullword area, containing the address of a user-provided fullword area to contain the size of the display screen, in row-column format. For example, a 1920-character screen is defined as X'00180050', because the screen is 24 rows (X'0018') by 80 characters (X'0050').

For TYPE=WINDOW, this is a register containing the address of a 12-byte area, or symbolic name of that area. The window size is returned in binary to the area. The window size is the number of lines available for output on the screen. The size varies depending on screen size and the number of input lines specified on the INPUT command. The syntax for the area is:

<table>
<thead>
<tr>
<th>Bytes (Decimal)</th>
<th>Bytes (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(2)</td>
<td>Minimum window size, rows</td>
</tr>
<tr>
<td>4</td>
<td>(4)</td>
<td>Minimum window size, columns</td>
</tr>
<tr>
<td>6</td>
<td>(6)</td>
<td>Current window size, rows</td>
</tr>
<tr>
<td>8</td>
<td>(8)</td>
<td>Current window size, columns</td>
</tr>
<tr>
<td>10</td>
<td>(A)</td>
<td>Maximum window size, rows</td>
</tr>
<tr>
<td>12</td>
<td>(C)</td>
<td>Maximum window size, columns</td>
</tr>
</tbody>
</table>

SWB

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

TYPE

Type of presentation services routine to be called:

ASYPANEL

Specifies that the issuing routine assumes control of the screen. Input and output are formatted as 3270 data stream commands. Notification of input availability is done asynchronously by the posting of event control blocks (ECBs).

After a DSIPSS TYPE=ASYPANEL, input to the terminal is treated as input to the process issuing the ASYPANEL request until a DSIPSS TYPE=CANCEL or a DSIPSS TYPE=OUTPUT is issued.

For normal operations, presenting a panel to a user and processing responses to it, specify both output data and input parameters in the PANEL parameter list. Using separate DSIPSS TYPE(ASYPANEL) macros for output and input introduces the possibility that operator input data may be discarded before the input DSIPSS can be processed.

Full-screen mode is not supported for unattended operator tasks, including those associated with a system console.

CANCEL

Cancels pending asynchronous full-screen input. Use this option when changing the characteristics of the asynchronous full-screen processor, such as the ECB address or the panel address. TYPE=CANCEL is valid only from an OST. You can invoke this option regardless of whether a DSIPSS TYPE=ASYPANEL is active or the input from TYPE=ASYPANEL has been posted as complete.
After TYPE=CANCEL is issued, no further input is received from the terminal until TYPE=OUTPUT, TYPE=IMMED, or TYPE=ASYPANEL is issued.

Issue DSIPSS TYPE(CANCEL) only when a program is going to free up or change the storage areas used for output, input, ecb, or parmlist for a previously issued DSIPSS TYPE(ASYPANEL), usually at the end of a program. If a full-screen command processor is long running, “ending” assumes issuing a DSIPOP and a return to NetView rather than postponing the return to later. Issuing unneeded CANCELs may cause operator input to be discarded between the CANCEL and the next TYPE(ASYPANEL) or TYPE(OUTPUT).

**Note:** For NetView versions prior to V3, DSIPSS TYPE(CANCEL) is required anytime a fullscreen command returned to NetView. If code must run on NetView V3 or prior versions, check MVTVER. If MVTVER returns something less than NV31, issue the CANCEL.

**FLASH**

These messages are not suppressed by &WAIT or WAIT processing or NetView automation, nor are they logged. These messages are not exposed. You can log them before calling DSIPSS if you choose. They are displayed regardless of the STIFLE state.

**IMMED**

 Specifies that the routine is to send a message to the operator work station’s immediate message area. The maximum message length before truncation occurs is 70 characters. Use this option only in immediate command processors or the DSIEX01 installation exit routine. When you specify this operand, no message header information is sent to the display screen. TYPE=IMMED terminates full-screen mode and causes subsequent terminal input to be treated as commands, except for the output of the following commands:

- BROWSE (member or netlog)
- NLDM
- NPDA
- STATMON
- TARA
- VIEW (NOINPUT option or special variables)

TYPE=IMMED also does not terminate the output of DSIPSS TYPE=ASYPANEL using PANEL parmlist-2.

DSIPSS calls DSIEX02A, message copy (ASSIGN), and logging functions.

**OUTPUT**

Specifies that the routine is to send a message to the operator’s terminal. Do not use this option:

- In immediate command processors
- In installation exit routines with TVBINXIT set on
- In installation exit DSIEX12

The maximum message length before truncation is 32000 characters for an OST and 256 characters for an NNT. Upon completion of the macro, the length of the text in the HDRMLENG field of the BUFHDR is set to the length of the data after any trailing blanks have been truncated.
A command may be sent from an NNT to its associated OST if a DSIPSS TYPE=OUTPUT passes either a HDRTYPEX buffer or a HDRTYPEI, IFRCODAI buffer with IFRAUTBA and IFRAUTBL pointing to a HDRTYPEX buffer.

DSIPSS calls DSIEX02A, message copy (ASSIGN), and logging functions.

**PSSWAIT**

Specifies that a command is to wait for a list of its own events and a list of events that can interrupt the command events. TYPE=PSSWAIT is only of use if issued by a command processor driven as a resume or abend routine. A logoff routine cannot issue a DSIPSS macro. TYPE=PSSWAIT is valid only from an OST.

**Note:** Use macro DSIWAT if you do not want the command to wait for the completion of events.

**SCRSIZE**

Specifies that the routine is to return the screen size in row-column format.

**TESTWAIT**

Enables a command processor to test whether an event has occurred that should interrupt the asynchronous full-screen command processor. TYPE=TESTWAIT is only of use if issued by a command processor driven as a resume or abend routine. A logoff routine cannot issue a DSIPSS macro. TYPE=TESTWAIT is valid only from an OST. You can use this option before issuing a DSIPSS TYPE=ASYPANEL to determine if the asynchronous full-screen panel input/output (I/O) should be attempted. If you use DSIPSS TYPE=PSSWAIT to wait for events, this option can prevent unnecessary screen I/O by enabling testing before panel I/O is requested.

**WINDOW**

Requests information about the size of the output area of the standard screen. This option is valid only from an OST. Under any other task, the request is considered null; register 15 contains a return code of 0, but no function is performed. Three output area sizes are returned:

- Minimum
- Current
- Maximum

You can use the minimum window size to produce panels that are independent of the current window size. The current window shows the panel size currently in effect. The maximum window size is useful for calculating the maximum storage needed to produce title-line panels.

**XSEND**

Specifies that the routine is to send data to another NetView program with which a session exists. Sessions are started with the START DOMAIN= command. The maximum data length before truncation is 240 characters.

### Return Codes in Register 15

The following return codes for macro DSIPSS are found in register 15:

- **0** The function was successful; the message is written. For TYPE=PSSWAIT, an ECB has been posted. Check the ECB list to determine which event has completed. For TYPE=ASYPANEL, the send or receive request has passed NetView syntax and buffer checking and has been sent to VTAM; it does not indicate the success or failure of VTAM completion of the receive. You
must check the ECB post code to determine the success or failure of the ASYPANEL request. The post code is put into the ECB specified in the panel parameter list.

4 For TYPE=XSEND, no request parameter list (RPL) was found and no data was sent.

8 Parameter error. There is an error in the formatting of the message buffer header. For TYPE=XSEND, the session is not active and no data is sent. For TYPE=ASYPANEL, the parameter list is inconsistent. If you specify the output buffer, you must also specify the length. If you specify the input ECB, you must specify the input area address, input area length, and the data length address of the returned length.

12 The NetView program does not have enough storage available to complete the request. No output is sent, and the input command processor cannot be scheduled.

16 DSIPSS TYPE=OUTPUT was issued for an immediate command or in an IRB exit routine. Use DSIPSS TYPE=IMMED or DSIMQS instead.

20 No terminal session exists. For TYPE=ASYPANEL, the panel request came from a task other than an OST. No input is received. For TYPE=CANCEL, the panel request came from a task other than an OST.

36 For TYPE=ASYPANEL, a temporary error occurred. The contents of the panel have been modified. Reformat the panel using an Erase/Write or Erase/Write Alternate 3270 command. Then retry the request.

40 A permanent I/O error occurred. Do not retry the request. No output is sent, and no input processor is scheduled. For TYPE=ASYPANEL, no input is received. For TYPE=CANCEL, the NetView program is unable to restart normal terminal activity.

48 For TYPE=ASYPANEL, no I/O is scheduled because the command processor issued a second DSIPSS TYPE=ASYPANEL requesting input before the previous request had completed.

56 For TYPE=PSSWAIT or TYPE=TESTWAIT, at least one NetView ECB was posted.

68 For TYPE=OUTPUT or TYPE=IMMED, a message being processed for the RMTCMD command failed to be transmitted. This error can occur when the DSJUDST task is inactive.

The following ECB post codes for PSS TYPE=ASYPANEL are found in the ECB if you specified one:

0 The function was successful; the requested data is available.

12 The NetView program does not have enough storage available to complete the request. The output data was sent, but the input data is not available.

36 A temporary error occurred during a full-screen read. Retry the request. The output data was sent, but the input data is not available.

40 A permanent error occurred during a full-screen read. Do not retry the request. The output data was sent, but the input data is not available.

52 The requested input was canceled by DSIPSS TYPE=CANCEL. Do not retry the request immediately. The output data was sent, but the input data is not available.
Notes:

1. The DSIPSS macro no longer supports TYPE=PANEL. If you have an application that uses the DSIPSS macro with TYPE=PANEL, rewrite the application using TYPE=ASYPANEL.

2. The DSIPSS macro of any type other than CANCEL issued by code running on a task that is terminating (TVBTERM is on) will be rejected with a return code of 40 (I/O error).

3. FIRST, MIDDLE, LAST, and ONLY (supported in previous NetView releases) are converted to a form similar to title-line output, with limited formatting capabilities.

4. You can change applications using the SEG option to use the title-line processing to improve the appearance of the output.

5. You can convert applications using FIRST, MIDDLE, LAST, or ONLY options to use title-line, ASYPANEL, or the VIEW command for enhanced panel management.

6. If your data is formatted as an automation internal function request (AIFR), it must be in buffers located in subpool 0 which were obtained by DSIGET with Q=NO specified. DSIPSS frees the AIFR structure. If your data is not formatted as an AIFR, there is no restriction on the type of storage used, and the storage is not freed; you must free it.

7. If an AIFR (IFRCODAI) is used, see the bits for IFRAUPHI and IFRAUPLO in the DSIIFR structure description for a means to control the queueing priority of the command being sent from the NNT to OST.

8. If your data is formatted as an automation internal function request (AIFR), it must be in buffers located in subpool 0 which were obtained by DSIGET with Q=NO specified. DSIPSS frees the AIFR structure. If your data is not formatted as an AIFR, there is no restriction on the type of storage used, and the storage is not freed; you must free it.

---

**DSIPUSH: Establish Long-Running Command**

Macro DSIPUSH can perform the following related functions:

- **Establish named storage.**
  
  A storage pointer passed to DSIPUSH is associated with a 16-character name chosen by the DSIPUSH caller. After a successful DSIPUSH, you can use the DSIFIND macro to obtain the storage pointer from the name. All task types supporting commands (PPT, OST, NNT, and DST) also support named storage.

- **Establish a resumable command.**
  
  A resumable command can return to its caller with the assurance that the specified RESUME routine is called when the task’s asynchronous event handler has completed processing. For example, you would exit to allow queued messages to be processed. You would also exit after queuing a simulated terminal command. All NetView components behave in this manner. Task types supporting regular commands also support resumable commands (PPT, OST, and NNT).

- **Establish an event routine.**
  
  An event routine is a regular command which should be called whenever some ECB is posted. Specifying an ECB address on the DSIPUSH macro specifies that your SWBLRCRE routine is called promptly when the ECB is posted and only when the ECB is posted.
These functions are task-level operations, global for the task where they occur, but invisible from all other tasks. All requests define recovery and termination procedures. (See note 1 on page 234.)

DSIMVT addressability is required.

When a command issues DSIPUSH for a RESUME routine, the following apply:
- If a command invoked by a command procedure (NetView command list language, REXX, or high-level language) specifies MAJOR when issuing DSIPUSH, the command procedure is suspended at that point until the RESUME routine is removed by DSIPOP.
- If such a command specifies MINOR when invoking DSIPUSH, the RESUME routine is invoked following the completion of the command procedure.

A command can schedule a command procedure and obtain a return code by taking the following steps:
1. Invoke DSIPUSH for its own RESUME routine.
2. Make a direct call to schedule the command procedure.
3. Upon return from the direct call, exit to enable the command procedure to complete.
4. When the RESUME routine gains control, check the return code from the command procedure in CWBRCODE.

Do not use RESUME or ABEND reinstate routines under a DST.

Do not use DSIPUSH while running in a LOGOFF or abend reinstate routine specification a previous DSIPUSH, or while TVBINXIT is on.

The syntax for macro DSIPUSH is:

```
DSIPUSH
```

**Where:**

**CORR**

Defines the correlation option for the DSIPUSH request.

Options are:
CMD

The current command correlation environment is saved even if no RESUME routine was specified. The correlation environment can then be reactivated later using DSIFIND. (When DSIFIND CORR=CMD is used, the saved correlation environment becomes the current one.)

In applications where a RESUME routine was not used, this function enables the DSIPSS output of a command using DSIFIND CORR=CMD to be treated as if it was issued by the command that issued this DSIPUSH. The output will, in most cases be asynchronous when processed by a NetView pipeline, and a CORRWAIT stage should be coded in the pipeline.

NO

No correlation is done for the DSIPUSH request.

RESUME

DSIPUSH saves the current command correlation environment only if a RESUME routine is specified. RESUME is the default when the CORR keyword is omitted.

ECB

Is either a register containing the address of an ECB or the symbolic name of a fullword area containing the address of an ECB. Specifying ECB means that you want the command specified in SWBLRCRE to be called promptly when the ECB is posted and only when the ECB is posted.

When you specify ECB, the command specified by SWBLRCRE is called an event routine. The event routine must clear the ECB to prevent looping. If a storage pointer was specified with the DSIPUSH startup (SWBLRCST value), this value is passed to the event routine in register zero upon startup.

LIST

Is the register containing the address of a parameter list used by the service routine, or symbolic name of that list. Do not specify this as register 1; register 1 contains the SWB address within DSIPUSH. Do not put this list in the SWB that is to be passed to DSIPUSH. DSIPUSH reads, but does not write to, your parameter list; nevertheless, reentrant code demands that you put the parameter list in dynamic storage if your code updates any part of it (such as the storage pointer) at run time.

The parameter list contains the fields shown in Table 50.

<table>
<thead>
<tr>
<th>Hex Offset</th>
<th>Length</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>8</td>
<td>SWBLRCAB = ABEND reinstate routine</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>SWBLRCFG = Flags</td>
</tr>
<tr>
<td>28</td>
<td>8</td>
<td>SWBLRCLG = LOGOFF routine</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>SWBLRCLN = Length</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>SWBLRCNM = Name</td>
</tr>
<tr>
<td>18</td>
<td>8</td>
<td>SWBLRCRE = RESUME routine</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>SWBLRCST = Storage address</td>
</tr>
</tbody>
</table>

Where:

SWBLRCAB

Specifies the load module name of the ABEND reinstate routine. This name
DSIPUSH

must not be 0 for tasks other than DST tasks. This name must be 0 for DST
tasks. The load module named must have a corresponding CMDMDL
statement. If more than one CMDMDL statement is coded for a module
name, the first CMDMDL statement found is used. The ABEND routine
must not be an immediate command. See note 6 on page 235.

**SWBLRCFG**

Indicates whether the DSIPUSH execution is minor or major. Bit 0 is the
indicator bit; it is defined when a RESUME routine is specified. When bit
0=1, a minor DSIPUSH is performed. When bit 0=0, a major DSIPUSH is
performed. See "RESUME Routines" on page 71 for more information
about major and minor invocations. When no RESUME routine is specified,
this field is ignored.

**SWBLRCLG**

Specifies the load module name of the LOGOFF routine. This name must
not be 0. The load module named must have a corresponding CMDMDL
statement. If you code more than one CMDMDL statement for a module
name, the first CMDMDL statement found is used. The LOGOFF routine
must not be an immediate command. See note 6 on page 235.

**SWBLRCLN**

Specifies the parameter list length. Set SWBLRCLN equal to SWBLRCPU
(decimal 52).

**SWBLRCNM**

Associates a name with your long-running command or storage address.
Use this name on subsequent calls to DSIFIND or DSIPOP. The name
should be unique within a particular task, but a duplicate name used
under another task does not interfere. A second use of DSIPUSH for
named storage with the same name under the same task temporarily hides
the first storage pointer. The first storage pointer becomes accessible
through DSIFIND after DSIPOP is issued for the duplicate name.

The name can be any combination of bits, as long as you specify the name
identically for all macro calls with the same name. Names beginning with
DSI are reserved for NetView names. The name field is used as is; it is not
padded or justified.

**SWBLRCRE**

Specifies the load module name of the RESUME routine. If this field is 0,
no RESUME routine is indicated. The load module named must have a
 corresponding CMDMDL statement. If more than one CMDMDL statement
is coded for a module name, the first CMDMDL statement found is used.
The RESUME routine must not be an immediate command. See Note 6 or
page 235.

**SWBLRCST**

You can use this operand to associate a storage address with the specified
name. However, the NetView program makes no use of the value specified.
It is returned only when DSIFIND is invoked, specifying the same name.

**PROMOTE**

When you specify YES or NEWGROUP, a search is made of all currently
stacked long-running commands. If a long-running command with the same
name is found, the entire related group associated with that long-running
command is made the active group. Any parent procedure that has started the
long-running command processor is processed for cancellation.
The NetView program cancels a long-running command by resuming it with a −5 completion code in CWBRCODE. The decision to cancel is left to the long-running command processor. Cancellable long-running commands should issue a DSIPOP and return to the NetView program. NetView cancels NetView command lists and high-level language (HLL) procedures in this way when a −5 completion code is returned to them. Noncancellable procedures ignore the −5 return code and are not automatically canceled by the NetView program.

When you specify GROUP, the effect is the same except that the parent procedure is not canceled. You can use the GROUP option only to complete a previously running long-running command as shown in the following scenario:

1. A NetView command procedure invokes a user command (for example, COMMANDX). COMMANDX does the following:
   - Issues a DSIPUSH to push itself onto the task long-running command queue
   - Sends a request to another task using the DSIMQS macro
   - Returns to the NetView program
2. The request running under the second task completes and returns a response in the form of a command (for example, COMMANDY) to the originating task.
3. The originating task runs COMMANDY, which does a DSIPUSH PROMOTE=GROUP for COMMANDX. This makes COMMANDX and associated long-running commands the active group. COMMANDY then ends and returns to the NetView program.
4. COMMANDX resumes and completes.
5. Because PROMOTE=GROUP was specified and NEWGROUP or YES was not specified, the parent of COMMANDX (the original NetView command procedure) is given control when COMMANDX completes. Any other use of the GROUP option can produce unpredictable results, including the following:
   - Incorrect panels were presented to the operator.
   - Commands were lost.
   - Return codes to and from long-running commands were processed incorrectly.
   - Storage associated with a long-running command was mishandled.

If the group does not exist, the request is treated as an ordinary DSIPUSH and a stack element is created.

The storage specified in the invocation becomes the current storage associated with the specified name (SWBLRCNM) and previous storage associated with the name is returned to the caller in register 0.

When you specify PROMOTE=NO, or when the PROMOTE operand is omitted, no search is performed. The DSIPUSH is considered new and a long-running command (possibly a duplicate) is created. PROMOTE=NO is the default.

