Tivoli NetView for z/OS Customization: Using PL/I and C

Copyright Notice

© Copyright IBM Corporation 1997, 2001. 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, AD/Cycle, BookManager, C/370, IBMLink, Language Environment, MVS/ESA, MVS/XA, NetView, OS/390, RACF, SAA, 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.

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.
Chapter 2. HLL Installation Exit Routines ........................................ 11
Overview of Installation Exit Routines ........................................ 11
  Coding Restrictions .......................................................... 11
  Performance Considerations ............................................... 12
  General Return Codes ...................................................... 12
  Deleting Messages ................................................................ 12
  Replacing Messages .......................................................... 12
Installation Exits ................................................................. 14
  BNJPALEX: Alert Generation Exit Routine ............................ 14
  DSIEX01: Input from the Operator ....................................... 14
  DSIEX02A: Output to the Operator ...................................... 14
  DSIEX03: Input Before Command Processing ......................... 15
  DSIEX04: Log Output .......................................................... 15
  DSIEX05: Before VTAM Command Invocation ........................ 16
  DSIEX06: Solicited VTAM Messages ..................................... 16
  DSIEX07: Cross-Domain Command Send .............................. 17
  DSIEX09: Output to the System Console ............................. 17
  DSIEX10: Input from the System Console ............................ 17
  DSIEX11: Unsolicited VTAM Messages ................................. 17
  DSIEX12: Logon Validation ................................................ 18
  DSIEX13: OST/NNT Message Receiver ................................. 18
  DSIEX14: Before Logoff ...................................................... 19
  DSIEX16: Post-Automation Table Installation Exit for Messages .... 20
  DSIEX16B: Post-Automation Table Installation Exit for MSUs .... 20
  DSIEX17: MVS Message and DOM Receive ............................ 20
  DSIEX18: Log Browse Installation Exit ............................... 20
  DSIEX19: RUNCMD Installation Exit ................................... 20
  DSIEX20: SAW Exit ........................................................... 21
  DSIEX21: Encryption Key for DSITCPF Installation Exit ......... 21
XITBN: BSAM Empty File ...................................................... 21
XITBO: BSAM Output ........................................................... 21
XITCI: CNM Interface Input .................................................. 22
XITCO: CNM Interface Output ............................................... 24
XITDI: Data Services Task Initialization .............................. 24
XITVI: VSAM Input .............................................................. 25
XITVN: VSAM Empty File ..................................................... 25
XITVO: VSAM Output ........................................................... 26
XITXL: External Logging ....................................................... 26

Chapter 3. HLL Data Services Command Processors ............................. 27
Installing the Data Services Task ............................................. 27
Data Services Command Processors ......................................... 28
  CNM Data Services ............................................................ 28
  VSAM Services ................................................................. 29
User-Defined Services ......................................................... 30
Scheduling Commands under the DST ..................................... 30

Part 2. Enabling a High-Level Language Program ............................... 31

Chapter 4. Preinitialized and Non-Preinitialized Environments .............. 33
Languages Supported ........................................................... 33
Advantages and Disadvantages of Preinitialized Environments .......... 33
Steps for Implementing Command Processors and Installation Exits ...... 34
  Non-Preinitialized Environments ......................................... 34
  Preinitialized Environments ............................................... 35
Chapter 5. Compiling, Link-Editing, and Running Your HLL Program ........................................ 43
Compiling .................................................................................................................. 43
Link-Editing ............................................................................................................. 43
Running ...................................................................................................................... 45

Part 3. Writing a PL/I Program .................................................................................. 47