**SWB**

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB). The task information block (DSITIB) address in SWBTIB must be correctly set.
Return Codes in Register 15

The following return codes for macro DSIPUSH are found in register 15:

0  The function was successful; the long-running command request is queued.
4  Storage is not available for the request.
8  The ABEND reinstate or LOGOFF routine is required but was not specified.
12 The request was issued from an incorrect task:
   • RESUME request issued under DST
   • ABEND request issued under DST
   • DSIPUSH issued and task is not an OST, NNT, DST, or PPT
16 The request was issued while in an immediate command or while the NetView program is in an exit, or in the middle of a LOGOFF routine or ABEND reinstate routine.
20 The RESUME routine is a command list, or the CMDMDL statement did not pass validity checking, or the operator’s authority does not permit access to the RESUME routine.
   Verify that the first CMDMDL statement for this command in DSICMD is not of the immediate type.
24 The ABEND reinstate routine is a command list, or the CMDMDL statement did not pass validity checking, or the operator’s authority does not permit access to the RESUME routine.
   Verify that the first CMDMDL statement for this command in DSICMD is not of the immediate type.
28 The LOGOFF routine is a command list, or the CMDMDL statement did not pass validity checking, or the operator’s authority does not permit access to the RESUME routine.
   Verify that the first CMDMDL statement for this command in DSICMD is not of the immediate type.
32 The macro call is not valid. Fix assembly errors before trying to run the program.

Notes:
1. DSTs always end after any failure (abend). Thus recovery routines are not appropriate under DSTs.
2. When a command procedure is canceled for any reason, its return code is -5.
3. WAIT and PAUSE statements in a command procedure called by a long-running command do not cause premature calling of the RESUME routine. The RESUME routine is not scheduled until the command procedure completes.
4. HLL command processors cannot be pushed as ABEND, LOGOFF, or RESUME routines.
5. Do not code a RESUME routine to use DSIPUSH PROMOTE to move itself to the top of the long-running command stack. This will cause the NetView program to treat the DSIPUSH PROMOTE as a “no-operation” routine. A RESUME routine can, however, specify another routine on a DSIPUSH PROMOTE invocation.
6. Command authorization checking is not done on routines specified by SWBLRCAB, SWBLRCLG, or SWBLRCRE. If command authorization checking is required, the long-running command procedure issuing the DSIPUSH can be protected.

7. If register 15 contains return code 20, 24, or 28, register 0 contains a secondary return code. Under DSICES: Command Entry Services, see "Return Codes in Register 15" on page 168 for an explanation of the return code in register 0.

Note: For more information, refer to the RESET command in Tivoli NetView for z/OS Customization: Using PL/I and C and the NetView online help.

**DSIQOS: Query Operator Services**

Macro DSIQOS determines whether an operator is defined to the NetView program, and whether the operator is presently logged on.

The syntax for macro DSIQOS is:

```
DSIQOS
```

Where:

**OPID**

Is the register containing the address of an 8-byte, left-justified operator identification field, or symbolic name of that field.

**SWB**

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**Return Codes in Register 15**

The following return codes for macro DSIQOS are found in register 15:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The operator ID is defined to the NetView program and is presently logged on.</td>
</tr>
<tr>
<td>4</td>
<td>The operator ID is defined to the NetView program but is not presently logged on.</td>
</tr>
<tr>
<td>8</td>
<td>The operator ID is not defined to the NetView program but is presently logged on.</td>
</tr>
<tr>
<td>12</td>
<td>The operator ID is not defined to the NetView program.</td>
</tr>
</tbody>
</table>

**DSIQRS: Query Resource Services**

Macro DSIQRS determines whether a resource or a view is in any of the active spans that an operator can access. The return code from the macro indicates whether the specified operator ID is authorized to access the resource or view.

DSIMVT addressability is required.

The syntax for macro DSIQRS is:
Where:

ACCLVL
Specifies the access level being checked for an operator's access to a resource. If ACCLVL is not specified, the default of ALTER is used. The ACCLVL value can be one of the following:

ALTER
Multiwrite access. This is the default.

CONTROL
Multiread and single-write access.

READ
Information-only access. This is the access level of commands such as LIST or DISPLAY.

UPDATE
Change access. This is the access level of commands such as VARY, MODIFY, REPLY, or generic GMF actions such as activate and deactivate.

The resource specified must be in an active span that was started by the operator at the specified ACCLVL or higher. For example, to access a resource at UPDATE level, you must have a span active that contains that resource, at least at the UPDATE level.
Note: For more information about defining and starting spans at specific levels, refer to the Tivoli NetView for z/OS Security Reference.

OPID
A register containing the address of an 8-byte area, or a symbolic name of that area. The area contains the address of an 8-character NetView operator ID.

RESLENG
A register containing the address of a fullword area, or a symbolic name of that area. The area contains the length of the name represented by the RESOURCE keyword. If RESLENG is not specified, the default resource length is the minimum of 8 or the actual length of the resource name.

RESOURCE
A register containing the address of an area, or a symbolic name of that area. The area contains the address of the name of a resource to be matched against the contents of the operator’s active spans. If you do not specify RESLENG and the resource name is less than 8 characters in length, the resource name is padded with blanks.

RODMNAME
A register containing the address of an 8-byte area, or a symbolic name of that area. The area contains the address of the 8-character hexadecimal RODM name.

RODMOBID
A register containing the address of an 8-byte area, or a symbolic name of that area. The area contains the address of the 8-byte hexadecimal RODM object ID.

RODMREAS
A register containing the address of a fullword area, or a symbolic name of that area. The area contains the address of the 4-byte RODM reason code.

RODMRET
A register containing the address of a fullword area, or a symbolic name of that area. The area contains the address of the 4-byte RODM return code.

SWB
A register containing the address of a fullword area, or a symbolic name of that area. The area contains the address of a service work block (DSISWB).

VIEW
A register containing the address of an area, or a symbolic name of that area. The area contains the address of the name of a view to be matched against the contents of the operator’s active spans. VIEWLENG is required if VIEW is specified.

VIEWLENG
A register containing the address of a fullword area, or a symbolic name of that area. The area contains the length of the name represented by the VIEW keyword.

Return Codes in Register 15
The following return codes for macro DSIQRS are found in register 15:

0 The NetView operator has access to the resource at the specified level.

64 The NetView operator does not have access to the resource at the specified level.

68 No name was defined for this RODM object.
DSIQRS

101  RODM query failure.
104  The length of the resource name or view name is not valid. For example, the length is zero (0).
128  The operator specified in OPID is not logged on to NetView.
160  No spans are defined to NetView.
200  Internal storage failure.
204  Request is not valid. For example, a query is made on a view name while SECOPTS.SPANAUTH=*NONE* is used.

MNOTES Issued

- MNOTE 8, REQUIRED KEYWORD SWB MISSING
- MNOTE 8, REQUIRED KEYWORD OPID MISSING
- MNOTE 8, INVALID VALUE FOR ACCLVL. VALUES ARE READ, UPDATE, CONTROL OR ALTER.
- MNOTE 8, KEYWORD MISSING - ONE OF THE FOLLOWING REQUIRED: RESOURCE, RODMID OR VIEW
- MNOTE 8, BOTH RESOURCE AND RODMID CODED, ONLY ONE IS VALID
- MNOTE 8, RODMNAME NOT VALID WITH RESOURCE
- MNOTE 8, BOTH RESOURCE AND VIEW CODED, ONLY ONE IS ALLOWED
- MNOTE 8, VIEWLENG NOT VALID WITH RESOURCE
- MNOTE 8, RODMRET NOT VALID WITH RESOURCE
- MNOTE 8, RODMREAS NOT VALID WITH RESOURCE
- MNOTE 8, BOTH RODMID AND VIEW CODED, ONLY ONE IS ALLOWED
- MNOTE 8, VIEWLENG NOT VALID WITH RODMID
- MNOTE 8, RESLENG NOT VALID WITH RODMID
- MNOTE 8, RODMREAS MUST BE SPECIFIED WITH RODMRET
- MNOTE 8, RODMRET MUST BE SPECIFIED WITH RODMREAS
- MNOTE 8, VIEWLENG MUST BE SPECIFIED WITH VIEW
- MNOTE 8, RESLENG NOT VALID WITH VIEW
- MNOTE 8, RODMNAME NOT VALID WITH VIEW
- MNOTE 8, RODMRET NOT VALID WITH VIEW
- MNOTE 8, RODMREAS NOT VALID WITH VIEW

DSIRDS: Resource Definition Services

The DSIRDS macro is no longer supported. This macro still exists for migration. For span authority checking, use the DSIQRS macro. See "DSIQRS: Query Resource Services" on page 233.

If you have any programs that call DSIRDS, you need to rewrite them. If you reassemble any code that calls the version of DSIRDS shipped with this release of NetView, the assembly will fail with a return code of 8. This gives you a pointer to the places in your programs that need to be rewritten.

If you do not reassemble your own applications, any application that calls DSIRDS will fail with a return code of 28.

DSIRXEBS: Get an EVALBLOK

Macro DSIRXEBS is an interface for getting an evaluation block (EVALBLOK).
DSIRXEBS

MVS/ESA users can use the NetView macro DSIRXEBS or the TSO/E macro IRXRLT.

The size of the EVALBLOK data area is passed. The header size is added to the data size, and the area is initialized. Parameters are passed in the system work block (DSISWB) that you provide. If you pass an EVALBLOK address in the EVBPTR, that block is freed before the new block is obtained. For additional information about DSIRXEBS, see "Chapter 6 Writing User Function Directories" on page 101.

DSIMVT addressability is required.

The syntax for macro DSIRXEBS is:

```
DSIRXEBS
   label
   DSIRXEBS -SWB= (register), LENGTH= (register)
   , ENVBPTR= (register)
   , EVBPTR= (register)
```

Where:

ENVBPTR
Is the register, or symbolic name of a fullword area, containing the address of the TSO/E environment block address.

EVBPTR
Is the register, or symbolic name of a fullword area, containing the address where the EVALBLOK address is placed when it is obtained. If this area is not 0 (X'00') initially, it is assumed that it contains an EVALBLOK address to be freed.

LENGTH
Is the register, or symbolic name of a fullword area, containing the address where the EVALBLOK data size is obtained. The data size should be the number of bytes of data that will be returned by the function.

SWB
Is the register, or symbolic name of a fullword area, containing the address of an SWB.

Return Codes in Register 15

The following return codes for macro DSIRXEBS are found in register 15:

- **0**: The function was successful; the address of the EVALBLOK was returned in EVBPTR. If an EVALBLOK address was passed, the block was freed.
- **8**: Storage was insufficient to obtain the EVALBLOK. If an address of an EVALBLOK was passed, the storage was freed.
- **12**: A request was made and found not valid. One of the required operands was not correctly specified.
- **20**: Processing was not successful. A new evaluation block was not allocated.
Processing was not successful. A valid language processor environment could not be located for the current task.

**DSISRCMV: Search for Subvector or Subfield**

Macro DSISRCMV establishes register conventions for a call to subroutine SR$SUB in sample CNMS4284. SR$SUB searches a major vector for a subvector, or a subvector for a subfield. SR$SUB returns the address of the subvector or subfield if it is found.

The syntax for macro DSISRCMV is:

```
DSISRCMV
```

```
label DSISRCMV
MVADDR= (register name)
SVADDR= (register name)
PRSVAD= (register name)
PRSFAD= (register name)
SVOCCUR= name
SFID= name
SEQ= Y
```

**SearchChoice**

```
, SVID= name
, SEQ= Y
```

**PRSVAD:**

```
, PRSVAD= (register name)
```

**SVOCCUR:**

```
, SVOCCUR= name
```

**PRSFAD:**

```
, PRSFAD= (register name)
```

**SFOCCUR:**
Where:

**MVADDR**
Is the register, or symbolic name of a fullword area, containing the address of the major vector.

**PRSFAD**
Is the register, or symbolic name of a fullword area, containing the address of the last subfield found. This is an optional operand. If you code PRSFAD, the search starts from this address rather than the address of the subvector (SVADDR).

**PRSVAD**
Is the register, or symbolic name of a fullword area, containing the address of the last subvector found. This is an optional operand. If you code PRSVAD, the search starts from this address rather than the address of the major vector (MVADDR).

**SEQ**
Specifies the search type to be performed.

**Y**
Specifies to search sequentially for the next subfield or subvector. If you specify both MVADDR and PRSVAD, the sequential search starts at the address specified by PRSVAD. If you specify both SVADDR and PRSFAD, the sequential search starts at the address specified by PRSFAD.

**SFID**
Is the symbolic name of a 1-byte hexadecimal field containing the subfield type for which to search.

**SFOCCUR**
Is the symbolic name of a 2-byte (halfword) hexadecimal field containing the subfield occurrence for which to search.

**SVADDR**
Is the register, or symbolic name of a fullword area, containing the address of the subvector.

**SVID**
Is the symbolic name of a 1-byte hexadecimal field containing the subvector type for which to search.

**SVOCCUR**
Is the symbolic name of a 2-byte (halfword) hexadecimal field containing the subvector occurrence for which to search.

**Note:** Issue the DSISRCMV macro only from the CNMS4284 sample.

---

**DSISYS: Operating System Indicator**

Macro DSISYS is used for testing the current operating system by setting global variables, as shown in Table 51 on page 242. This macro is used in programs that are to be run on multiple operating systems, to vary compilation according to the current system. This macro enables system-dependent code to be placed in common programs.
The syntax for macro DSISYS is:

```
DSISYS
```

&DSISYSE

Is a global variable that must be declared. &DSISYST is set at compilation time by the DSISYS macro to determine if the current operating system is an ESA-type operating system. Use the AIF assembler statement to test its value. For example, if you want to distinguish between MVS/XA and MVS/ESA, check the DSISYSE. Possible values are:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>&amp;DSISYST</th>
<th>&amp;DSISYSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVS/370 (V1R2 only)</td>
<td>VS2/MVS</td>
<td>(not defined)</td>
</tr>
<tr>
<td>MVS/XA</td>
<td>MVS/XA</td>
<td>(not defined)</td>
</tr>
<tr>
<td>MVS/ESA</td>
<td>MVS/XA</td>
<td>MVS/ESA</td>
</tr>
</tbody>
</table>

&DSISYST

Is a global variable that must be declared. &DSISYST is set at compilation time by the DSISYS macro to reflect the current operating system. Use the AIF assembler statement to test its value.

---

**DSITECBS: Manage a Dynamic ECB List for DSTs**

Macro DSITECBS adds or deletes an ECB to or from a task’s ECB list.

DSITECBS enables a DST to dynamically manage the ECB and specify the command processor that is invoked when the ECB is posted.

MVT addressability is required.

The syntax for macro DSITECBS is:

```
DSITECBS
```

Where:
ECB
Is the register, or symbolic name of the fullword area, containing the address of the area that contains the ECB address.

ECBPRC
Is the register, or symbolic name of the fullword area, containing the address of the area that contains the name of the ECB command processor. This parameter is required if you specify FUNC = ADD.

ENV
Is the register, or symbolic name of the fullword area, containing the address of the environment area you want to accompany the ECB address and the ECB processor. The environment address is returned when the ECB is posted.

FUNC
Specifies the function you want to perform. There is no default. Select one of the following:

ADD
Adds an ECB to the task’s ECB list.

DELETE
Deletes an existing ECB from the list.

MAXCALL
Is the register, or symbolic name of the fullword area, containing the address of the area that specifies a binary value for the maximum number of times the ECB processor is invoked for the indicated ECB before other ECBs can be serviced.

SWB
Is the register, or symbolic name of the fullword area, containing the address of a service work block.

Return Codes in Register 15
The following return codes for macro DSITECBS are found in register 15:

0  The function was successful.
4  The load module name was not found in DSICMD.
8  The ECB address is already in the list.
12  There is insufficient storage to push the ECB.
16  A SWB cannot be obtained for internal work.
20  The ECB to be deleted was not found in the ECB list.
24  This task type is not valid; use a DST.

DSIVARS: Variable Services
Macro DSIVARS accesses, sets, and references NetView task global and common global variables. Task global variables enable communication among command lists, command processors, and installation exits executing on the same NetView task. Common global variables enable communication among command lists, command processors, and installation exits executing on all NetView tasks.

You can also use DSIVARS to set and reference local variables of the program that called the assembler program that uses DSIVARS, if the assembler program was
DSIVARS

called from a command procedure written in the NetView command list language, REXX, PL/I, or C; or from a PL/I or C installation exit.

MVT addressability is required.

The syntax for macro DSIVARS is:

DSIVARS

Where:

NUMVARS
Specifies the register, or symbolic name of a fullword area, containing the number of variables to be processed. The parameter NUMVARS is optional.

If you do not specify a value for NUMVARS, the default is 1. Do not use a value of 0.

REQUEST
Specifies which NetView variable operation to perform. The parameter REQUEST is required.

GET
Retrieves a copy of the current value of the variable and returns it into the area specified by VALUE, up to the length specified by VALLEN. Upon return, VALRETL is set to the actual length of the returned value.

If the variable value is longer than the length specified by VALLEN, the value is truncated and return code 4 is issued. If the variable has not been previously set, VALRETL is set to zero.

PUT
Sets the value specified by VALUE and VALLEN of the variable name specified by VARNAMES and VARLEN into the variable pool specified by VARTYPE.

SWB
Specifies the register, or symbolic name of a fullword area, containing the
address of a service work block (SWB). Set field SWBTIB to indicate your task information block (TIB). The parameter SWB is required.

**VALLEN**
For GET requests, VALLEN specifies the address of an area containing a list of fullword values that contain the maximum length of values to be returned into the matching VALUE area or the symbolic name of the list of fullword values. If the length of the actual variable value is greater than the length specified by VALLEN, the returned value is truncated and return code 4 is issued.

The actual length of the variable value is set in the VALRETL area. This storage is in autodata or another addressable storage area because the DSIVARS service will write a value to it.

For PUT requests, VALLEN specifies the address of an area containing a list of fullword values that contain the length of the matching VALUE area that is assigned to the variable or the symbolic name of the list of fullword values. The maximum value length is 255. If you specify a value length of 0, the variable’s value is set to null.

**VALRETL**
For GET requests, VALRETL specifies the address of an area containing a list of fullword fields or the symbolic name of the list of fullword fields. The DSIVARS service sets the fullword fields to the actual length of the value returned for each variable.

If a variable value is truncated, the VALRETL field for that variable contains the actual length of the variable value, and the VALLEN field for that variable contains the truncated length of the variable value.

The parameter VALRETL is required for GET requests and ignored for PUT requests.

**VALUE**
For GET requests, VALUE specifies the address of an area containing a list of fullword addresses of storage locations to which the variable values are returned as a result of the DSIVARS operation or the symbolic name of the list of fullword addresses. This storage is in autodata or another addressable storage area because the DSIVARS service will write a value to it. The parameter VALUE is required.

For PUT requests, VALUE specifies the address of an area containing a list of fullword addresses of storage locations that contain the variable values to be associated with the variable names specified by VARNAME or the symbolic name of the list of fullword addresses.

**VARLEN**
Specifies the address of an area containing a list of fullword variable name lengths for the DSIVARS macro or the symbolic name of the list of fullword variable name lengths. The parameter VARLEN is required.

A variable name can be from 1 to 31 characters.

**VARNAME**
Specifies the address of an area containing a list of fullword addresses of storage locations that contain the variable names for the DSIVARS macro or the symbolic name of the list of fullword addresses. The parameter VARNAME is required.

Valid characters for variable names are A–Z, 0–9, @, #, $, ¢, ., !, ?, and _. Do not use a number or a period as the first character in a variable name.
VARTYPE
Specifies the type of NetView variable to be referenced. The parameter
VARTYPE is required.

CALLER
Specifies the local variable pool of the calling command procedure or
installation exit that should be accessed. The command procedure can be
written in REXX, NetView command list language, PL/I, or C.

You can use VARTYPE=CALLER only if the assembler program invoking
the DSIVARS macro was called from a command procedure or installation
exit written in PL/I or C. Specifying a VARTYPE of CALLER from an
assembler installation exit is not supported. Using VARTYPE=CALLER you
can set a variable to be referenced by the command procedure upon return
from an assembler command processor.

CGLOBAL
Specifies that the NetView common global variable pool can be accessed.

TGLOBAL
Specifies that the NetView task global variable pool can be accessed.

Return Codes in Register 15
The following are return codes for macro DSIVARS:

0  Operation specified was successfully completed.
4  For a GET request, one or more variable values returned were truncated.
The actual value length was greater than that specified by VALLEN for
those variables.
8  For VARTYPE=CALLER, this macro was issued from a command processor
that was not called from a command procedure or high-level language
installation exit. VARTYPE=CALLER can only be used to access the calling
command procedure or the HLL program variable pool.
12  The number of variables specified is not valid. The value specified for
NUMVARS must be greater than 0.
16  Variable name is not valid. One or more variable names specified
contained characters that are not valid or had a length greater than 31
characters.
20  Value length not valid. One or more variable lengths specified are too long
(for a PUT request), or are less than 0 (for a GET or PUT request). You
must specify a variable value length no larger than 255 for PUT requests,
and 0 or greater for all requests.
24  Environment not valid. Global variables cannot be accessed while running
in asynchronous exit mode. NetView determines asynchronous exit mode
by examining the TVBINXIT flag.
28  Storage failure.

Notes:
1. In the NetView command list language, variable names are limited to 11
characters, and more restrictions apply to the characters used in the variable. If
the variable name you specify is referenced in the NetView command list
language, the name can be no longer than 11 characters, and can only contain
characters that are acceptable in that language.
2. Variables are indicated in the NetView command list language by an ampersand (&) at the front of the name. Do not use an ampersand to specify a VARNAME.

3. For NetView common global variable requests using DSIVARS, the NetView common global variable pool is locked for the duration of the DSIVARS processing from other NetView tasks accessing it. This ensures that if multiple variables are specified on one DSIVARS macro invocation, the common global values will be constant at that time.

4. Specifying a VARTYPE of CALLER from an assembler installation exit is not supported and may cause unpredictable results. If there is an active command procedure on the task executing the installation exit, a variable can be set in the executing command procedure. The return code in this case is 0.

Note: For more information about using NetView variables in NetView command list language and REXX command lists, and about using NetView variables in PL/I and C, refer to Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language and Tivoli NetView for z/OS Customization: Using PL/I and C.

DSIWAT: ECB Wait Services

Macro DSIWAT causes a subtask to wait for completion of an event.

The syntax for macro DSIWAT is:

**DSIWAT**

Where:

**ECB**

Is the register (2–12) containing the address of an aligned fullword to be used as an event control block (ECB), or symbolic name of that aligned fullword.

**ECBLIST**

Is the register containing the address of a contiguous list of fullword addresses of ECBs, or symbolic name of that list. The last entry in the list of ECB addresses has the high-order bit 0 set to 1 to indicate the end of the list.

The following example shows how you can code DSIWAT:

```
DSIWAT ECBLIST=LISTAREA
ECB1 DC F'0'
ECB2 DC F'0'
LISTAREA DC A(ECB1)
DC A(ECB2)
DC A(ECB3)
DC A(ECB4+X'80000000')
ECB3 DC F'0'
ECB4 DC F'0'
```
**DSIWAT**

Execution resumes when any one of the ECBs is posted, and you can use the DSIPOS macro to post an ECB.

**Notes:**
1. If your code is going to run on any NetView task other than a user-optional task:
   - DSIWAT should not be invoked to wait on any event that may take a long time to complete.
   - Your code should not repeatedly issue DSIWAT for your own ECB, process the ECB, and reissue the DSIWAT.
2. All NetView tasks must have adequate time to process their own internal work, primarily represented by an internal ECB list. User code that may run for an extended period of time or be re-driven repeatedly must periodically “return” to NetView to enable the task to do this internal work.

This can be done by writing your code as a long-running command processor. (See [Chapter 4. Writing Command Processors](#) on page 61.) Alternatively, you can invoke DSIPSS with the TYPE=PSSWAIT attribute, which will wait on your ECBs and the NetView task ECBs and then return to NetView when a return code of 56 is received. (See [DSIPSS: Presentation Services](#) on page 222.)

---

**DSIWCS: Write Console Services**

Macro DSIWCS writes a message to the system operator’s console. However, you should not use this macro to produce a double-byte character string (DBCS) message. DBCS messages are not supported by the operating systems.

The message is truncated at 120 characters. The message buffer must have an initialized buffer header.

This macro issues the DSIWLS macro to log the write to the console.

MVT addressability is required.

The syntax for macro DSIWCS is:

```
DSIWCS
```

Where:

**BFR**

Is the register, or symbolic name of a fullword area, containing the address of a buffer with the message. This buffer must have an initialized BUHDR.

**SWB**

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**Return Code in Register 15**

The following is the return code for macro DSIWCS:

- 0  The function was successful; the message was sent to the system console.
**DSIWLS: Write Log Services**

Macro DSIWLS writes a record to the network log, the hardcopy log, the MVS system log, an external log, or a NetView sequential log, depending on the use of the DEFAULTS command, the OVERRIDE command, DSIEX02A, or DSIEX04, as follows:

- Without the EXTLOG parameter, DSIWLS can send records to the network log, the MVS system log, the operator’s hardcopy device or, with the SAMREC parameter, to a NetView sequential log. For the network log, the record can be truncated, depending on the user-defined VSAM record size. For the MVS system log, system truncation rules apply.
- With the EXTLOG parameter, DSIWLS sends records to an external task that can log the data.

MVT addressability is required.

The syntax for macro DSIWLS is:

```
DSIWLS
```

**Options:**

```
(label) DSIWLS SWB= Options

BFR= Options ,HCT= Options ,EXTLOG= 'xxx'

SAMREC= Options ,SAMLEN= Options ,SAMTASK= Options
```

**Where:**

BFR

Is the register, or symbolic name of a fullword area, containing the address of a user-provided input buffer. This buffer contains the record that is to be logged. This buffer must have an initialized BUFHDR.

EXTLOG

Indicates that the buffer is to be logged externally under the DSIELTSK subtask.

You can accomplish external logging in one of the following ways:

- Write to a system management facility (SMF) data set. This option is restricted to MVS systems. The data to be logged is a standard SMF record, with an SMF record type greater than or equal to 128. The DST XITXL installation exit is called before writing the record to SMF.
Write to a data set other than SMF. This option is not restricted to any particular operating system. Code and install the DST XITXL installation exit. This installation exit then performs the actual logging.

**register**
- Is the register that contains the address of a 3-byte area that represents the last 3 letters in the name of an external logging command processor.

**symbolic name**
- Is the symbolic name of a 3-byte area that represents the last 3 letters in the name of an external logging command processor.

**xxx**
- Are 3 characters that become the last 3 characters of the command name used to select the command processor that logs the data. The first characters of the verb must begin with DSIEL. For example, if EXTLOG='ABC', the CMDMDL statement in DSICMD is:
  ```
  DSIELABC CMDMDL MOD=DSIELSMF,TYPE=D
  ```

**HCT**
- Is the register, or symbolic name of a fullword area, containing the hardcopy task’s TVB address.

**SAMLEN**
- Is the register, or symbolic name of a fullword area, containing the length of the record to be logged. If the value of SAMLEN is greater than 32000, the record to be logged is truncated and the macro returns a nonzero return code.

  The BLKSIZE for a sequential logging task must be at least as large as SAMLEN plus 42, or the record to be logged is truncated. SAMLEN is required with SAMREC.

**SAMREC**
- Is a keyword that either points to or names the record that should be written to the NetView sequential log that is controlled by the task indicated by the SAMTASK keyword. Unlike the BFR keyword, this operand only points to the data that is to be logged. NetView services put the record into the correct format for scheduling the record to the sequential log task.

**SAMTASK**
- Is the register, or symbolic name of a 2-fullword area, containing the name of the NetView task that has been defined to do sequential logging. This task can be verified for sequential logging capability. SAMTASK is required with SAMREC.

**SWB**
- Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

### Return Codes in Register 15

If you do not specify EXTLOG or SAMREC, the following return codes for macro DSIWLS are found in register 15:

- **0**: The function was successful; the record has been sent to the network log and to the hardcopy log.
- **4**: No storage is available for logging.
- **8**: Successful log to the hardcopy log; network log is not active.
- **12**: Successful log to the network log; hardcopy log is not active for this task.
The hardcopy log for this task and this network log are both inactive.

If you specify SAMREC, the following are the return codes:

0  The function was successful; the record has been queued to the sequential log task.
4  No storage is available.
16 The task is not active, no data set is available, or the specified task is not a sequential log task.
32 DSIWLS invoked with a length that is not valid or the buffer pointer is zero.

If you specify EXTLOG, the following are the return codes:

0  The function was successful. A copy of the caller's buffer is queued to the external logging task (DSIELTSK).
4  No storage is available for copying the user input buffer for logging.
24 No external logging command processor was found.
28 DSIMQS failed attempting to send the log record to the external logging task.

Notes:
1. You must specify either BFR or SAMREC, but not both.
2. You can specify either EXTLOG or HCT with BFR, but not both.

DSIZCSMS: CNM Data Services

Macro DSIZCSMS is a data services macro. It requests and sends CNM data over the communication network management interface (CNMI).

DSIZCSMS embeds the caller's network services request or response unit (RU) in a forward RU that is passed to the SSCP over the access method's CNMI. The SSCP then sends the embedded RU to the specified destination.

MVT addressability is not required.

The syntax for macro DSIZCSMS is:

\[
\text{DSIZCSMS} \label{label} \text{DSIZCSMS SWB=} (\text{register}) \text{DSRB=} (\text{register}) \symbolic_name \text{DSIZCSMS INPUT RTYPE=REPLY OPTION TYPE ,RTYPE=REPLY SECONDS RESPONSE) x0-VVVV}
\]

DSIWLS
DSIZCSMS

OPTION

\[ \text{OPTION} = \begin{cases} \text{NOCNM} \\ \text{NOPRID} \\ \text{SALT} \end{cases} \]

TYPE

\[ \text{TYPE} = \begin{cases} \text{CHAIN} \\ \text{RU} \\ \text{NOEMBED} \end{cases} \]

SECONDS

\[ \text{SECONDS} = \begin{cases} \text{(register)} \\ \text{symbolic\_name} \end{cases} \]

INPUT:

\[ \begin{align*} \text{INPUT} & \quad , \text{LENGTH=} (\text{register}) & \text{,RU= (register)} \\ & \quad \text{,RULENG=} (\text{register}) & \text{,DEST= (register)} \\ & \quad \text{,TARGET= (register)} \end{align*} \]

INPUT

\[ \text{INPUT} = (\text{register}) \]

Note:

See the description for TYPE for information about conditions that apply to CHAIN, NOEMBED, and RU.

For information about conditions that apply to OPTION, SALT, REPLY, SECONDS, and DEST, see their respective descriptions.

Where:

DEST

Is the register, or symbolic name of a fullword user area, containing the
address of the network destination to which the embedded RU is sent. This network destination is 8 characters long, left-justified, and padded with blanks if necessary.

**DSRB**
Is the register, or symbolic name of a fullword area, containing the address of a data services request block (DSIDSRB) to be passed to the CNMI service routine DSIZCSMM.

**INPUT**
Is the register, or symbolic name of a fullword storage location, containing the address of a user input buffer. The input buffer must be large enough to contain a 24-byte buffer header plus the length of the RU to be sent (if a forward RU is to be built, add 28 bytes to the size of the input buffer). This buffer contains a buffer header followed by text. It also holds the deliver RU that is returned by the access method. To enable command processors or installation exit routines that operate in 24-bit addressing mode to access the buffer, it resides below 16 MB. To receive a reply over the CNMI, the buffer accommodates at least a 32-byte reply (a 24-byte NetView buffer header and an 8-byte positive or negative response).

**LENGTH**
Is the register, or symbolic name of a fullword storage location, containing the length in binary of the input buffer.

**OPTION**
Enables nonstandard forward RUs to be sent by CNMI services.

- **NOCNM**
  Indicates that a forward RU is sent that does not contain a CNM header, or procedure-related ID (PRID). An example is a forward RU containing NS IPL command types of INIT, TEXT, and FINAL.

- **NOPRID**
  Indicates that a forward RU is sent that does not contain a procedure-related ID (PRID) in the CNM header. An example is a REQMS that does not require a reply RECFMS because of user protocols.

- **SALT**
  Indicates that a forward RU is sent that is associated with a target. This flag alerts VTAM that a target to the SNA address list translation (SALT) is required for an NMVT to be transported using a forward RU.

**RU**
Is the register, or symbolic name of a fullword storage location, containing the address of a user area. The area is an RU that is to be embedded within the forward RU.

**RULENG**
Is the register, or symbolic name of a fullword user area, containing the length in binary of the embedded RU buffer. The RULENG cannot exceed 32743 decimal bytes.

**RTYPE**
Describes the response required for the request.

- **REPLY**
  Indicates that a REPLY RU is required for completion of the request. This is the default.

- **RESPONSE**
  Indicates that a positive response is sufficient to complete the request.
SECONDS
Specifies the number of seconds to wait before canceling the outstanding request. The number must have a positive value no greater than 86400. If you do not specify the SECONDS operand or the value is 0, no time-out function is performed (an indefinite wait is possible). If you specify SECONDS, you cannot specify any of the following:
• TYPE=RU
• TYPE=CHAIN
• OPTION=NOCNM
• OPTION=NOPRID

If you specify SECONDS and RTYPE=REPLY, specify DEST.

SWB
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB) to be passed to the CNMI service routine DSIZCSMM.

TARGET
Is the register, or symbolic name of a fullword user area, containing the address of the network component that is the object of the embedded RU, or symbolic name of that network component. This network component is 8 characters long, left-justified, and padded with blanks if necessary.

TYPE
Controls the processing for the RU:

CHAIN
Indicates that the DSRB has received data and remains in use to accept further RUs associated with the specific request. If you specify TYPE=CHAIN, the SWB and DSRB operands are required; all other operands are not valid. This operand is not valid with an unsolicited DSRB.

NOEMBED
Indicates that a forward RU is not to be built for this request.

RU
Indicates that the input RU is not to be embedded in a forward RU.

Return Codes in Register 15
The following major return codes for macro DSIZCSMS are found in register 15:

0  The function was successful; data was sent to VTAM.
4  The requested function could not be performed.
8  The input buffer was too small to build a forward RU.
12 An error was found in a parameter specification.
16 The program did not execute under a data services task.
20 The RULENG exceeded the maximum RU length required.

Minor Return Codes in Register 0
The following minor return codes for macro DSIZCSMS are found in register zero (0):

0  The function was successful.
4  The SWB was not valid.
The DSRB was not valid.
The DSRB that was passed was in use.
An unsolicited DSRB was passed.
An operator ID specified in the DSRB was not valid.
Reserved.
There was insufficient storage to process the request.
The CNMI is inactive.
The request was rejected by the access method.
An installation exit rejected the request.
Data truncation occurred during the installation exit processing.
The specified SECONDS value was not valid.

**CNM Data Services Module Examples**

The following examples show how you can specify the DSIZCSMS macro.

```plaintext
DSIZCSMS SWB=SWBADDR,DSRB=DSRBADDR,INPUT=RAQDDR,LENGTH=FORWDLEN,RU=READYRU,RULENG=READLEN,DEST=DESTNAME,RTYPE=REPLY,SECONDS=TIMEOUT
```

*Figure 25. Example 1*

In Example 1, the data is sent across the CNMI to DESTNAME. READYRU contains the request RU that is to be sent. The reply information is returned in the RQADDR buffer. After TIMEOUT seconds, the request is canceled. If an associated reply is received later, it is discarded.

```plaintext
DSIZCSMS SWB=SWBADDR,DSRB=DSRBADDR,INPUT=RAQDDR,LENGTH=FORWDLEN,RU=READYRU,RULENG=READLEN,DEST=DESTNAME,RTYPE=REPLY.
```

*Figure 26. Example 2*

In Example 2, the data is sent across the CNMI to DESTNAME. READYRU contains the request RU that is to be sent. The reply information is returned in the RQADDR buffer. The request remains outstanding until a reply is received.

**Notes:**

1. You can specify either OPTION or TYPE keywords, but not both.
2. When RTYPE=REPLY, you cannot specify OPTION=SALT; you can specify only one of the following:
   - OPTION=NOCNM
   - OPTION=NOPRID
   - The keyword SECONDS
3. You must specify TARGET if OPTION=SALT.
4. You can specify only one of the following:
   - OPTION=NOCNM
   - OPTION=NOPRID
   - The keyword SECONDS
5. If TYPE=CHAIN, SWB and DSRB are required.
6. If TYPE does not equal CHAIN, SWB, DSRB, INPUT, RU, and DEST are optional.

7. You can specify either TYPE=CHAIN or the keyword SECONDS, but not both.

8. If you specify TYPE=NOEMBED, you cannot enter TARGET or DEST.

9. If you specify TYPE=RU, you cannot enter SECONDS, TARGET, DEST, or RTYPE=RESPONSE.

10. For major return codes of 8 or 20, register 0 contains the RULENG.

Note: For more information about SNA RUs, refer to the SNA library.

**DSIZVSMS: VSAM Data Services**

Macro DSIZVSMS is a data services macro. It requests VSAM services for a data services command processor (DSCP).

DSIZVSMS provides access to VSAM services that perform input/output to the specified problem determination file or data set. The operands provide access for data recording, data retrieval, and data deletion.

MVT addressability is not required.

The syntax for macro DSIZVSMS is:

```
DSIZVSMS
```

```plaintext
DSIZVSMS

,label

DSIZVSMS SWB= (register)

DSRB= (register)

,symbolic_name

,symbolic_name

,FUNC= GET

,PUT

,POINT

,ENDREQ

,ERASE

,KEY= (register)

,KEYLEN= (register)

,symbolic_name

,symbolic_name

,OPTION= (Options)

,DATAAREA= (register)

,symbolic_name

,KEYLEN= 1

,Options:

,SEQ

,DIR

,SKP

,ARD

,LRD

,FWD

,NSP

,BWD

,UPD

,KEQ

,KGE

,FKS

,GEN
```

```
```
Where:

**DATAREA**
Is the register containing the address of a user work buffer, or symbolic name of that buffer. The buffer must be large enough to contain the maximum size record in the file or data set. The buffer is used by VSAM in the processing of records. This buffer contains an initialized BUFHDR, followed by text.

**DSRB**
Is the register, or symbolic name of a fullword, containing the address of a data services request block (DSIDSRB). The DSRB contains request information such as request parameter list (RPL), ACB, ECB, and fields used by the data services task VSAM service routine for VSAM I/O.

**FUNC**
Describes the VSAM request macro to be issued.

**KEY**
Is the register containing the address of the VSAM key to be used for access to the requested data, or symbolic name of a fullword that contains the key. Because DSIZVSMS will restore the key to its original location, the key must also be in NetView-accessible storage.

**KEYLEN**
Is the register, or symbolic name of a fullword, containing the length in bytes of the key pointed to by KEY. The default key length is 1.

**OPTION**
Specifies the type of access to the file through requests defined by the NetView request parameter list (RPL). Options are arranged in groups. The first time the RPL is set up, you must specify one, and only one, option from each group. Subsequently, you can specify one option from one or more groups. Separate multiple options with commas.

This OPTION operand has no defaults. This operand is not valid when you specify FUNC=ERASE or FUNC=ENDREQ.

**SWB**
Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB). The SWB contains a save area, work area, and task information block (DSITIB) address data. The caller must initialize the SWBTIB field in the SWB with a valid TIB address.

**Return Codes in Register 15**
The following major return codes for macro DSIZVSMS are found in register 15:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion of VSAM function.</td>
</tr>
<tr>
<td>4</td>
<td>Manipulative macro error occurred during processing.</td>
</tr>
<tr>
<td>8</td>
<td>An error occurred in the EXECUTE form of a manipulative macro. An operand was not in the list.</td>
</tr>
<tr>
<td>12</td>
<td>Unsuccessful completion.</td>
</tr>
<tr>
<td>16</td>
<td>DSIZVSMS was issued while not executing under a DST.</td>
</tr>
</tbody>
</table>

**Minor Return Codes in Register 0**
The following minor return codes for macro DSIZVSMS are found in register zero (0):

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
</tbody>
</table>
The specified DSRB was not valid or in use.

An ACB was unavailable or was not open. This may be due to a `SWITCH taskname`, `T` command having been issued.

Resume verb processing error.

An installation exit rejected the request.

The VSAM I/O request was not valid or there was an I/O scheduling error.

Data truncation occurred during substitution of data in an installation exit; or control block storage could not be obtained.

An installation exit returned a return code that was not valid.

Note: For more information about specifying FUNC and OPTION, refer to the OS/VS VSAM library. For an explanation of RPL feedback codes, refer to the OS/VS VSAM library and the MVS/ESA library.

**DSI6REGS: LU 6.2 Registration**

Macro DSI6REGS registers a management services (MS) application name with the MS transport, or a served application name with operations management.