Chapter 6. Coding HLL Command Processors and Installation Exits .................. 49
Initial Parameters ................................................................................................. 49
Run-Time Options ................................................................................................. 50
   CEEUOPT Run-Time Option CSECT ..................................................................... 50
   LE/390 Preinitialization Defaults ....................................................................... 50
   FSTACK and PHEAP .......................................................................................... 51
   LEOPTS Static External Variable ....................................................................... 51
Parameters Passed to HLL Service Routines ...................................................... 51
   Pointer Variables .............................................................................................. 52
   Integer Variables .............................................................................................. 53
   Fixed-Length Character Strings ....................................................................... 53
   Varying Length Character Strings .................................................................... 54
Control Blocks and Include Files .......................................................................... 55
PL/I Input and Output Considerations ................................................................. 56
   PL/I Input and Output Considerations in a Preinitialized Environment ........... 57
   PL/I Run-Time Considerations ......................................................................... 57
Considerations for HLL Command Processors ..................................................... 58
Return Codes ......................................................................................................... 58
Restrictions for HLL Programs Written in PL/I .................................................... 58
   Restrictions for PL/I Programs Running in a Preinitialized Environment ......... 58

Chapter 7. PL/I High-Level Language Services ...................................................... 61
PL/I Sample Template ........................................................................................... 61
Data Queue Management ...................................................................................... 63
Sending Messages .................................................................................................. 64
Parsing Input Strings Similar to NetView Command List Language ..................... 65
Parsing Input String Similar to REXX ..................................................................... 66
Parsing Input String in CNMSCAN ....................................................................... 67
Invoking Synchronous Commands ...................................................................... 70
Sending Commands .............................................................................................. 70
Waiting and Trapping ............................................................................................ 71
   Example of Using the TRAP and WAIT Commands (PL/I) ......................... 71
   Example of Using the PIPE Command (PL/I) .............................................. 73
Retrieving Information .......................................................................................... 75
Command List Variable Access ............................................................................ 76
   Accessing Common Global Variables (PL/I) .................................................. 77
   Accessing Stem Variables Using the PIPE Command (PL/I) ...................... 78
Using Locks ............................................................................................................ 79
Operator Input ....................................................................................................... 80
VIEW Command Processor .................................................................................. 80
Message Processing ............................................................................................... 82
Command Authorization Checking ....................................................................... 82
   Command Authorization Checking Using CMDAUTH=TABLE ..................... 83
   Command Authorization Checking Using CMDAUTH=SAF ....................... 83
Command Authorization Checking Sample Code ..................................................... 85
Altering Data ........................................................................................................... 86
Storage Access ........................................................................................................ 88
Data Set Access ....................................................................................................... 89
Communicating with Devices ................................................................................... 90
Performing I/O on a VSAM File (Keyed File Access) ............................................. 93
Coding a DST Installation Exit .............................................................................. 95
Coding an Installation Exit ..................................................................................... 96
Coding the WAIT FOR DATA Function ................................................................. 97
  Requesting Data (PL/I) ....................................................................................... 98
  Sending Data (PL/I) ........................................................................................ 100
Automating MSUs ................................................................................................... 101
Translating Code Points ........................................................................................... 102
Registering Applications with MS Transport and Operations Management ......... 103
Registering Applications with the High Performance Transport ......................... 106
Sending an Alert over the MS Transport ............................................................... 109
Sending an Alert over the High Performance Transport ........................................ 110
Sending an MDB to NetView for Processing ......................................................... 112

Part 4. Writing a C Program ...................................................................................... 117

Chapter 8. Coding Your C Program-Interfaces and Restrictions ............................. 119
Initial Parameters .................................................................................................... 119
C Run-Time Options ................................................................................................ 119
  #pragma runopts .............................................................................................. 120
  LE/390 Preinitialization Defaults ....................................................................... 120
  PSTACK and PHEAP ....................................................................................... 121
  LEOPTS Static External Variable .................................................................... 121
Parameters Passed to HLL Service Routines ......................................................... 121
  Pointer Variables ............................................................................................. 122
  Integer Variables ............................................................................................ 123
  Fixed-Length Character Strings ..................................................................... 124
  Varying Length Character Strings ................................................................ 124
Cnmvlc: Converting a String to a Varying Length Character String ..................... 126
Cnmnvlc: Converting a String to a Varying Length Character String Using Length .... 127
Control Blocks and Include Files .......................................................................... 128
C Input/Output Considerations ............................................................................ 129
C Run-Time Considerations ................................................................................... 130
Considerations for HLL Command Processors ..................................................... 131
Return Codes ......................................................................................................... 131
Restrictions for HLL Programs Written in C Language ........................................ 131
  Restrictions for C Programs Running in a Preinitialized Environment ............... 132