The registration includes the command name of the command processor to be invoked when unsolicited or asynchronous solicited data is routed to an MS application or served application. The NetView task where the service routine is executing is registered as the task where the command is to run when unsolicited data is received. The invoked service program keeps track of the registered applications and served applications internally to provide routing.

Before accepting the registration request, the NetView program verifies that the task has the authority to issue the specified command to be driven with data received.

An MS application or operations management served application can also indicate on the registration request that it wishes to be informed of any focal point information for a specified focal point category. In addition, an MS application can specify whether it is a focal point application.

The registration request can specify whether it replaces a current registration for the MS application or served application. When the request replaces prior registration, subsequent registration requests can change the task or command processor that is to receive unsolicited data or the focal point category of interest. Concurrent requests from multiple tasks are sequenced at points of potential conflict. An application can register as both an MS application and an operations management served application. A registration of one type does not affect the registration of the other type.

Deregistration terminates the NetView MS transport’s awareness of the MS application or operations management’s awareness of the served application. No further data can be sent or received by the application using the NetView MS transport except outstanding replies.

The service routine returns control to the invoking component after it has processed the registration or deregistration request.
The DSI6REGS macro can be invoked only from a task within the address space of NetView.

The syntax for macro DSI6REGS is:

```
DSI6REGS

label

);x0-VVVV);

DSI6REGS SWB=swbname

(label)

,TYPE=REGMSAPPL

,DEREGMSAPPL

,REGOMSERVD

,DEREGOMSERVD

);x0-VVVV

);x0-VVVV

,APPL=applname

(label)

,FOCALPT=NO

,COMMAND

,FOCALPT=YES

,FPCAT

,REPLACE=YES

,REPLACE=NO

,REPLACE=YES

,NOTIFY=NONE

,NOTIFY=ALL

,NOTIFY=ERROR

,NOTIFY=NONE

,PRI=LOW

,PRI=HIGH

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PRI=LOW

,PR

```
name within this *applname* field is either an architected served application name, made up of hexadecimal and EBCDIC characters, or an installation-defined name, using only the EBCDIC characters 0 - 9 and A - Z (capital letters only). In either case, the name is padded with blanks to 8 characters. If you use an architected name, supply a field name because hexadecimal data in a string literal is not supported. The CNMREGIST service routine truncates any trailing blanks before placing the name in a registration table.

NetView reserves application names categories for the following application names:

<table>
<thead>
<tr>
<th>Reserved Name</th>
<th>Hex Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALERT</td>
<td>X'23F0F3F1'</td>
</tr>
<tr>
<td>EP_OPS</td>
<td>X'23F0F1F6'</td>
</tr>
<tr>
<td>EP_SPCS</td>
<td>X'23F0F1F4'</td>
</tr>
<tr>
<td>HMON_DST</td>
<td>X'30F0F8F5'</td>
</tr>
<tr>
<td>HMON_OST</td>
<td>X'30F0F8F4'</td>
</tr>
<tr>
<td>LINKSERV</td>
<td>X'23F0F3F5'</td>
</tr>
<tr>
<td>MDS_ROUT</td>
<td>X'23F0F1F0'</td>
</tr>
<tr>
<td>MS_CAPS</td>
<td>X'23F0F1F1'</td>
</tr>
<tr>
<td>OPS_MGMT</td>
<td>X'23F0F1F7'</td>
</tr>
<tr>
<td>R_BRIDGE</td>
<td>X'30F0F5F9'</td>
</tr>
<tr>
<td>RMTCMD_O</td>
<td>X'30F0F7F2'</td>
</tr>
<tr>
<td>RMTCMD_R</td>
<td>X'30F0F5F5'</td>
</tr>
<tr>
<td>RMTCMD_S</td>
<td>X'30F0F7F0'</td>
</tr>
<tr>
<td>SPCS</td>
<td>X'23F0F1F5'</td>
</tr>
<tr>
<td>STATUS</td>
<td>No hexadecimal equivalent</td>
</tr>
<tr>
<td>No character equivalent</td>
<td>X'23F0F0F1'</td>
</tr>
<tr>
<td>No character equivalent</td>
<td>X'30F0F7F3'</td>
</tr>
</tbody>
</table>

This is a required operand.

**COMMAND**

Is an 8-byte character field that specifies the name of the command procedure that is driven with any data that has a destination name equal to the one in the *applname* operand. The only exception is for replies to requests that specified a command on the CNMSENDMU invocation. The data is received as a message unit. This is either a command name or a register containing a pointer to it.

This operand is required for registration requests and not valid for deregistration requests.

**FOCALPT**

A fullword character field that specifies whether the MS application is a focal point application.

**NO**

- Specifies that the MS application is not a focal point application. This is the default.

  This operand is required for registration requests and not valid for deregistration requests.

**YES**

- Specifies that the MS application is a focal point application. A value of YES is appropriate only for an MS application registration. It is not acceptable for an operations management served application.
Notes:
1. If an MS application registers as a focal point application and later the same MS application registers again with a focalpt of NO, the application still receives focal point data from any node that did not attempt to send while the application was unregistered.
2. When you specify YES, the focal point category name is the application name specified in applname. If you specify fpcategory as well, it must be the same.

FPCAT
An 8-byte character field that specifies the focal point category about which the registrant wishes to receive focal point information. Focal point category names are the applnames of MS applications that are focal point applications (registered with FOCALPT = YES). They are specified as described under APPL.

If a focal point for this category exists at the time of registration, the command processor specified with the cmdname operand is scheduled to run under the current task. On the command processor's initial data queue is an MDS-MU with a control point management services unit (CP-MSU) containing an MS capabilities (MS_CAPS) major vector with a X'E1' subvector informing the application of its focal point information. Whether a focal point exists initially or not, all subsequent changes in the focal point information for the fpcategory are conveyed to the registering application from the MS_CAPS application or from EP_OPERATIONS_MGMT function in the same way using an X'E1' subvector in an MS capabilities major vector.

NetView provides focal-point applications for the following focal point categories:
- ALERT_NETOP (X'23F0F3F1')
- OPERATIONS_MGMT_NETOP (X'23F0F1F7')
- SPCS (X'23F0F1F5. )

This operand is required for registration requests and is not valid for deregistration requests.

NOTIFY
Is a literal value that specifies whether or not this registration is to supply an MDS error message if connectivity to other nodes is lost. NONE is the default.

ALL
Specifies that this registration receives all the notifications that ERROR provides, and in cases when the high-performance transport classifies a session outage as normal and does not attempt to reestablish connectivity.

ERROR
Specifies that this registration receives notification if there is a normal or abnormal loss of connectivity to another node. A normal loss can occur if you have outstanding transactions with the other node. An abnormal loss can occur if you have a session outage and connectivity cannot be reestablished.

NONE
Specifies that this registration does not receive any error notification. NONE is the default.

PRI
Supplies the MQS priority for incoming requests. The MQS priority is used when the MS or high performance transport uses the MQS for processing of
any received solicited MDS-MUs. The value is one of the four strings HIGH, NORMAL, LOW, or TEST. The priority value must be entered as 8-bytes, including blanks. For example:

'LOW'
'HIGH'
'NORMAL'
'TEST'

The single quotes are shown only to demonstrate the 8-byte field. Do not include them as part of the priority specification.

Priorities are:

**HIGH**
Processing begins after any NORMAL requests currently in progress, but before queued NORMAL or LOW requests.

**LOW**
Processing is preempted by HIGH and NORMAL priority requests. This is the default.

**NORMAL**
Processing priority preempts a queue of LOW priority requests.

**TEST**
DSIMQS queues the request at either HIGH or LOW priority, according to the destination task’s command priority. Refer to the OVERRIDE command in the NetView online help for an explanation of command priorities.

**REPLACE**
Specifies whether this registration supersedes a previous registration.

**NO**
Specifies that this registration cannot replace any current registration for the same application. If the same MS application or operations management served application is already registered, the service routine returns a return code of 2.

This operand is required for registration requests and not valid for deregistration requests.

**YES**
Specifies that this registration replaces any current registration for the same application. This is the default. If you specify YES, an application can:
- Change the name of the command driven to process data received
- Change the name of the task where the command is driven with unsolicited data
- Change whether or not the application is to receive focal point information

**SWB**
Is the symbolic name, or register of a fullword area, containing the address of the SWB. This is a required operand.

**TYPE**
Specifies the type of request:

**REGMSAPPL**
Register an MS application to the NetView MS transport.

**REGOMSERVD**
Register a second-level application to operations management. This makes the application a served application of operations management.
DEREGMSAPPL
Deregister an MS application.

DEREGOMSERVD
Deregister an operations management served application.

Return Codes in Register 15

The following return codes for macro DSI6REGS are found in register 15:

0  The function was successful. The application (APPL) was registered or deregistered.
20 Deregistration unsuccessful. APPL is not registered.
24 Registration unsuccessful. No storage available.
44 Deregistration unsuccessful; issued from an asynchronous exit.
104 Registration unsuccessful. APPL is already registered.
472 Registration/deregistration unsuccessful. The resource user queue is full.
476 Registration/deregistration unsuccessful. APPL name syntax is not correct.
480 Registration/deregistration unsuccessful. APPL name is restricted.
484 Registration/deregistration unsuccessful. FPCAT name syntax is incorrect.
488 Registration/deregistration unsuccessful. FPCAT has an incompatible value.
592 Registration/deregistration unsuccessful. Incorrect NOTIFY value.
9012 Registration unsuccessful. COMMAND not valid.
9016 Registration unsuccessful. The current task is not authorized for COMMAND.
9020 Registration unsuccessful. The current task and COMMAND are not compatible.

Notes:

1. When an operations management served application is registering, the only valid value for fpcategory is OPERATIONS_MGMT_NETOP (X'23F0F1F7'). This is either a fpcategory field or a register containing a pointer to it.
2. If two nodes in two networks have the same LU name, VTAM can locate either one, depending on the active configuration.
3. The NetView task where an MS application or an operations management served application receives an MDS-MU is determined as follows:
   • For a multiple-domain support (MDS) reply, the task is the task under which the requesting MS application or operations management served application was running.
   • For a request MDS-MU:
     – For an MS application, the task is the registered task for the MS application.
     – For an operations management served application, the task is determined from the destination instance identifier in the R&TI if it is present. Otherwise, the task is the registered task for the served application.
   • For an MDS error message:
If the unit of work correlator (UOWC) matches an active unit of work in one of the active transaction lists:
- For an outgoing request, the task is the task under which the requesting MS application or operations management served application that received the request was running.
- For an incoming request, the task is the task under which the MS application or operations management served application that received the request was running.
- If the UOWC does not match an active unit of work, the task is the registered task for the MS application or operations management served application.

DSI6SNDS: Send a Message Unit

Macro DSI6SNDS enables MS applications or operations management served applications on NetView to send data to a specified target in the same domain or any LU in any domain. DSI6SNDS is not available if DSI6DST is inactive.

Invoking this service routine causes the requested data to be given to the NetView MS transport. The NetView MS transport uses an LU 6.2 conversation, and VTAM selects the appropriate session for the actual transmission.

The data to be sent is in the form of an MDS-MU. You can supply the following information for the MDS-MU header:
- A completely built MDS-MU.
- An MDS-MU that is missing one or more of the following:
  - A unit-of-work correlator (UOWC)
  - An origin NETID
  - An origin LUNAME

These are added by the service routine.
- Data (for example, a CP-MSU) and sufficient other parameters for the service routine to build an MDS-MU header.

If the destination MS application is OPERATIONS_MGMT_NETOP or EP_OPERATIONS_MGMT, the invoking program must include the routing and targeting instructions (R&TI) GDS variable within the CP-MSU embedded in the MDS-MU being sent.

The DSI6SNDS macro builds the necessary NetView message queueing service (MQS) buffer with the specified data and enqueues it for the NetView MS transport.

The service routine returns control to the invoking component after it has successfully queued the request to the NetView MS transport.

The syntax for macro DSI6SNDS is:

```
DSI6SNDS

label DSISNDS swbname  (register) DATATYP=MDSMU

```

DSI6REGS
Chapter 8. Macros 265
DESTLU

\[ ,\text{DESTLU} = \text{destlu} (\text{register}) \]

DESTAPP

\[ ,\text{DESTAPP} = \text{destappl} (\text{register}) \]

MUTYPE

\[ ,\text{MUTYPE} = \text{mutype} \]

PRI

\[ ,\text{PRI} = \text{priority} (\text{register}) \]

Where:

CORAREA

Is a 52-byte varying length character field in which a new unit of work correlator (X'1549') GDS variable is created and returned by the DSI6SNDS macro. The length subfield (the first 2 bytes) indicates the length of the correlator. The correlator length is always 52 bytes for this release of the NetView program.

If you specify CORAREA for an MDS-MU, the NetView program creates the unit of work correlator in this area and inserts it into the buffer for the high-performance transport. In this case, NETID and LUNAME are inserted into the origin location (X'81') subvector at the same time if they are not already present (if there are no X'01' and X'02' subfields within the subvector). If you omit CORAREA, the MDS-MU must be complete and ready to be transmitted as supplied.

For NONMDSMU specify either CORAREA or SUPCORR. If you specify CORAREA, DSI6SNDS creates the unit of work correlator GDS variable in this area and uses it in building the MDS header. The service routine uses the supplied value in building the MDS header if you specify SUPCORR. No validity checking is done for a correlator the invoker supplies.

If this is an MDS reply or an MDS error message, do not use CORAREA, because an MDS reply or error message must return the correlator sent with the request. The invoking application must supply the original correlator either in the MDS-MU or with the SUPCORR keyword.

CORAREA is mutually exclusive with the SUPCORR keyword.

You can specify either the name of the area or a register containing a pointer to the area.
DATA
Is a varying length character field containing the data being sent. For either
MDSMU or NONMDSMU, the first two bytes must contain the entire length of
the data and the next two bytes must contain the key. The maximum length of
the data is as defined in the SNA architecture for the data object being sent.

For an MDS-MU, all fields within the MDS-MU header must be properly
prepared before invocation (with the possible exception of the correlator and
the origin NETID and LUNAME). If the correlator is not contained in the data,
specify correlarea.

You can specify either the name of the data or a register containing a pointer
to the data.

DATATYP
Is an 8-byte character field that indicates that the data item specified with the
DATA keyword is an MDS-MU or a non-MDS-MU.

MDSMU
Indicates that the DATA keyword is an MDS-MU. MDSMU is the
default.

NONMDSMU
Indicates that the DATA keyword is not a complete MDS-MU because
it does not contain an MDS-MU header. The DSIHSNDS macro
envelopes this data in an MDS-MU header before sending it.

DESTAPP
Is an 8-byte character field that specifies the destination application name.
The application name can be one of the following:
• An architecturally defined fullword value (padded with blanks to 8 bytes)
  for MS application programs.
• A 1- to 8-character installation-defined name (padded with blanks). Use the
  EBCDIC characters 0–9 and A–Z (capitals only).

The DSI6SNDS macro truncates any trailing blanks before putting this value in
the MDS-MU header.

You can specify either the destination application name or a register containing
a pointer to the destination application. The macro automatically translates the
following destination application names to hexadecimal:
• EP_OPS
• OPS_MGMT
• ALERT

DESTAPP is required for NONMDSMU.

DESTLU
Is an 8-byte character field that specifies the LU or VTAM CP name of the
destination LU or VTAM CP. Specify the 1- to 8-character LU or VTAM CP
name (padded with blanks to 8 characters) using only the EBCDIC characters
0–9 and A–Z (capitals only). If DESTLU is not specified, the default is the CP.

The DSI6SNDS macro truncates any trailing blanks before putting this value in
the MDS-MU header.

This field is required for NONMDSMU.

DESTNET
Is an 8-byte character field that specifies the ID of the network of the
DSI6SNDS

destination LU or VTAM CP name. You must specify the 1- to 8-character NETID (padded with blanks to 8 characters) using only the EBCDIC characters 0–9 and A–Z (capitals only).

The DSI6SNDS macro truncates any trailing blanks before putting this value in the MDS-MU header. If you specify eight blanks as the value, the effect is the same as not specifying the keyword. The default is the network name determined by VTAM based on the LU or VTAM CP name of the remote node you specify with the DESTLU keyword.

You can specify either the destination NETID name or a register containing a pointer to the destination NETID.

If you do not specify the DESTNET keyword when using DSI6SNDS for NONMDSMU data types, the value of this field defaults to the NETID determined by VTAM. Otherwise, this field is required for NONMDSMU data types.

MUTYPE

Is a 4-byte integer field that specifies the index number that identifies the type of MDS-MU to build. The type identifies whether or not the MDS-MU is a request, a reply, or an error message, and whether or not additional messages are expected. The following types are defined as constants:

<table>
<thead>
<tr>
<th>MUTYPE</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_WITH_REPLY</td>
<td>1</td>
</tr>
<tr>
<td>REQUEST_WITHOUT_REPLY</td>
<td>2</td>
</tr>
<tr>
<td>REPLY_ONLY</td>
<td>3</td>
</tr>
<tr>
<td>REPLY_NOTLAST</td>
<td>4</td>
</tr>
<tr>
<td>REPLY_LAST</td>
<td>5</td>
</tr>
<tr>
<td>ERROR_MESSAGE</td>
<td>6</td>
</tr>
</tbody>
</table>

The DSI6SNDS macro uses the MUTYPE value to determine the settings for the MDS message type and first and last message indicator bits in the flags (X'90') MDS routing information subvector.

This is a required keyword for NONMDSMU.

ORIGAPP

Is an 8-byte character field that specifies the origin high-performance application name.

The application name can be one of the following:

• An architecturally defined fullword value (padded with blanks to 8 bytes) for MS application programs.

• A 1- to 8-character installation-defined name (padded with blanks). You must use the EBCDIC characters 0–9 and A–Z (capitals only).

The DSI6SNDS macro truncates any trailing blanks before putting this value in the MDS-MU header.

You can specify either the origin application name or a register containing a pointer to the origin application.

This field is required for NONMDSMU.

PRI

Supplies the MQS priority for incoming reply or MDS error message resulting
from any outgoing MDS-MU. It must be the name or address of an 8-byte (including blanks) character field that contains the desired priority. Its value, including blanks, can be:

'LOW'
'HIGH'
'NORMAL'
'TEST'

SYNCH
Specifies the buffering options available on requests_with_replies:

**NO_BUF**
Specifies to buffer replies until reply_last is received. This is the default.

**NO_UNBUF**
Specifies to send each reply to the application as it is received.

REPCMD
Is an 8-byte character field containing the name of the command to be driven with the reply. You can specify REPCMD only in an application that is sending REQUEST_WITH_REPLY, with the reply being received asynchronously.

If REPCMD is not specified when a high-performance application sends a REQUEST_WITH_REPLY, the value defaults to the registered command of the high-performance application at sending time. If the high-performance application issues the DSIHREGS macro with the REPLACE(YES) option before the reply is received, the reply is sent to the original registered command when it comes in.

You can specify either the reply command processor name or a register containing a pointer to the reply command processor.

This is an optional field. The default is the registered command for the invoking application.

SECONDS
Is a fullword integer field that specifies the number of seconds to wait for the reply of an outstanding REQUEST_WITH_REPLY. For a REQUEST_WITH_REPLY that generates multiple replies, the time-out value applies only to the last reply.

The NetView program initializes default and maximum time-out values for the LU 6.2 transport send services. The initial default and maximum time-out values are 120 and 86400 seconds, respectively. If you specify a value of X'FFFFFFFF' (-1), the maximum time-out value is used. The maximum value is initialized to 86000 (24 hours). You can change these values with the DEFAULTS command.

The following values are valid for SECONDS:

1 ... X
Where X is the maximum time-out value
0 Indicates the default time-out value
-1 Indicates the maximum time-out value

If you do not specify the SECONDS keyword when using the DSI6SNDS macro for a REQUEST_WITH_REPLY, the default time-out value is used. Otherwise, this field is required for REQUEST_WITH_REPLY.

SUPCORR
Is a varying length character field containing a complete unit of work correlator (X'1549') GDS variable. The SUPCORR field must contain a 2-byte
length, a 2-byte key, and at least 1 byte of correlator data. Refer to the SNA library for more information about defining the correlator.

SUPCORR is not valid for an MDS-MU. If you use an existing unit of work or create a new one, the MDS header must contain the unit of work and the MDS-MU must be ready to be transmitted as supplied. An MNOTE is returned if you specify this keyword for an MDS-MU.

SUPCORR is optional for NONMDSMU. For NONMDSMU, specify either SUPCORR or CORAREA. The supplied value is used to build the MDS header if you specify SUPCORR. No validity checking is done for a correlator supplied by the invoker. If an MDS reply or an MDS error message is sent, specify SUPCORR because an MDS reply or error message must return the correlator sent with the request. The invoking application must supply the original correlator either in the MDS-MU or with the SUPCORR keyword.

You can specify the correlator name or a register containing a pointer to the data.

**SWB**

Is the register, or symbolic name of a fullword area, containing the address of a service work block (DSISWB).

**Return Codes in Register 15**

The following return codes for macro DSI6SNDS are found in register 15:

- 0  Requested function was performed.
- 4  Task is terminating (TVBABEND/TVBLOGOF is on), and a request-with-reply is sent.
- 24  No storage.
- 44  Send MU service is invoked in an asynchronous exit.
- 56  Time-out value is not valid.
- 88  MDS-MU length is not valid.
- 220  DSI6DST task inactive.
- 400  Data type is not valid.
- 404  DATA missing or is not valid.
- 408  MS application cannot send data to the same MS application within the same NetView program.
- 416  MS application not registered.
- 420  Operations management served application not registered.
- 424  UOW missing or is not valid.
- 428  R&TI missing or is not valid.
- 432  Origin application name (OAN) missing or is not valid.
- 436  Destination application name (DAN) is not valid.
- 440  OAN is not valid.
- 444  Destination network ID missing or is not valid.
- 448  Destination LU name missing or is not valid.
- 452  Destination application name missing or is not valid.
OII in R&TI does not match TVBOPID.

Reply is not valid.

Incorrect MUTYPE given.

User list is full.

Operations management served application cannot send data to the same operations management served application within the same NetView program.

The task does not have authorization to execute the registered command associated with the origin application name or OAN.

The operations management served application cannot send MDS error message. Routing report should be sent instead.

MDS error message does not have SNA Condition Report.

NETID is not available. The probable cause is that VTAM is not active.

The variable \( x \) is the return code from DSIMQS.

The variable \( x \) is the return code from DSIPUSH.

A return code of 24 or 28 from DSIPUSH indicates that DSIOLOGFP is not defined or is not defined properly in DSICMD.

The variable \( Y \) is the return code from DSICES. Ensure that DSi6SNDS is defined correctly in DSICMD. If you specify a reply command processor, ensure that it is also defined properly in DSICMD. If the sending application is an operations management served application, make sure DSIOARCP is defined properly in DSICMD.

Notes:

1. For MUs sent within the same NetView program, the send services uses the NetView LU name as the origin LU.
2. A request can time out in less than the specified interval if the receiving partner has implemented a maximum time-out value of less than 24 hours, or the one specified with the DEFAULTS command.
3. The parameter specified by SECONDS is either the variable name (\( timeout \)) containing the time interval value, or the value itself.
4. Control is returned to the invoking program after DSI6SNDS successfully queues the request to the MS transport.
5. For MDSMU, all fields within the MDS-MU header must be correct except for origin NETID and LUNAME. The macro can determine and set these fields. If the data does not contain the correlator, you must specify CORAREA.
6. For REPLY_ONLY, REPLY_NOTLAST, REPLY_LAST, and ERROR_MESSAGE, you must specify SUPCORR to return the correlator sent with the request.
7. The MS transport implements a time-out value for the application receiving the data. If the invocation of DSI6SNDS specifies a time-out value greater than the time-out value set by the transport at the receiving node, the sending application might time out in less than the specified interval.
8. When VTAM is active, you can use DSi6SNDS to send data to another application in the same domain.
9. If DESTNET is not the NETID determined by VTAM for the LU specified in DESTLU, the send fails.

10. An MS application or operations management served application cannot send data to the same application within the same NetView system.

### DUIFSMTE: Customize the DisplayStatus Table for Exception Views

NetView’s exception views function is used to show an operator network resources that are not functioning correctly. The views are set up through RODM and display only those resources assigned to them. You can use the macro DUIFSMTE to customize the mapping of exception view DisplayStatus values in RODM using the table DUIFSMT. You create statements for this table with the DUIFSMTE macro and then assemble and link-edit the table (refer to sample CNMSJH13) to create a load module.

The syntax for macro DUIFSMTE is:

```plaintext
DUIFSMTE

CLASS=class_name,

XCPT=(DEGRD),

INTER,

SATIS,

SDGRD,

UNKNOWN,

UNSAT,

RESOURCE=resource_name,

USRXMETH=method_name

END
```

**Notes:**

1. More than one value can be specified, but no value can be specified more than once.

2. More than one keyword can be specified, but no keyword can be specified more than once.

**Where:**

**CLASS=class_name**

The name of the class in RODM for which you are customizing DisplayStatus mapping. If you wish to specify the default values for classes not included in the DUIFSMT table, use the value `All_Other_Classes` for `class_name`.

If you want to customize the DisplayStatus mapping for all of the objects of a class, use one statement for that class. If you want to customize the DisplayStatus mapping for specific objects, or groups of objects, of a class, use multiple statements. Each statement with the same class name should have a different value for the RESOURCE keyword.

**Note:** Remember that names in RODM are case-sensitive.

**DEGRD**

Specifies objects with a DisplayStatus value of 133 (degraded).
END

This keyword ends table processing. DUIFSMTE END must always be the last statement in your source for the table.

INTER

Specifies objects with a DisplayStatus value of 131 (intermediate).

RESOURCE=resource_name

The name of the specific resource to which these values apply. You can use the wildcard character * (asterisk) to specify groups of resources. See "Specifying Resource Names for DisplayStatus Mapping" for more information.

SATIS

Specifies objects with a DisplayStatus value of 129 (satisfactory).

SDGRD

Specifies objects with a DisplayStatus value of 134 (severely degraded).

UNKWN

Specifies objects with a DisplayStatus value of 132 (unknown).

UNSAT

Specifies objects with a DisplayStatus value of 130 (unsatisfactory).

USRXMETH=method_name

The name of a RODM method to trigger for objects in this class; if specified, the method determines the final DisplayStatus mapping.

XCPT

Specifies DisplayStatus values of objects considered to be exceptions. Objects with these DisplayStatus values are added to an exception view if the UserStatus and ExceptionViewList criteria are also met. If only one value is specified for XCPT, the parentheses can be omitted.

Default Values for DisplayStatus

To specify the default values for all classes not defined in the DUIFSMT table, use the value All_Other_Classes for class_name. For example:

DUIFSMTE CLASS=All_Other_Classes,XCPT=(DEGRD,INTER,SDGRD,UNSAT)

These values apply to all classes unless they are overridden by other statements. You only need to code the specific classes that differ from the values you specify for All_Other_Classes.

Specifying Resource Names for DisplayStatus Mapping

You can specify the DisplayStatus mapping for specific resources or groups of resources within a class. To specify the resource name, use the RESOURCE keyword of the DUIFSMPE macro. You can use the * (asterisk) wildcard to specify groups of resources.

If you want to customize a specific resource, code the statement for that resource before any statements that would match for its class. (See Usage Note 3 on page 275.) For example, if you want resource DECNETRALV4 in class GMFHManagedList_Real_Objects_Class to map to XCPT if it has an unsatisfactory status, but you do not want other resources in that class to do the same, you must code the statement for the resource first, as shown in Figure 27 on page 274.
If the second and third DUIFSMTE statements in Figure 27 had been coded before the first DUIFSMTE statement, resource DECNET.RALV4 would have matched, and the statement for RESOURCE=DECNET.RALV4 would never be invoked.

For resources managed by the SNA topology manager, the name you specify using the RESOURCE keyword is the DisplayResourceName of the object in RODM that represents that resource. These DisplayResourceNames are composed of two or three names concatenated with periods.

The rules for the RESOURCE keyword are the same as the rules for the RESOURCE keyword in the customization tables of the SNA topology manager.

Examples of Customizing DisplayStatus Mapping

For Figure 28, assume the following:

- You want to display all objects of the t4Node (1.3.18.0.0.1844) class with a DisplayStatus of unsatisfactory or unknown in an exception view. Use the alias from Table 52 on page 275 for the class name.
- You want to display all objects of the appnEN (1.3.18.0.0.1821) class with a DisplayStatus of unsatisfactory, intermediate, or unknown in an exception view. Use the actual MyName value from Table 52 on page 275 for the class name.
- You want to display selected objects of the GMFHS_Aggregate_Objects_Class in an exception view if their DisplayStatus value is severely degraded.
- For objects in all other classes, you want to place them in exception views only if their DisplayStatus is unsatisfactory or severely degraded.

Figure 28 shows the coding of the DisplayStatus mapping table for the conditions defined above. The first statement sets the defaults.

DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class, C
   XCPT=UNSAT,RESOURCE=DECNET.RALV4
DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class
DUIFSMTE CLASS=1.3.18.0.0.1821,XCPT=(DEGRD,INTER, C
   SDGRD,UNSAT)
DUIFSMTE CLASS=GMFHS_Managed_Real_Objects_Class
DUIFSMTE CLASS=GMFHS_Aggregate_Objects_Class,XCPT=SDGRD
DUIFSMTE CLASS=All_Other_Classes,XCPT=(SDGRD,UNSAT)
DUIFSMTE END

Figure 28. DisplayStatus Mapping Table Coding Example 1

The example in Figure 29 on page 275 assumes the following:

- You have created a RODM method named CUSTMTH1 to decide whether objects of the t2_1Node should be displayed in Exception views based on the values of other fields in RODM.
- You do not want any objects of the crossDomainResourceManager class to be displayed in any exception views.
- You want the object in the appnEN class with a DisplayResourceName of USIBMNT.NCPPU1 to appear in exception views regardless of its status.
• You want any object in the appnEN class with the SNA network ID portion of the DisplayResourceName of USIBMNT to appear in exception views if its status is not satisfactory.

Figure 29 shows the coding of the DisplayStatus mapping table for the conditions defined above.

```none
DUIFSMTE CLASS=T21NODE,USRXMETH=CUSTMTH1
DUIFSMTE CLASS=CROSSDOMAINRESOURCEREPER
DUIFSMTE CLASS=APPEN, XCPT=(DEGRD, INTER, SATIS, SDGRD, C UNKNW, UNSAT, RESOURCE=USIBMNT.NCP1)
DUIFSMTE CLASS=APPEN, XCPT=(DEGRD, INTER, SDGRD, UNKNW, C UNSAT), RESOURCE=USIBMNT.*
DUIFSMTE END
```

Figure 29. DisplayStatus Mapping Table Coding Example 2

### Assembling and Link-Editing the DisplayStatus Mapping Table

After you modify the DisplayStatus mapping table (DUIFSMTE), assemble and link-edit it. NetView supplies a sample job named CNMSJH13 to assemble and link-edit DUIFSMTE and refresh the NetView method that uses DUIFSMTE. The modified values will apply to all objects that change their DisplayStatus value after you run CNMSJH13. You can optionally specify that CNMSJH13 will trigger a recalculation of DisplayStatus mapping for all objects in RODM.

**Notes:**

1. DUIFSMTE must start in column 1. The keywords start in column 16.
2. If a statement exceeds 71 characters, put a continuation character in column 72 and continue the statement in column 16 of the next line.
3. If you enter more than one statement with the same *class_name* and *resource_name* values, the first statement is used and the other statements are ignored; a warning message is issued.
4. You can use alias values for classes managed by the SNA topology manager. Table 52 lists the aliases you can enter and their corresponding actual class names as known to RODM; both are accepted by the DUIFSMTE macro.

#### Table 52. Aliases for RODM Class Names

<table>
<thead>
<tr>
<th>Alias for Class</th>
<th>MyName Value for Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPNENP</td>
<td>1.3.18.0.0.1821</td>
</tr>
<tr>
<td>APPNENN</td>
<td>1.3.18.0.0.1822</td>
</tr>
<tr>
<td>APPNTRANSMISSIONGROUPPP</td>
<td>1.3.18.0.0.1823</td>
</tr>
<tr>
<td>APPNTRANSMISSIONGROUPPCIRCUITPP</td>
<td>1.3.18.0.0.2058</td>
</tr>
<tr>
<td>CROSSDOMAINRESOURCEP</td>
<td>1.3.18.0.0.2281</td>
</tr>
<tr>
<td>CROSSDOMAINRESOURCEREPER</td>
<td>1.3.18.0.0.2278</td>
</tr>
<tr>
<td>DEFINITIONGROUPPP</td>
<td>1.3.18.0.0.2267</td>
</tr>
<tr>
<td>INTERCHANGENODEP</td>
<td>1.3.18.0.0.1826</td>
</tr>
<tr>
<td>LENNODEP</td>
<td>1.3.18.0.0.1827</td>
</tr>
<tr>
<td>LOGICALLINKPP</td>
<td>1.3.18.0.0.2085</td>
</tr>
<tr>
<td>LOGICALUNITPP</td>
<td>1.3.18.0.0.1829</td>
</tr>
<tr>
<td>MIGRATIONDATAHOSTPP</td>
<td>1.3.18.0.0.2155</td>
</tr>
</tbody>
</table>
### Table 52. Aliases for RODM Class Names (continued)

<table>
<thead>
<tr>
<th>Alias for Class</th>
<th>MyName Value for Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTP</td>
<td>1.3.18.0.0.2089</td>
</tr>
<tr>
<td>T2-1NODEP</td>
<td>1.3.18.0.0.1843</td>
</tr>
<tr>
<td>T4NODEP</td>
<td>1.3.18.0.0.1844</td>
</tr>
<tr>
<td>T5NODEP</td>
<td>1.3.18.0.0.1845</td>
</tr>
</tbody>
</table>

**Note:** For more information, refer to the [Tivoli NetView for z/OS SNA Topology Manager Implementation Guide](#) and the [Tivoli NetView for z/OS Resource Object Data Manager and GMEHS Programmer’s Guide](#).
Chapter 9. Called Service Routines

This chapter explains called service routines and lists interfaces for the assembler language.

Overview of Called Service Routines

Called service routines are services that NetView provides. These routines are similar to the ones provided by NetView macros (such as DSIPSS), but they are called directly through branch and link. These routines adhere to standard linkage conventions.

The called service routines are supported in both 24-bit and 31-bit addressability mode. You should use these routines in 31-bit mode whenever possible. Using 24-bit mode uses more storage below the 16-megabyte line and increases CPU usage.

Upon routine entry, the registers contain the following information:

1 The address of a standard parameter list.
   The list consists of addresses to parameter data. The parameter data is specified by the individual routine descriptions. The end of the list needs to be indicated by the last address having the high-order bit set to binary 1. All other addresses need to have that bit set to binary zero.

2–12 Unspecified.

13 The address of a standard 72-byte save area.

14 The return address at the time of the call.

15 The address of the service routine to be called. The DSISVL control block typically has the addresses of the services defined in it.

On routine return, register 15 contains a return code. All other registers are reserved.

In addition, fields defined by the service routine description in the parameter list can be used for return values.

Free NetView Buffers Service Routine (DSIFREBF)

You can use this service routine to free NetView buffer structures, such as automation internal function request buffers. This routine provides an easy method for freeing a buffer structure without concern for the number of buffers that are related. This is especially helpful for upgrading existing programs to handle the structured automation internal function request (AIFRs) with DSIAIFRO automation extensions.

You can also use this routine to free a chain of buffers that use HDRNEXTM as the chaining field, provided the chain is terminated with a zero value in the last buffer. If a buffer on the chain is an AIFR structure, all buffers chained to the AIFR are freed, using all of the defined buffer pointers within the IFRAUTO mapping.
The DSIFREBS macro is provided for calling DSIFREBF. By using DSIFREBS you do not have to calculate the offset of DSIFREBF in the SVL.

The format for the DSIFREBS macro is:

```assembly
LA 13,savearea
LA 1,plist           Your parameter list
DSIFREBS            Call DSIFREBF
LTR 15,15           Check for errors
BNZ ERROR
```

Where:

- **plist**
  - Is a standard parameter list in the following format:
    - **Word 1**
      - The address of a fullword containing the TVB address for the task.
    - **Word 2**
      - The address of a fullword containing the address of the buffer chain to be freed.

- **savearea**
  - Is a standard 72-byte save area, usually at the start of the code.

**Return Code**

The following return code is for the DSIFREBF routine:

- **0** The requested function was performed.

**Abend Code**

The following abend code is for the DSIFREBF routine:

- **18** The DSIFREBF service was unable to free one of the chained buffers.

**Notes:**

1. The DSIFREBF service routine requires that the parameter list contain 31-bit addresses of the parameters. The address to be freed must be a valid 31-bit address.
2. Buffers being freed need to be in subpool zero, and they need to be obtained with the Q=NO option of DSIGET.
3. HDRNEXTM in the buffer to be freed is zero unless a chain of buffers is to be freed.
4. The linkage registers of this routine are standard.

---

**Process Message Data Block Routine (DSIMMDB)**

This routine accepts an MDB and a source object, transforms them into an AIFR, and processes the result as in normal NetView message processing. A source object is optional, and you can indicate no source object by using an address of zero. Also, specify a 16-byte field, which is used as a correlator value.

Any program in the NetView address space, except those running in an asynchronous exit, can invoke the process message data block (MDB) service routine. However, you cannot run this routine if TVBINXIT is on.

Although the MDB architecture is of MVS origin, any operating system can introduce messages to NetView by building MDBs and calling this service routine.
This service routine executes on the NetView task of the invoking program and returns control to the invoker after it scans the NetView automation table and after it completes the synchronous portion of message processing.

The DSIMMDBS macro is provided for calling DSIMMDB. By using DSIMMDBS, you do not have to calculate the offset of DSIMMDB in the SVL.

The format for the DSIMMDBS macro is:

```
LA 13,savearea
LA 1,plist       Your parameter list
DSIMMDBS        Call DSIMMDB
LTR 15,15       Check for errors
BNZ ERROR
```

Where:

**plist**
Is a standard parameter list in the following form:

**Word 1**
The address of a fullword containing the TVB address for the task.

**Word 2**
The address of a fullword containing the address of the MDB to be processed.

**Word 3**
The address of a fullword containing the address of the source object. The source object is optional. If you do not want to specify a source object, Word 3 contains the address of a value of zero.

**Word 4**
The address of a fullword containing a 16-byte correlator field. This field needs to be all binary zeros for a single MDB, or all binary zeros for the first MDB of a set of related MDBs.

This field needs to be task reentrant, and enable DSIMMDB to write a 16-byte correlator value into it. For related MDBs, save this correlator value and pass it back to DSIMMDB on each subsequently related MDB. When an MDB is processed with an end-of-text indication, the correlator is returned to you with a zero value.

**savearea**
Is a standard 72-byte save area, usually at the start of the code.

**Return Codes**
The following return codes are for this macro:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The requested function was performed.</td>
</tr>
<tr>
<td>24</td>
<td>Storage was unavailable.</td>
</tr>
<tr>
<td>564</td>
<td>An error was detected in the correlation parameter. A diagnostic dump of the MDB will be written to the NetView log if the address of the MDB is not zero. A diagnostic dump of the source object will be written to the NetView log if the address of the source object is not zero.</td>
</tr>
<tr>
<td>568</td>
<td>The first operand is not a valid MDB. A diagnostic dump of the MDB will be written to the NetView log if the address of the MDB is not zero. Additionally, a diagnostic dump of the source object will be written to the NetView log if the address of the source object is not zero.</td>
</tr>
</tbody>
</table>
DSIMMDB

572 The second operand is not a valid source object. A diagnostic dump of the MDB will be written to the NetView log if the address of the MDB is not zero. Additionally, a diagnostic dump of the source object will be written to the NetView log if the address of the source object is not zero.