Chapter 9. C High-Level Language Services ......................................................... 133
C Sample Template ................................................................................................ 133
Varying Length Character Strings ........................................................................ 136
  Cnmvlc ............................................................................................................ 136
  Cnmnvlc ......................................................................................................... 137
  Defining Varying Length Character Strings ...................................................... 138
Data Queue Management ....................................................................................... 138
Sending Messages ................................................................................................. 139
Invoking Synchronous Commands ........................................................................ 140
Sending Commands ............................................................................................... 141
Waiting and Trapping ............................................................................................. 143
  Example of Using the TRAP and WAIT Commands (C) .................................. 143
  Example of Using the PIPE Command (C) ....................................................... 146
Retrieving Information ......................................................................................... 149
Command List Variable Access ............................................................................ 151
  Accessing Common Global Variables (C) ......................................................... 152
  Accessing Stem Variables Using the PIPE Command (C) ............................... 154
Using Locks ................................... 155
Operator Input .................................. 158
VIEW Command Processor ...................... 159
Message Processing .............................. 161
Command Authorization Checking ............... 163
  Command Authorization Checking Using CMDAUTH=TABLE 164
  Command Authorization Checking Using CMDAUTH=SAF 164
  Command Authorization Checking Sample Code 166
Altering Data .................................. 167
Storage Access .................................. 169
Data Set Access ................................. 171
Communicating with Devices ..................... 173
Performing I/O on a VSAM File (Keyed File Access) .. 178
Coding a DST Installation Exit ................... 181
Coding an Installation Exit ..................... 183
Coding the WAIT FOR DATA Function ............. 185
  Requesting Data ............................... 186
  Sending Data ................................ 188
Automating Management Services Units (MSUs) ........ 190
Translating Code Points .......................... 191
Registering Applications with MS Transport and Operations Management ....... 192
Registering Applications with the High Performance Transport .................. 193
Sending an Alert Over the MS Transport .......... 195
Sending Data Over the High Performance Transport ..................... 197
Sending an MDB to NetView for Processing 197

Part 5. HLL Debugging and Service Routine Reference ............... 201

Chapter 10. Testing and Debugging ....................... 203
Using RID to Monitor a Task ...................... 203
RID Command .................................. 203
RID Scenario .................................. 205

Chapter 11. Service Reference .............................. 209
Composite Return Codes ......................... 209
Command Reference ................................ 212
  GLOBALV Command .............................. 212
  GO Command .................................. 213
  PIPE Command ................................ 214
  QUEUE Command ............................... 214
  RESET Command ............................... 215
  TRAP Command ................................ 218
  WAIT Command ................................ 218
HLL Service Routine Reference ..................... 219
  CNMALTD (CNMALTDATA): Alter Data on a Queue .... 219
  CNMAUTO (CNMAUTOTAB): Invoke Automation Table .... 221
  CNMCELL (CNMSTRCELL): Storage Cell ................ 222
  CNMCMDC (CNMCOMMAND): Invoke NetView Commands .... 223
  CNMCNMI (CNMI): CNMI Access Under a DST .......... 224
  CNMCPSY (CNMCOPYSTR): Copy Storage ................ 226
  CNMCTFT (CNMCOPIE2TXT): Code Point Translation .... 228
  CNMGETA (CNMGETATTR): Query Message Attributes .... 229
  CNMGETD (CNMGETDATA): Data Queue Manipulation .... 240
  CNMHRGS (CNMHRGSTR): High Performance Transport Application Registration .... 244
  CNMHSMU (CNMHSENDMU): Send High Performance Message Unit .......... 247
  CNMINFR (CNMINFOC): Query NetView Character Information .......... 253
  CNMINF (CNMINFOR): Query NetView Integer Information .......... 256
  CNMKIO (CNMKKEYIO): Keyed File Access Under a DST .......... 259
  CNMLK (CNMLOCK): Controlling a Lock ................ 261
  CNMMEMC (CNMCLMEM): Close NetView Partitioned Data Set .......... 263