Notes:
1. This service does not free the MDB or source object.
2. Use the MVS control block IEAVM105 as a guide for building your MDB.
3. To chain related MDBs, use the correlator parameter (Word 4) to DSIMMDB in place of the IEAVG132 map.
4. If IEAVM105 is not available, you can use parts of the DSIAIFRO map to build your MDB, as follows:
   a. Begin your MDB at DSIMGO level.
   b. Change the type code for DSIMGO to X'0001' to indicate that this is an incoming MDB.
   c. Include one DSICPO, one DSIGOJ, and as many DSITOJ objects as needed to contain your text.
5. The first character in your text object is discarded; it cannot be a meaningful part of your message. The second character is discarded if both of the following conditions exist:
   The character is a plus sign (+)
   MDBCAUTH (CPOCAUTH) is set on (set to 1).
6. MDBs created using this service routine are submitted directly to NetView; the operating system is not involved in the processing or routing of the MDB. Therefore, items that would normally be subject to MVS system routing (such as console ID and route code) are ignored; the MDB is delivered only to the task that invokes the service routine.
7. When creating DOM MDBs, be aware that not all forms of DOM can be transported over OST-NNT sessions. When using those sessions, only create DOM MDBs that indicate DOM by MSGID.

Note: For detailed information about MDB formats, refer to the description of MDB in the z/OS library.
Part 3. Appendixes
Appendix A. Assembler Samples

This appendix provides a table of the assembler samples that are shipped as part of the NetView sample library. These samples are distributed as members of the CNMSAMP data set.

Each sample described in this section has two names, such as ATMPCMDP (CNMS4202). The CNM name is the name under which the file or member is distributed on the sample library tape. The first name is a more meaningful alias name to which the CNM file is copied upon installation.

Follow these steps to enter the member names as commands:

1. Assemble and link-edit the samples using the alias name.
2. Delete the asterisk (*) in column 1 of the appropriate CMDMDL statement in DSICMD, enabling you to execute the alias name as a command. No entries are needed in DSICMD for installation exits.
3. Recycle NetView to pick up the DSICMD changes.

This appendix also contains a description of each sample. The last section provides coded examples of an installation exit routine, a command processor, a user-written function, and a user-written optional subtask.

Notes:

1. Refer to the prologs of the samples for information about how certain samples are related and about special cases for installation exit routines and other samples (such as DSIUSR00).
2. The alias name is the control section (CSECT) name.

Assembler Samples Reference Table

Table 53 shows the assembler language samples that are shipped with the NetView product. The table contains the function, the alias name, and the name of the member in NETVIEW.V5R1M0.CNMSAMP.

Table 53. Assembler Language Samples Shipped with NetView

<table>
<thead>
<tr>
<th>Assembler Alias</th>
<th>CNMSAMP Member</th>
<th>Sample Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAUTOTB</td>
<td>CNMS4291</td>
<td>Sends MSU to automation table</td>
</tr>
<tr>
<td>ABLDMSG</td>
<td>CNMS4278</td>
<td>Uses DSIMBS to build messages</td>
</tr>
<tr>
<td>ACALLCMD</td>
<td>CNMS4280</td>
<td>Calls another command</td>
</tr>
<tr>
<td>ADATTIM</td>
<td>CNMS4274</td>
<td>Uses DSIPSS to display date and time</td>
</tr>
<tr>
<td>AGETDS</td>
<td>CNMS4294</td>
<td>DSIGETDS sample command processor</td>
</tr>
<tr>
<td>AHREGSTR</td>
<td>CNMS4296</td>
<td>Register/deregister an application to the LU 6.2 high-performance transport</td>
</tr>
<tr>
<td>AHSNDMU</td>
<td>CNMS4286</td>
<td>Sends an alert through the LU 6.2 high-performance transport</td>
</tr>
<tr>
<td>ALERTMSG</td>
<td>CNMS4284</td>
<td>Creates a customizable message for automation</td>
</tr>
<tr>
<td>ALISTMEM</td>
<td>CNMS4276</td>
<td>Lists a member of DSIPARM</td>
</tr>
<tr>
<td>AMLWTO</td>
<td>CNMS4273</td>
<td>Uses DSIPSS for title-line output</td>
</tr>
</tbody>
</table>
### Assembler Samples

<table>
<thead>
<tr>
<th>Assembler Alias</th>
<th>CNMSAMP Member</th>
<th>Sample Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMSGMOD</td>
<td>CNMS4271</td>
<td>Uses DSIMDS to build a message module</td>
</tr>
<tr>
<td>AOPTTSK</td>
<td>CNMS4277</td>
<td>User-written optional subtask</td>
</tr>
<tr>
<td>APRSMDM</td>
<td>CNMS4299</td>
<td>Example usage of process MDB service</td>
</tr>
<tr>
<td>APSSFULL</td>
<td>CNMS4279</td>
<td>Uses DSIPSS to display a full-screen panel</td>
</tr>
<tr>
<td>AREGISTR</td>
<td>CNMS4292</td>
<td>LU 6.2 register/deregister application</td>
</tr>
<tr>
<td>ARODMCON</td>
<td>CNMS4290</td>
<td>Uses CNMQAPI to access RODM</td>
</tr>
<tr>
<td>ASENDMU</td>
<td>CNMS4293</td>
<td>Sends alert to ALERT_NETOP</td>
</tr>
<tr>
<td>ASEQLOG</td>
<td>CNMS4275</td>
<td>Logs text to a sequential log</td>
</tr>
<tr>
<td>ATMPCMDP</td>
<td>CNMS4202</td>
<td>Template for assembler command processor</td>
</tr>
<tr>
<td>ATPUXIT</td>
<td>CNMS4282</td>
<td>Template for assembler installation exit</td>
</tr>
<tr>
<td>AUSRFUNC</td>
<td>CNMS8002</td>
<td>Example of a REXX user-written function</td>
</tr>
<tr>
<td>AWRTLOG</td>
<td>CNMS4272</td>
<td>Uses DSIWLS to write a message to the NetView log</td>
</tr>
<tr>
<td>AXITCM</td>
<td>CNMS4504</td>
<td>Sample XITCM exit</td>
</tr>
<tr>
<td>AXITST</td>
<td>CNMS4285</td>
<td>Sample XITST exit</td>
</tr>
<tr>
<td>AXITVN</td>
<td>CNMS4270</td>
<td>Defines an XITVN DST exit to initialize an empty VSAM data set</td>
</tr>
<tr>
<td>CNMRECV</td>
<td>CNMS4289</td>
<td>Receives data buffers from a receiver’s queue</td>
</tr>
<tr>
<td>CNMSEND</td>
<td>CNMS4288</td>
<td>Sends a data buffer to a receiver</td>
</tr>
<tr>
<td>CNMSGENA</td>
<td>CNMS4287</td>
<td>Generic alert sender application</td>
</tr>
<tr>
<td>DSICTMOD</td>
<td>CNMS0055</td>
<td>Specifies NetView constants such as time-out values and sense code filtering</td>
</tr>
<tr>
<td>DSIEX02A</td>
<td>CNMS4283</td>
<td>Uses DSIEX02A to manipulate messages</td>
</tr>
<tr>
<td>DSIEX17</td>
<td>CNMS4297</td>
<td>Manipulates message IEF233A and its associated DOM</td>
</tr>
<tr>
<td>DSIEX19</td>
<td>CNMS4307</td>
<td>Provides command authority checking for the RUNCMD command</td>
</tr>
<tr>
<td>DSIEX21</td>
<td></td>
<td>Provides an example of a DSITCPRF encryption installation exit</td>
</tr>
<tr>
<td>DSIIRXPRM</td>
<td>CNMSJ1M11</td>
<td>Specifies the NetView REXX environment initialization parameters</td>
</tr>
<tr>
<td>DSIUSR00</td>
<td>CNMS4281</td>
<td>User-defined message member</td>
</tr>
<tr>
<td>OPERID</td>
<td>CNMS4295</td>
<td>Sample automation table function (ATF)</td>
</tr>
</tbody>
</table>

### Assembler Samples Description

Each sample provides a description of the function and the NetView service macros that are used.

**AAUTOTB (CNMS4291)**

This sample sends an MSU to the automation table for evaluation. The NetView service macros used in this sample are DSICBS, DSIAUTO, and DSIPSS.
ABLDMSG (CNMS4278)
This sample uses DSIMBS to build user-defined messages. The NetView service macros used in this sample are DSICBS, DSIDATIM, DSIDEL, DSILOD, DSIMBS, and DSIPSS.

ACALLCMD (CNMS4280)
This sample calls another command processor. The NetView service macros used in this sample are DSICBS, DSICES, DSIDATIM, DSIFRE, DSIGET, DSILCS, and DSIPRS.

ADATTIM (CNMS4274)
This sample uses DSIPSS to display date and time. The NetView service macros used in this sample are DSICBS, DSIDATIM, and DSIPSS.

AGETDS (CNMS4294)
This sample issues DSIGETDS to retrieve an MSU pointed to by TVBAIIFR. The NetView service macros used in this sample are DSICBS, DSIDATIM, DSIGETDS, and DSIPSS.

AHREGRSTR (CNMS4296)
This sample registers or deregisters an application as an LU 6.2 high performance application. The NetView service macros used in this sample are DSICBS, DSIPSS, DSI6REGS.

AHSNDMU (CNMS4286)
This sample sends a software alert from a registered LU 6.2 high-performance application. The NetView service macros used in this sample are DSICBS, DSIPSS, DSI6SNDS.

ALERTMSG (CNMS4284)
This sample creates a customizable message for automation from generic and nongeneric network major vector transport alerts. The prolog of CNMS4284 contains installation directions and customization guidelines. The DSTINIT statements of CNMS4284 show how to install the sample on the XITCI installation exit of the BNJDSERV task. The NetView macros used in this sample are DSIBAM, DSIBAMKW, DSISRCMV, and DSICVTHE.

ALISTMEM (CNMS4276)
This sample reads and displays a member from the NetView DSIPARM data set. It also does command authorization checking for the supplied parm member name to prevent unauthorized display of DSIOPF. The NetView service macros used in this sample are DSICBS, DSIDATIM, DSI6KS, DSI6KV, and DSIPSS.

AMLWTO (CNMS4273)
This sample uses DSIPSS for title-line output. The NetView service macros used in this sample are DSICBS, DSIDATIM, and DSIPSS.

AMSGMOD (CNMS4271)
This sample uses the message definition services (DSIMDS) to build a user-defined message module (AMSGMOD). It is used in conjunction with the ABLDMSG sample command processor and DSIEX02A. The NetView service macro used in this sample is DSIMDS.

AOPTTSK (CNMS4277)
This sample is an example of a user-written optional subtask. This template is available as part of the NetView sample library. It can be found as member CNMS4277 of the CNMSAMP data set.
Assembler Samples

APRSMDB (CNMS4299)
This sample shows how to build an MDB with a source object and submit it to NetView for processing.

APSSFULL (CNMS4279)
This sample uses DSIPSS to display a full-screen panel, wait for terminal input, and echo the input. The NetView service macros used in this sample are DSICBS, DSIFRE, DSIGET, DSIPSS, and DSIWAT.

AREGISTR (CNMS4292)
This sample registers or deregisters an application as an MS application or as an operation management served application. The NetView service macros used in this sample are DSICBS, DSIPSS, and DSI6REGS.

ARODMCON (CNMS4290)
This sample invokes CNMQAPI, which enables access RODM under the control of NetView. The coded example shows a RODM CONNECT function. The NetView service macros used in this sample are DSICBS and DSINOR.

ASENDMU (CNMS4293)
This sample sends a software alert to ALERT_NETOP from a registered MS application named USERAPPL. The NetView service macros used in this sample are DSICBS, DSIPSS, and DSI6SNDS.

ASEQLOG (CNMS4275)
This sample logs text to a sequential log. The NetView service macros used in this sample are DSICBS, DSIDATIM, DSIMQS, DSIPSS, and DSIWLS.

ATMPCMDP (CNMS4202)
This sample is a template for command processors in the assembler language. This sample is included in "Template for a Command Processor" on page 28.

ATMPUXIT (CNMS4282)
This sample is a template for assembler installation exits. See "Template for an Installation Exit Routine" on page 61.

AUSRFUNC (CNMS8002)
This sample is an example of a user-written REXX function. The example returns the character string “Hello World” to the calling REXX command list. It shows entry linkage, allocates an SWB, sets up a result, issues DSIRXEBS to obtain a new evaluation block, returns the result to the caller, and frees the SWB and all other temporary storage. The DSIRXEBS is necessary only when the evaluation block that was passed on entry is too small to contain the result. DSIRXEBS is in the example only to show a sample usage. Also, the SWB is needed only if a NetView service call is generated.

AWRTLOG (CNMS4272)
This sample uses DSIWLS to write a message to the NetView log. The NetView service macros used in this sample are DSICBS, DSIDATIM, and DSIWLS.

AXITCM (CNMS4504)
This sample provides an example for how to modify a command string sent to the host from an NMC workstation. The sample can be modified to suit the needs of your installation.

AXITST (CNMS4285)
This sample takes information from the resource status manager’s database
and sends a command (DYNASTAM) to the NetView autotask AUTO1. The NetView service macros used in this sample are DSICBS, DSILCS, and DSIMQS.

**AXITVN (CNMS4270)**
This sample is an XITVN DST exit. It provides the initial record for an empty VSAM data set. The NetView service macros used in this sample are DSICBS and DSIDATIM.

**CNMRECV (CNMS4289)**
This sample receives data buffers from the receiver buffer’s queue. The NetView service macro used in this sample is DSISYS.

**CNMSEND (CNMS4288)**
This sample sends a data buffer to a receiver using the NetView program-to-program interface.

**CNMSGENDA (CNMS4287)**
This sample sends a generic alert to the NetView using the NetView program-to-program interface.

**DSICTMOD (CNMS0055)**
This sample specifies a wide range of constants for NetView. You can specify timeout values for many functions such as hardware monitor, trace NCP command, view requests, and RUNCMDs. You can specify retry intervals for functions such as status collector, RTM, and WTORs. You can specify message queue thresholds, heap sizes, and sense code filtering. Refer to the sample module for a complete list of constants.

**DSIEX02A (CNMS4283)**
This sample is used to manipulate messages. The installation exit is invoked for standard output to the operator’s terminal. If DWO403I is the incoming message, it issues DSIMQS to start the task specified in DWO403I and swaps the message buffer to indicate that the task has been started and that the request can be reissued.

NetView service macros used in this sample are DSICBS, DSIDEL, DSIFRE, DSIGET, DSILCS, DSILOD, DSIMBS, DSIMQS, and DSIPRS.

**DSIEX17 (CNMS4297)**
This installation exit is invoked upon receipt of MVS messages and delete operator messages (DOMs) and is called before automation or ASSIGN processing.

CNMS4297 manipulates message IEF233A and its associated DOM. IEF233A is a held message displayed upon receipt of an MVS MOUNT request. Once the MOUNT request is honored, a DOM is issued to delete the held message.

The intent of this sample is to provide a logged message that includes the MOUNT request, the time that it was received, and the time that it was honored.

CNMS4297 also shows how to invoke DSIFREBS to delete a message so that it is not propagated for further processing.

NetView service macros used in this sample are:
- DSICBS
- DSIDATIM
- DSIFRE
- DSIGET
- DSILCS
Assembler Samples

DSIPRS
DSIVARS
DSIWLS

Assembler language called service routine DSIFREBF is also used in the sample.

DSIEX19 (CNMS4307)
This sample is an example of a RUNCMD security checking exit. This sample checks various commands that are embedded within a RUNCMD command. The checking is done using the DSICES and DSIKVS macros.

DSIEX21
This sample is an example of a DSITCPRF encryption installation exit. Refer to the Tivoli NetView for z/OS Security Reference for more information.

DSIRXPRM (CNMSJM11)
This sample contains the TSO/E REXX initialization parameters used by NetView for NetView REXX support. The sample includes the TSO/E REXX parameter block, module name table, subcommand table, and function package table. It also includes JCL, which must be modified, to assemble and link-edit the NetView REXX parameters into a NetView load library. The sample may be modified (except as noted in the sample) to suit the needs of your installation.

DSIUSR00 (CNMS4281)
This sample is an example of a user-defined message member. It is used in conjunction with the ABLDMSG sample command processor.

OPERID (CNMS4295)
This sample is an example of an automation table function (ATF). It can be called from a NetView automation table, and returns the operator ID of the NetView task that is automating a message or an MSU. See Writing an Automation Table Function” on page 297 for more information about ATFs.

Note: For more information about CNMRECV(CNMS4289), CNMSEND(CNMS4288), and CNMSGENA(CNMS4287), refer to the Tivoli NetView for z/OS Application Programmer’s Guide.
### Appendix B. MDB Field to AIFR Cross-Reference Table

This appendix provides a cross-reference table for MDB control block fields. The MDB fields are listed in control block order and cross-referenced to existing automation internal function request (AIFR) fields.

#### Table 54. MDB to AIFR Field Cross-Reference Table

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BUFHDR, IFRAUTO, or DSIAIFRO field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note:</strong> The following fields support full WTO attributes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBGMID</td>
<td>4-byte message ID field; decimal</td>
<td>GOJGMID</td>
</tr>
<tr>
<td>MDBGSYID</td>
<td>1-byte system ID; decimal</td>
<td>GOJGSYID</td>
</tr>
<tr>
<td>MDBSEQ</td>
<td>3-byte sequence number; decimal</td>
<td>GOJSEQ</td>
</tr>
<tr>
<td>MDBTIMH</td>
<td>8-character time in the format of: hh:mm:ss GOJGTIMH</td>
<td></td>
</tr>
<tr>
<td>MDBTIMT</td>
<td>3-character time in the format of: .th GOJGTIMT</td>
<td></td>
</tr>
<tr>
<td>MDBDSTP</td>
<td>7-character date stamp in the format of: yyyyddd GOJGDSTP</td>
<td></td>
</tr>
<tr>
<td>MDBMFLG(nn)</td>
<td>2-byte flags GOJGMFLG</td>
<td></td>
</tr>
<tr>
<td>MDBMFLG(1)</td>
<td>This is a DOM IFRAUDOM IFRAUWDO GOJGDOM</td>
<td></td>
</tr>
<tr>
<td>MDBDOM</td>
<td>Sound processor alarm GOJGALRM</td>
<td></td>
</tr>
<tr>
<td>MDBMFLG(3)</td>
<td>Hold message until DOM or otherwise deleted GOJG HOLD</td>
<td></td>
</tr>
<tr>
<td>MDBFGPA</td>
<td>4 characters of foreground presentation attributes GOJFGPA</td>
<td></td>
</tr>
<tr>
<td>MDBFGPA(1)</td>
<td>Foreground control field GOJFGCON</td>
<td></td>
</tr>
<tr>
<td>MDBFGPA(2)</td>
<td>Foreground color field GOJF GCOL</td>
<td></td>
</tr>
<tr>
<td>MDBFGPA(3)</td>
<td>Foreground highlighting GOJG FHIL</td>
<td></td>
</tr>
<tr>
<td>MDBFGPA(4)</td>
<td>Foreground intensity GOJGFINT</td>
<td></td>
</tr>
<tr>
<td>MDBBGPA</td>
<td>4 characters of background presentation attributes GOJGBGA</td>
<td></td>
</tr>
<tr>
<td>MDBBGPA(1)</td>
<td>Background control field GOJGBCON</td>
<td></td>
</tr>
<tr>
<td>MDBBGPA(2)</td>
<td>Background color field GOJGBCOL</td>
<td></td>
</tr>
<tr>
<td>MDBBGPA(3)</td>
<td>Background highlighting GOJGBHIL</td>
<td></td>
</tr>
<tr>
<td>MDBBGPA(4)</td>
<td>Background intensity GOJGBINT</td>
<td></td>
</tr>
<tr>
<td>MDBOSNM</td>
<td>Originating system name IFRAUWSN GOJGOSNM</td>
<td></td>
</tr>
</tbody>
</table>
### MDB Field to AIFR Cross-Reference Table

Table 54. MDB to AIFR Field Cross-Reference Table  (continued)

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BUFHDR, IFRAUTO, or DSIAIFRO field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDBCJOBN</td>
<td>Job name</td>
<td>IFRAUWJA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOCOJOBN</td>
</tr>
</tbody>
</table>
| MDBCPROD                | 16-byte SCP product level  
4-character MVS CP object version level  
4-character control program name  
8-character FMID of originating system | CPOCPROD |
| MDBCERC                 | 128 bits routing codes | IFRAUWRT |
|                         |             | CPOCERC                           |
| MDBCDESC                | 2-byte descriptor codes | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCA                 | System failure | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCB                 | Immediate action required | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCC                 | Eventual action required | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCD                 | System status | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCE                 | Immediate command response | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCF                 | Job status  | IFRAUWDS                          |
|                         |             | CPOCDESC                          |
| MDBESCG                 | Application program/processor DOM at end of task | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCH                 | Out-of-line | IFRAUWDS                          |
|                         |             | CPOCDESC                          |
| MDBESCI                 | Operator’s request | IFRAUWDS |
|                         |             | IFRAUMCS(3)                       |
|                         |             | CPOCDESC                          |
| MDBESCJ                 | Track command response | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCK                 | Critical eventual action | IFRAUWDS |
|                         |             | CPOCDESC                          |
| MDBESCL                 | Delivered but not held | IFRAUWDS |
|                         |             | CPOCDESC                          |
### MDB Field to AIFR Cross-Reference Table

#### Table 54. MDB to AIFR Field Cross-Reference Table (continued)

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BUFHDR, IFRAUTO, or DSIAIFRO Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDBDESCM</td>
<td>NetView automation table had opportunity to process this message before the WTO was issued</td>
<td>IFRAUWDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOCDESC</td>
</tr>
<tr>
<td>MDBDESCN</td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MDBDESCO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBDESCCP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMVL</td>
<td>Message level flags</td>
<td>CPOCMLVL</td>
</tr>
<tr>
<td>MDBCMVL(1) MDBMLR</td>
<td>WTO</td>
<td>IFRAUWWR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMLR</td>
</tr>
<tr>
<td>MDBCMVL(2) MDBMLIA</td>
<td>Immediate action</td>
<td>IFRAUWDS(2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMLIA</td>
</tr>
<tr>
<td>MDBCMVL(3) MDBMLCE</td>
<td>Critical eventual action</td>
<td>IFRAUWDS(11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMLCE</td>
</tr>
<tr>
<td>MDBCMVL(4) MDBMLE</td>
<td>Eventual action</td>
<td>IFRAUWDS(3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMLE</td>
</tr>
<tr>
<td>MDBCMVL(5) MDBMLI</td>
<td>Informational</td>
<td>CPOMLI</td>
</tr>
<tr>
<td>MDBCMVL(6) MDBMLBC</td>
<td>Broadcast</td>
<td>IFRAUWBD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUMCS(6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMLBC</td>
</tr>
<tr>
<td>MDBCMVL(7) MDBCMVL(8)</td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MDBCMVL(9) MDBCMVL(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMVL(11) MDBCMVL(12)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMVL(13) MDBCMVL(14)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMVL(15) MDBCMVL(16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCATTR</td>
<td>2-byte message attributes</td>
<td>CPOCATTR</td>
</tr>
<tr>
<td>MDBCATTR(1)</td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MDBCATTR(2) MDBCATTR(3)</td>
<td>Message is command response</td>
<td>IFRAUMCS(3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOCMCSC</td>
</tr>
<tr>
<td>MDBCATTR(4) MDBCATTR(5)</td>
<td>Message issued by authorized program</td>
<td>CPOCAUTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUPLS</td>
</tr>
<tr>
<td>MDBCATTR(6) MDBCATTR(7)</td>
<td>Message is to be retained by AMRF</td>
<td>CPOCRETN</td>
</tr>
</tbody>
</table>

Appendix B. MDB Field to AIFR Cross-Reference Table 291
### MDB Field to AIFR Cross-Reference Table

*Table 54. MDB to AIFR Field Cross-Reference Table (continued)*

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BufHdr, IfrAuto, or Dsiaifro Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDBCATTR(5)</td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MDBCATTR(6)</td>
<td>CPOCPRTY</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(7)</td>
<td>CPOCRETN</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(8)</td>
<td>IFRAUURET</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(9)</td>
<td>CPOCRRET</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(10)</td>
<td>IFRAURRT</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(11)</td>
<td>CPOCRNRT</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(12)</td>
<td>IFRAUNRT</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(13)</td>
<td>IFRAUASI</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(14)</td>
<td>IFRAUWAS</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(15)</td>
<td>CPOCASID</td>
<td></td>
</tr>
<tr>
<td>MDBCATTR(16)</td>
<td>CPOCTCB</td>
<td></td>
</tr>
<tr>
<td>MDBCPRTY</td>
<td>2-byte message priority; decimal</td>
<td>CPOCPRTY</td>
</tr>
<tr>
<td>MDBCRETN</td>
<td>AMRF retained message</td>
<td>CPOCRETN</td>
</tr>
<tr>
<td>MDBCRRET</td>
<td>Retain in AMRF</td>
<td>IFRAUURET</td>
</tr>
<tr>
<td>MDBCRNRT</td>
<td>Do no retain in AMRF</td>
<td>IFRAURRT</td>
</tr>
<tr>
<td>MDBCASID</td>
<td>ASID of issuer; decimal</td>
<td>IFRAUASI</td>
</tr>
<tr>
<td>MDBCTCB</td>
<td>TCB address of issuer</td>
<td>IFRAUTCB</td>
</tr>
<tr>
<td></td>
<td>4-byte hexadecimal</td>
<td>IFRAUWJT</td>
</tr>
<tr>
<td>MDBCTOKN</td>
<td>4-byte DOM token associated with message; decimal</td>
<td>CPOCTOKN</td>
</tr>
<tr>
<td>MDBCYSYID</td>
<td>1-byte system ID (for DOM); decimal</td>
<td>CPOCSYID</td>
</tr>
<tr>
<td>MBDDOMFL</td>
<td>1-byte DOM flags</td>
<td>CPODOMFL</td>
</tr>
<tr>
<td>MBDDOMFL(1)</td>
<td>DOM by message ID</td>
<td>MSGDOMAT</td>
</tr>
<tr>
<td>MDBDMSGI</td>
<td>MSGDOMAT</td>
<td>IFRAUWDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUWDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMSGI</td>
</tr>
<tr>
<td>MBDDOMFL(2)</td>
<td>DOM by system ID</td>
<td>CPODSYSI</td>
</tr>
<tr>
<td>MDBDSYSI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBDDOMFL(3)</td>
<td>DOM by ASID</td>
<td>IFRAUWDT</td>
</tr>
<tr>
<td>MDBDASID</td>
<td>IFRAUWDT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUWDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPODASID</td>
</tr>
</tbody>
</table>
### Table 54. MDB to AIFR Field Cross-Reference Table (continued)

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BUFHDR, IFRAUTO, or DSIAIFRO field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDBDOMFL(4) MDBDJTCB</td>
<td>DOM by job step TCB</td>
<td>IFRAUWDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUWDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPODJTCB</td>
</tr>
<tr>
<td>MDBDOMFL(5) MDBDTOKN</td>
<td>DOM by token</td>
<td>IFRAUWDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUWDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDBDTOKN</td>
</tr>
<tr>
<td>MDBCMISC</td>
<td>1-byte miscellaneous routing information</td>
<td>CPOCMISC</td>
</tr>
<tr>
<td>MDBCMISC(1) MDBCUD</td>
<td>Display UD messages</td>
<td>CPOCUUD</td>
</tr>
<tr>
<td>MDBCMISC(2) MDBCFCUDO</td>
<td>Display only UD messages</td>
<td>CPOCFUDO</td>
</tr>
<tr>
<td>MDBCMISC(3) MDBCFCIDO</td>
<td>Queue by ID only</td>
<td>CPOCFIDO</td>
</tr>
<tr>
<td>MDBCIOJID</td>
<td>8-character originating job ID</td>
<td>IFRAUWJU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOCIOJID</td>
</tr>
<tr>
<td>MDBCKEY</td>
<td>8-byte key associated with message</td>
<td>CPOCKEY</td>
</tr>
<tr>
<td></td>
<td>8 bytes of hexadecimal or character value</td>
<td></td>
</tr>
<tr>
<td>MDBCAUTO</td>
<td>8-byte MPF automation token</td>
<td>CPOCAUTO</td>
</tr>
<tr>
<td></td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>MDBCCART</td>
<td>8-byte command and response token</td>
<td>CPOCCART</td>
</tr>
<tr>
<td></td>
<td>8 bytes of hexadecimal or character value</td>
<td></td>
</tr>
<tr>
<td>MDBCCNID</td>
<td>MVS target console ID number (4 byte); decimal</td>
<td>CPOCCNID</td>
</tr>
<tr>
<td></td>
<td>Use CONVCON to find 8-character console name, save in IFRAUCON</td>
<td>IFRAUCON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFRAUWUC</td>
</tr>
<tr>
<td>MDBCMGMT</td>
<td>16-bit message type</td>
<td>CPOCMGMT</td>
</tr>
<tr>
<td>MDBCMSGT</td>
<td>Display job names</td>
<td>IFRAUWF1(9)</td>
</tr>
<tr>
<td>MDBCMSGT(1) MDBCMSGTA</td>
<td>Display status</td>
<td>IFRAUWF1(10)</td>
</tr>
<tr>
<td>MDBCMSGT(2) MDBCMSGTB</td>
<td>Monitor active</td>
<td>CPOMSGTC</td>
</tr>
<tr>
<td>MDBCMSGT(3) MDBCMSGTC</td>
<td>Monitor active</td>
<td>CPOMSGTC</td>
</tr>
<tr>
<td>MDBCMSGT(4) MDBCMSGTD</td>
<td>Indicates existence of QID field in WPL (AOS/1)</td>
<td>CPOMSGTD</td>
</tr>
<tr>
<td>MDBCMSGT(5)</td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MDBCMSGT(6) MDBCMSGTF</td>
<td>Monitor SESS</td>
<td>IFRAUWF1(14)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPOMSGTF</td>
</tr>
</tbody>
</table>
### MDB Field to AIFR Cross-Reference Table

**Table 54. MDB to AIFR Field Cross-Reference Table (continued)**

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BUFHDR, IFRAUTO, or DSIAIFRO field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDBCMSGT(7)</td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MDBCMSGT(8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(11)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(12)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(13)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(14)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(15)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCMSGT(16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDBCRPYL</td>
<td>2-byte reply ID length; decimal CPOCRPYL</td>
<td></td>
</tr>
<tr>
<td>MDBCRPYI</td>
<td>8-character reply ID</td>
<td>CPOCRPYI</td>
</tr>
<tr>
<td>MDBCTOFF</td>
<td>Offset in the message text field of the beginning of the message</td>
<td>CPOCTOFF</td>
</tr>
<tr>
<td></td>
<td>(Internal field, no external variable.)</td>
<td></td>
</tr>
<tr>
<td>MDBCRPYB</td>
<td>4-byte binary reply ID</td>
<td>CPOCRPYB</td>
</tr>
<tr>
<td>MDBCLCNT</td>
<td>2-byte count of number of lines in message; decimal</td>
<td>CPOCLCNT</td>
</tr>
<tr>
<td></td>
<td>Note: CPOCLCNT and MDBCLCNT are not supported by the NetView program. The</td>
<td></td>
</tr>
<tr>
<td></td>
<td>count of buffers on the IFRAUTBA chain is used instead. GETMSIZE provides</td>
<td></td>
</tr>
<tr>
<td></td>
<td>this function.</td>
<td></td>
</tr>
<tr>
<td>MDBCJOJBN</td>
<td>8-character originating job name</td>
<td>CPOCOJBN</td>
</tr>
<tr>
<td>MDBTLEN</td>
<td>2-byte text object length</td>
<td>HDRTLLEN</td>
</tr>
<tr>
<td>MBDTTYPE</td>
<td>2-byte text object type flags</td>
<td>HDRLNNTYP in each data buffer</td>
</tr>
<tr>
<td></td>
<td>Control text</td>
<td>HDRLNCT</td>
</tr>
<tr>
<td></td>
<td>Label text</td>
<td>HDRLNLBL</td>
</tr>
<tr>
<td></td>
<td>Data text</td>
<td>HDRLNDAT</td>
</tr>
<tr>
<td></td>
<td>End text</td>
<td>HDRLNEND</td>
</tr>
<tr>
<td></td>
<td>Prompt text</td>
<td>HDRTPROT</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>None</td>
</tr>
<tr>
<td>MBDTTYPE(16)</td>
<td>Text object presentation field overrides general object presentation</td>
<td>HDRTFPAF</td>
</tr>
<tr>
<td>MBDTFPAF</td>
<td>attribute field</td>
<td></td>
</tr>
<tr>
<td>MBDTTPA</td>
<td>4-byte presentation attributes</td>
<td>HDRTTTPA</td>
</tr>
</tbody>
</table>
### Table 54. MDB to AIFR Field Cross-Reference Table (continued)

<table>
<thead>
<tr>
<th>MDB Control Block Field</th>
<th>Description</th>
<th>BUFHDR, IFRAUTO, or DSIAIFRO field</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDBTMTPA (1)</td>
<td>Presentation control</td>
<td>HDRTPCON</td>
</tr>
<tr>
<td>MDBTMTPA (2)</td>
<td>Presentation color</td>
<td>HDRTPCOL</td>
</tr>
<tr>
<td>MDBTMTPA (3)</td>
<td>Presentation highlighting</td>
<td>HDRTPHIL</td>
</tr>
<tr>
<td>MDBTMTPA (4)</td>
<td>Presentation intensity</td>
<td>HDRTPINT</td>
</tr>
<tr>
<td>MDBTMSGT</td>
<td>Variable length message text</td>
<td>Message text is in buffers chained from IFRAUTBA and IFRAUTBL</td>
</tr>
</tbody>
</table>

**Note:** The following fields and flags from WQE are not mapped by MDB, are set to zero (0), and generally apply to how the write-to-operator switch virtual circuit (WTO SVC) was issued and not to what the message is about.

- First message of MLWTO; can be inferred from IFRAUTBA chain for each buffer on the chain
  - IFRAUWFR (Not recommended)
- Middle message of MLWTO; can be inferred from IFRAUTBA chain for each buffer on the chain
  - IFRAUWMD (Not recommended)
- Last message of MLWTO; can be inferred from IFRAUTBA chain for each buffer on the chain
  - IFRAUWLS (Not recommended)
- Single message line; can be inferred from IFRAUTBA chain for each buffer on the chain
  - IFRAUWSI (Not recommended)
- Suppressed message; bit is always set to zero (0)
  - IFRAUWSP
- Routing and Descriptor codes exist; inferred from other data
  - IFRAUMCS(1)
- Queue conditionally to REG0 console; bit is always set to zero (0)
  - IFRAUMCS(2)
- Message type flag field exists; can be inferred from other data
  - IFRAUMCS(4)
- Message is reply to WTOR; bit is always set to zero (0)
  - IFRAUMCS(5)
- Queue to hardcopy only; bit is always set to zero (0)
  - IFRAUMCS(7)
- Queue unconditionally to console in REG0; bit is always set to zero (0)
  - IFRAUMCS(8)
- No time stamp; bit is always set to zero (0)
  - IFRAUMCS(9)
- Do not log minor WQEs; bit is always set to zero (0)
  - IFRAUMCS(11)
- Extended WPL exists; bit is always set to zero (0)
  - IFRAUMCS(12)
- Bypass queue to hardcopy; bit is always set to zero (0)
  - IFRAUMCS(14)
- WQEBLK keyword specified; bit is always set to zero (0)
  - IFRAUMCS(15)

**Note:**
1. Fields indicated as “inferred” mean that fields from earlier releases of NetView must be set by testing the values of other fields. For example, if at least one route code is nonzero, set the “route-codes included” flag on.
Appendix C. Writing an Automation Table Function

This appendix describes how to design, code and install an assembler language automation table function (ATF) for NetView. You can write an ATF to evaluate a complex condition and return a value to automation table processing. An ATF is useful for gaining access to message, management services unit (MSU), or MSU hierarchy information that cannot be accessed directly from the automation table.

Overview of Automation Table Function

An ATF conditional performs a module call from the automation table. An ATF module has the option of returning data to be compared against a value in the automation table statement to make synchronous automation decisions. Data returned by an ATF module can also be passed by the automation table to a command or a command list. The ATF conditional can be contained in IF-THEN automation statements.

The syntax of an ATF conditional is:

\[
\text{ATF} \left( \text{BIT 'modname parameterlist'} \right) = \text{value}
\]

Where:

BIT
- Specifies that the automation table is bit data.

modname
- Specifies a 1- to 8-character load module name.

parameterlist
- Specifies the parameter list to be passed to modname.

value
- Specifies the value that the automation table uses to compare with the value returned by modname, if any.

The ATF module gets control only if the automation table scans to that statement. Because performance can be affected negatively, IF-THEN statements that contain ATF module calls cannot be executed with each scan of the automation table. Place these IF-THEN statements within BEGIN-END sections to prevent unnecessary calls.

ATF resembles a function as defined in REXX and, from the automation table language perspective, ATF is a function that calls functions.

Note: Refer to the Tivoli NetView for z/OS Automation Guide for more information about the IF-THEN statement and ATF syntax.
Designing and Coding an ATF Module

ATF modules must adhere to the guidelines for user-written programming described in "General Coding Guidelines" on page 6. In addition, ATF modules must conform to the special requirements described in this section.

Note: A sample ATF module OPERID (CNMS4295) is provided with NetView. You can use it as pattern when coding your own ATFs.

Input to and Output from the ATF

When the ATF module gains control, the registers contain the following information:

0  Unspecified
1  The address of an 8-byte parameter list that contains:
   1. A 4-byte address of a command work block (CWB)
      The CWB contains the following:
      • A user save area (CWBSAVEA) that is the ATF 72-byte save area.
      • The address of the command buffer (CWBBUF) in which NetView loads the command string. The buffer has a standard BUFHDR.
      • The address of a service work block (CWBSWB) for calling service facilities.
      • The address of the task information block (CWBTIB).
      • A work area (CWBADATD) that is the ATF 256-byte temporary storage for keeping variables while remaining reentrant.
   2. The 4-byte address of the AIFR being automated
13  The address of a standard 72-byte save area used to store the caller's registers.
14  The return address
15  The entry address of the ATF module
2-12 Unspecified

The AIFR being automated can contain either a message or an MSU buffer. See Figure 30 on page 300 for an example of the parameter list and of control blocks and their relevant pointers.

For bit-string functions, the first byte of the returned value from ATF must contain the first bit needed for the evaluation. The bit-string on the right side of the equal sign specifies X for any bit positions whose value is irrelevant to the evaluation.

For example, if ATF needs to determine whether or not the third and fourth bits of a byte are B'01', you can code this as:

IF ATF (BIT 'pname ....') = 'XX01' THEN ....;

You can also write the pname to shift the bits so that leading Xs are not needed. This does not affect NetView as long as the pname design is coordinated with the automation entry.

When NetView regains control, the registers and their contents are as follows:

15  A return code:
   0  Normal
The ATF module replaces the command string, found in the buffer pointed to by CWBBUF, with the value to be compared to the right side of the equal sign on the automation table IF-THEN statement. The maximum size of the returned data is limited to 256 bytes minus the length of the BUFHDR. The ATF places the actual length of the returned data in HDRMLENG. If you put zero in HDRMLENG, a null value is returned.

If you need to extract more data, you can use multiple ATF invocations from one automation table statement.

Do not return a swapped buffer, because the buffer that NetView provides is the maximum size supported.

ATF return codes 1–8 will cause a false ATF comparison. Return codes greater than 8 can also cause a false ATF comparison. Error message CNM588E is issued for return codes greater than 8. In an operator station task, the error message is sent to the operator. In a data services task (DST) or optional subtask (OPT), the error message is sent to the operator that started the DST or OPT, if one exists. Otherwise, it is sent to the authorized receiver. You see the return code only if it is greater than 8 (which results in an error message containing the return code).

Notes:
1. Do not pass a return code greater than 8 unless you want to indicate a severe error from the ATF module.
2. Freeing the buffer passed to ATF causes an abend.

Control Blocks
An ATF module has access to the command buffer and these control blocks:
- AIFR
- DSICWB
- DSISWB
- DSITVB
- DSITIB
- DSMVT
- DSISVL
Installing an ATF module

Link-edit the ATF load module into the NetView load library. Code an automation table IF-THEN statement specifying your ATF module name and other required information. Refer to the *Tivoli NetView for z/OS Automation Guide* for information about coding an IF-THEN statement. Use the AUTOTBL command to activate your automation table. Refer to the NetView online help for information about the AUTOTBL command. ATF modules coded in the automation table are loaded when the automation table is activated by the AUTOTBL command. You do not need to recycle Netview to be able to use an ATF load module once it has been added to the NetView load library.

You can add new ATF modules (using link-editing) and activate them (using the AUTOTBL command) while NetView is running, even if automation was previously in effect. To change an existing ATF module that is already loaded, you have four alternatives:
Writing an Automation Table Function

1. Rename the ATF module using link-editing, change the corresponding ATF name or names in the automation table, and reissue the AUTOTBL command.

2. Stop NetView and then restart it.

3. Inactivate the ATF using the AUTOTBL OFF command. Wait for message CNM593I to be issued to ensure all existing ATFs have been deleted, then reissue AUTOTBL to put the ATFs back into effect.

4. If it is not practical to shut automation off, issue the AUTOTBL command with a substitute member. Wait for the current automation process to finish using the ATF. Run a command processor to delete one or more ATFs, and reissue AUTOTBL with the main member.

When AUTOTBL command processing ends abnormally because of a STOP FORCE or a RESET IMMED, all of the ATFs may not be deleted, even if message CNM593I is received. If one of these situations exists, you can run a command processor to delete one or more ATFs before reissuing AUTOTBL to make sure the old ATFs of the same name that you are replacing are deleted.

Points to Remember

When you write ATFs, remember the following points:

- Write ATFs in assembler.
- Do not define ATFs in DSICMD.
- ATFs cannot be invoked as commands.
- ATFs are different from NetView commands. Do not invoke ATFs outside the automation table. Do not code NetView commands as ATFs.
- Define an ATF as being a reentrant load module (by the name to be referenced in automation statements) stored in a NetView load library defined by the NetView PROC, which is similar to a command processor.
Writing an Automation Table Function
## Appendix D. Assembler Macros and HLL Service Routine Interfaces for NetView

Table 55 lists each assembler macro or service routine and its corresponding high-level language (HLL) service routine name. "Chapter 8. Macros" on page 161 contains detailed information on each assembler macro.

**Note:** For more information about the HLL service routines, refer to [Tivoli NetView for z/OS Customization: Using PL/I and C](#).

### Table 55. Assembler and HLL Service Interfaces for NetView

<table>
<thead>
<tr>
<th>Assembler Macro</th>
<th>HLL Service Routine Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIAUTO</td>
<td>CNMAUTO</td>
<td>Invoke automation services</td>
</tr>
<tr>
<td>DSIBAM</td>
<td>–</td>
<td>Build automation message</td>
</tr>
<tr>
<td>DSIBAMKW</td>
<td>–</td>
<td>Build automation message keyword</td>
</tr>
<tr>
<td>DSICBS</td>
<td>–</td>
<td>Control block services</td>
</tr>
<tr>
<td>DSICES</td>
<td>CNMSCOP</td>
<td>Command entry services</td>
</tr>
<tr>
<td>DSICVTHE</td>
<td>–</td>
<td>Convert to hex</td>
</tr>
<tr>
<td>DSIC2T</td>
<td>CNMC2T</td>
<td>Code point translation service reference</td>
</tr>
<tr>
<td>DSIDATIM</td>
<td>CNMINFC</td>
<td>Date and time</td>
</tr>
<tr>
<td>DSIDEL</td>
<td>–</td>
<td>Delete user-defined module</td>
</tr>
<tr>
<td>DSIDKS</td>
<td>CNMMEM(C</td>
<td>O</td>
</tr>
<tr>
<td>DSIFIND</td>
<td>CNMNAMS</td>
<td>Find long-running command storage</td>
</tr>
<tr>
<td>DSIFRE</td>
<td>CNMPOOL</td>
<td>Free storage</td>
</tr>
<tr>
<td>DSIFREBF*</td>
<td>–</td>
<td>Free buffers service</td>
</tr>
<tr>
<td>DSIGET</td>
<td>CNMPOOL</td>
<td>Get storage</td>
</tr>
<tr>
<td>DSIGETDS</td>
<td>CNMGETD</td>
<td>Data queue manipulation service</td>
</tr>
<tr>
<td>DSIHREGS</td>
<td>CNMHRGS</td>
<td>High-performance transport application registration</td>
</tr>
<tr>
<td>DSISHNDS</td>
<td>CNMHSMU</td>
<td>Send high-performance message unit</td>
</tr>
<tr>
<td>DSIID</td>
<td>–</td>
<td>Inserts an readable identifier in the code</td>
</tr>
<tr>
<td>DSIKVS</td>
<td>CNMSCOP</td>
<td>Keyword and or value services</td>
</tr>
<tr>
<td>DSILCS</td>
<td>–</td>
<td>Obtain and or release control blocks</td>
</tr>
<tr>
<td>DSILOD</td>
<td>–</td>
<td>Load user-defined module</td>
</tr>
<tr>
<td>DSIMBS</td>
<td>–</td>
<td>Message build services</td>
</tr>
<tr>
<td>DSIMDS</td>
<td>–</td>
<td>Message definition services</td>
</tr>
<tr>
<td>DSIMMDB*</td>
<td>CNMPMDB</td>
<td>Process MDB service</td>
</tr>
<tr>
<td>DSIMQS</td>
<td>CNMSMSG</td>
<td>Message queueing services</td>
</tr>
<tr>
<td>DSINOR</td>
<td>CNMQAPI</td>
<td>Resource Object Data Manager</td>
</tr>
<tr>
<td>DSIPAS</td>
<td>–</td>
<td>Parameter alias services</td>
</tr>
<tr>
<td>DSIPOP</td>
<td>CNMNAMS</td>
<td>Remove long-running command</td>
</tr>
<tr>
<td>DSIPOS</td>
<td>–</td>
<td>ECB post services</td>
</tr>
</tbody>
</table>
### Assembler Macros and HLL Service Routine Interfaces

Table 55. Assembler and HLL Service Interfaces for NetView (continued)

<table>
<thead>
<tr>
<th>Assembler Macro</th>
<th>HLL Service Routine Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIPRS</td>
<td>CNMSCAN</td>
<td>Parsing services</td>
</tr>
<tr>
<td>DSIPSS</td>
<td>CNMSMSG</td>
<td>Presentation services</td>
</tr>
<tr>
<td>DSIPUSH</td>
<td>CNMNAMS</td>
<td>Establish long-running command</td>
</tr>
<tr>
<td>DSIQOS</td>
<td>–</td>
<td>Query operator services</td>
</tr>
<tr>
<td>DSIQRS</td>
<td>–</td>
<td>Query resource services</td>
</tr>
<tr>
<td>DSIRDS</td>
<td>–</td>
<td>Resource definition services</td>
</tr>
<tr>
<td>DSIRXCOM</td>
<td>–</td>
<td>Access REXX variables (VM only)</td>
</tr>
<tr>
<td>DSIRXEBS</td>
<td>–</td>
<td>Get an EVALBLOK</td>
</tr>
<tr>
<td>DSISRCMV</td>
<td>–</td>
<td>Search for subvector/subfield</td>
</tr>
<tr>
<td>DSISYS</td>
<td>CNMINFC</td>
<td>Operating system indicator</td>
</tr>
<tr>
<td>DSITECBS</td>
<td>–</td>
<td>Manage a dynamic ECB list for DSTs</td>
</tr>
<tr>
<td>DSIVARS</td>
<td>CNMVARS</td>
<td>NetView variable services</td>
</tr>
<tr>
<td>DSIWAT</td>
<td>–</td>
<td>ECB wait services</td>
</tr>
<tr>
<td>DSIWCS</td>
<td>CNMSMSG</td>
<td>Write console services</td>
</tr>
<tr>
<td>DSIWLS</td>
<td>CNMSMSG</td>
<td>Write log services</td>
</tr>
<tr>
<td>DSIZCSMS</td>
<td>CNMCNMI</td>
<td>CNM data services</td>
</tr>
<tr>
<td>DSIZVSMSS</td>
<td>CNMKIO</td>
<td>VSAM data services</td>
</tr>
<tr>
<td>DS16REGS</td>
<td>CNMRGS</td>
<td>Registration API</td>
</tr>
<tr>
<td>DS16SNDS</td>
<td>CNMSMU</td>
<td>Send a message unit</td>
</tr>
<tr>
<td>DUIFSMTE</td>
<td>–</td>
<td>Defines a status mapping table entry</td>
</tr>
</tbody>
</table>

**Note:** * = Assembler service routine.
control blocks (continued)
DSISVL (service routine vector list) 149
DSISWB (service work block) 149
DSITECBR (branch table of ECB)
processor load modules) 150
DSITIB (task information block) 150
DSITVB (task vector block) 152
DSIUSE (installation exit parameter list) 156
how to establish addressability 11
how to include 9
installation exit routine 9
optional subtask 85
overview 109
user subtasks 9
user-written programming 10
work block service 11
control point-management service unit
(CP-MSU) 54, 260
converting, hexadecimal DSICVTHE macro 169
CP-MSU (control point management service unit) 54, 260
cross-domain
call send 25, 38
ECB 68
input 32
tasks 19, 23
Customer Support xi

D

data queue manipulation service
(DSIGETDS) macro 182
data service
command processor (DSCP)
coding instructions 92
description 62
example of design 97
example program 99
solicited 93
structure 98
unsolicited 93
VSAM service interface 95
request block (DSRB)
call send 122
DSCP 92
major, minor return codes 94, 96
data service (DSIZCMSG) macro,
CNM 251
data service block 122
data transport request (DSIDTR) control block 124
date and time (DSIDATIM) macro 171
defining messages, disk 206
delete user-defined module (DSIDEL) macro 172
delimiters 220
destination application name (DAN) 194, 270
disability information xi
disk service (DSIDKS) macro 20, 172
display flexibility 3
dSCP redrive 94
dSIO90I message 60

DSI6REGS (LU 6.2 registration) macro 258
DSI6NSDS (send a message unit) macro 264
DSIAIFRO control block 116
DSIAUTO (invoke automation service) macro 161
DSIBAM (build automation message) macro 162
DSIBAMKW (build automation message keyword) macro 164
DSICT2 (code-to-text) macro 169
DSICBH (control block header) 120
DSICBS (control block service) macro
description and syntax 165
eample 10
DSICES (command entry service) macro
calling a command directly 18
description and syntax 166
the ROLL function 75
DSICLD data set 172
DSICMD macro 61
DSICTMOD macro 86
DSICTMOD sample 287
DSICVTHE macro, converting to hexadecimal 169
DSICW (command work block) fields 121
input to command processor 63
overview 121
DSIDATIM (date and time) macro 171
DSIDEL (delete user-defined module) macro 172
DSIKS (disk service) macro
buffer head use 116
description and syntax 172
DSIDSB use 22
example 20, 86
DSIDSG (data service block) control block 122
DSIDSRB (data service request block) control block
description 122
fields 122
major, minor return code 94, 96
DSIDTR (data transport request) control block 124
DSIELB (external logging block) control block 124
DSIELTSK (external logging task) 16
DSIEX01 installation exit 25, 32
DSIEX02 installation exit (obsolete) 25, 33
DSIEX02A installation exit
DSIPSS macro 227
interface with DSIX16 45
message handling, OST/NNT 29
message handling, PPT 30
output, operator installation exit 33
summary 25
title-line mode 14
DSIEX03 installation exit 25, 35
DSIEX04 installation exit 25, 36
DSIEX05 installation exit 25, 37
DSIEX06 installation exit 25, 30, 38
DSIEX07 installation exit 25, 38
DSIEX08 installation exit (obsolete) 25
DST (data service task)
concatenated installation exit routine 25, 29
description 5
installation exit routine 25, 53
unique service 24
writing user subtasks
initialization 27, 57, 91
installation 91
overview 90
processing 92
return codes 92
DSTINIT statement 91
DUIFSMTE (customize DisplayStatus table) macro 272
e-mail contact xi
ECB (event control block)
loop 86
post service 20, 219
wait service 20, 247
entry specifications 102
environments, installation exit 25
Erase/Write Alternate command 68
Erase/Write command 68
error recovery 63
establish operator validity and logon status 235
evaluation block (EVALBLOK) 102, 103
event routine 229
exit 16 interface data 146
extension, XITST 156
external function parameter list 102
external logging 16, 27
external logging block (DSIELB) 124
feedback about publications xi
find long-running command storage (DSIFIND) macro 175
focal point transfer RU header 55
free storage (DSIFRE) macro 177
FREEMAIN failure 179
full-screen command processor
coding instruction 67
screen formatting 68
FUNCT keyword 91
function package directory
entry format 105
example 106
header 104
writing 101
G
general data stream (GDS) 260
get an EVALBLOK (DSIRXEBS) macro 238
get storage (DSIGET) macro 180
getting and freeing storage
freeing 177
getting 180
named 22
getting and freeing storage (continued)
non-queued 12
queued 12
global installation exit routine 25, 32
global variable service (DSIVARS) macro 243
H
hardcopy log 249
HCT (hardcopy task) 5
HDRIND field values 112
HDRMTYPE field values 112
high-performance, send message unit (DSIHSDS) macro 189
high-performance transport application registration (DSIHREGS) macro 184
HIGH priority 211
high-priority command processor 61
HLL service routine, corresponding macros 303
I
IFRCODAI buffer 29, 33, 44
IFRCODCR buffer 19, 98
IFRCODUS message buffer 88
immediate command processor 61
immediate message 226
information, accessibility xii
information, disability xii
INIT keyword 85, 91
initialization member 84
input, command processor 63
installation exit buffer for RUNCMD 160
installation exit environments 25
installation exit parameter list (DSIUSE) control block 156
installation exits
DST (data service task) 25
global 25
list 25
dx 31
control blocks 31
input 27
installing 60
output 28
overview 25
template for 60
writing 27
summary 31
unused 60
installing command processor 78
internal function request (DSIFR) control block 125
intertask communication 86
invoke automation service (DSIAUTO) macro 161
invoking commands 17
IRB exits, special requirements for 90
IRXEVALB mapping macro 103
ISTMGC00 default table 93
K
keyboard, shortcut keys xii
keyword/value service (DSIKVS) macro 195
L
LIST command 90
load failure 60
load user-defined module (DSILOD) macro 200
log input, output 69
log output installation exit 25, 36
logoff, before 25, 42
LOGOFF routine 77, 92
logon validation 25, 41
long-running command (DSIPUSH) macro 229
long-running command processor
completion codes 72
description 62
scheduling RESUME routine 71
writing 70
LOW priority 211
LU 6.2 registration (DSI6REGS) macro 258
M
MACRF keyword 95
macro use 7
macros
DSI6REGS (LU 6.2 Registration) 258
DSI6SNDS (send a message unit) 264
DSIAUTO (invoke automation service) 161
DSIBAM (build automation message) 162
DSIBAMKW (build automation message keyword) 164
DSICT2 (code-to-text) 169
DSICBS (control block service) 165
DSICES (command entry service) 166
DSICVTHE, hexadecimal 169
DSIDATIM (date and time) 171
DSIDEL (delete user-defined module) 172
DSIDKS (disk service) 172
DSIFIND (find long-running command storage) 175
DSIFRE (free storage) 177
DSIFREBS (free storage) 180
DSIGET (get storage) 180
DSIGETDS (data queue manipulation service) 182
DSIHREGS (high-performance transport application registration) 184
DSIHSDS (sending high-performance message unit) 189
DSIID (store sysmod level, csect) 195
DSIKVS (keyword/value service) 195
Q
query support 68
queued storage 12, 179, 180

R
redrive, DSCP 94
reentrant code 8
register use 8
regular command 32, 35
releasing queued storage 89
remove long-running command (DSIPOP) macro 217
reset unit header, focal point transfer 55
RESET command 73
residency mode (RMODE) 8
resource definition service (DSIRDS) macro 238
resource object data manager (DSINOR) macro 213
resource status manager installation exit DSIUSE control block 156
extension for XITST 156
summary 25
Restructured Extended Executor (REXX) language
writing user function 101
resumable command 229
RESUME routine
completion codes 72
DSIPUSH macro 217
general 71
issuing a message 73
return code 72
ROLL
command 62
full-screen function 75
function 75
group 71, 75
ROUTE command 20
routing and targeting instruction (R&TI)
APPL field, DSI6REGS macro 260
return code, DSI6SNDS macro 270, 271
used in DSi6REGS macro 263
with DSi6NDS macro 264
RUNCMDC, installation exit buffer control block 160
samples (continued)
AHEREGR (CNMS4296) 285
AHSNDMU (CNMS4286) 285
ALERTMSG (CNMS4284) 285
ALISTMEM (CNMS4276) 285
AMLWTO (CNMS4273) 285
AMSGMOD (CNMS4271) 285
A0OPTTSK (CNMS4277) 285, 286
APSFULL (CNMS4279) 286
AREGISTR (CNMS4292) 286
ARODMCON (CNMS4290) 286
ASENDMU (CNMS4293) 286
ASEQLOG (CNMS4275) 286
ATMPCMDP (CNMS4202) 78, 286
ATMPUXIT (CNMS4282) 286
AUSRFUNC (CNMS8002) 286
AWRLOG (CNMS4272) 286
AXITCM (CNMS4504) 286
AXITST (CNMS4285) 286
AXITVNX (CNMS4270) 287
CNMRECV (CNMS4289) 287
CNMSENDO (CNMS4288) 287
CNMSGENA (CNMS4287) 287
DSICTMOD (CNMS4055) 287
DSIEX19 (CNMS4307) 288
DSIEX21 288
DSIRXPME (CNMS5141) 288
DSIUSR00 (CNMS4281) 288
OPERID (CNMS4295) 288
overview 283
sequential logging 16
table of assembler 283
S
samples
AAUTOTB (CNMS4291) 284
ABLDMSG (CNMS4278) 285
ACALLCMD (CNMS4280) 285
ADATTIM (CNMS4274) 285
AGETDS (CNMS4294) 285
samples (continued)
SPASS keyword 95
special data handling 3
STIFLE message 74
STIMER macro 7
storage service 12
store symod level in csect (DSIID) 195
subtask organization 84
supervisor state 6
symbolic substitution 145
system command entry (DSISECE) control block 148
system command table (DSISCT) control block 149
system console
input from 25, 40
message display 16, 39
output to 25, 26, 39
T
task information block (DSITIB) control block 150
task priority, relative 84, 91
TASK statement 84
task vector block (DSITVB) control block description 152
fields 152
use attaching the OPT subtask 85
intertask communication 152
template
command procedure processor 78
optional task 285, 286
terminal input, simulating 18
title-line output 15
Tivoli Customer Support 18
TPEND exit, VTAM 90
TRACE facility 3
translating code points 169
TRAP condition 29
TVBINXIT bit
coding restrictions 7
processing 6
TVBMEMNM field 86
U
unattended and MVS console operator task command considerations 62
unattended operator task 62
unbalanced quote 221
UNSOL keyword 93
unsolicited CNM data interface 93
user-defined command processor 88
user-defined service 100
user memory protection key 6
user subtasks, types
DST (data service task) 83
OPT (optional task) 83
user-written
function 101
programming
coding guidelines 6
testing 3
subtask 83
USERASIS return code 28, 32, 91
USERDROP return code
  deleting message and MSU 28
  initializing a DST 92
  installation exit 25
  installation exit output 28
  operator input 33
USEREVNT return code 56
USEREXLG return code 56
USERHCL return code 37
USERHCLR return code 37
USERLOG return code 37
USERLOGR return code 37
USERMSG buffer 27
USERSWAP return code 28, 33, 92

V
VALCLASS checking 196
variable name 7
variable service (DSIVARS) macro 243
VIEW command 3
virtual storage, loading and deleting module 20
Virtual Storage Access Method (VSAM)
data service (DSIZVSMS) macro 256
  empty file 27, 59
  I/O (input/output) service 256
  input 27, 58
  key 257
  output 59
  request 96
  service interface 25, 95
Virtual Telecommunications Access Method (VTAM)
  command invocation, before 25, 37
  solicited message 25, 38
  unsolicited message 25, 40

W
waiting completion, events 247
work block service 11
write console service (DSIWCS) macro 248
write log service (DSIWLS) macro 249

X
XITBN installation exit 25, 53
XITBO installation exit 25, 54
XITCI installation exit 25, 54
XITCO installation exit 25, 57
XITDI
  installation exit 25, 57
  keyword 91
XITST installation exit 156
XITVI
  installation exit 27, 58
  keyword 95
XITVN
  installation exit 27, 59
  keyword 95
XITVO
  installation exit 27, 59
  keyword 95
XITXL installation exit 27