Contents vii
Part 6. Appendixes ............................................................. 299

Appendix A. PL/I Control Blocks and Include Files ................. 301
DISPL Include File .......................................................... 301
DISPCONS Control Block .................................................. 302
DISPHLB Control Block .................................................... 305
DISPORIG Control Block ................................................... 306
DISPHLLS Control Block ................................................... 307
DISPNCN Control Block .................................................... 315
DISPPRM Control Block ................................................... 319

Appendix B. PL/I Samples ....................................................... 321
PL/I Samples Table .......................................................... 321
PL/I Samples Descriptions .................................................. 322
PTMPPLT (CNMS4200) ....................................................... 322
PEXIT3 (CNMS4210) .......................................................... 322
PSNDDDAT (CNMS4211) ..................................................... 322
PWATDAT (CNMS4212) ....................................................... 323
PEXIT2A (CNMS4213) ....................................................... 323
PCNMI (CNMS4214) .......................................................... 323
PKEYIO (CNMS4215) ......................................................... 323
PSICOPCK (CNMS4216) ...................................................... 324
PFPLVIEW (CNMS4217) ..................................................... 324
PACTLU (CNMS4218) ........................................................ 324
PSEQLOG (CNMS4219) ....................................................... 324
PXITDDI (CNMS4220) ......................................................... 324
PXITLOI (CNMS4221) ........................................................ 324
PSNDDDST (CNMS4222) ..................................................... 325
PDOVSAM (CNMS4223) ....................................................... 325
PPRIME (CNMS4224) ........................................................ 325
PHESPNDMU (CNMS4226) ................................................... 325
PRODMCON (CNMS4230) .................................................... 325
PAUTOTB (CNMS4231) ....................................................... 325
PREGSTR (CNMS4232) ....................................................... 325
PSENDMU (CNMS4233) ...................................................... 326
PHECREAT (CNMS4236) ...................................................... 326
PISTRMDBC (CNMS4239) ..................................................... 326
PACTPI (CNMS4305) ........................................................ 326

Appendix C. C Language Control Blocks and Include Files. .......... 327
DISC Include File ............................................................ 327
DISCCCONS Control Block .................................................. 328
DISCVARSC Type Structure ................................................ 331
DISCHLB Control Block ..................................................... 332
DISCORG Control Block ..................................................... 333
DISCRP Control Block ....................................................... 334
### Appendix D. C Samples

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Language Samples Table</td>
<td>345</td>
</tr>
<tr>
<td>C Language Samples Descriptions</td>
<td>346</td>
</tr>
<tr>
<td>CTMPPLT (CNMS4201)</td>
<td>346</td>
</tr>
<tr>
<td>CEXIT3 (CNMS4240)</td>
<td>346</td>
</tr>
<tr>
<td>CSNDDAT (CNMS4241)</td>
<td>346</td>
</tr>
<tr>
<td>CWATDAT (CNMS4242)</td>
<td>346</td>
</tr>
<tr>
<td>CEXIT2A (CNMS4243)</td>
<td>347</td>
</tr>
<tr>
<td>CCNMI (CNMS4244)</td>
<td>347</td>
</tr>
<tr>
<td>CKEYIO (CNMS4245)</td>
<td>347</td>
</tr>
<tr>
<td>CSCOPCK (CNMS4246)</td>
<td>347</td>
</tr>
<tr>
<td>CFLVIEW (CNMS4247)</td>
<td>347</td>
</tr>
<tr>
<td>CACTLU (CNMS4248)</td>
<td>347</td>
</tr>
<tr>
<td>CSEQLOG (CNMS4249)</td>
<td>347</td>
</tr>
<tr>
<td>CXITDI (CNMS4250)</td>
<td>348</td>
</tr>
<tr>
<td>CXITVN (CNMS4251)</td>
<td>348</td>
</tr>
<tr>
<td>CSNDDST (CNMS4252)</td>
<td>348</td>
</tr>
<tr>
<td>CDVSAM (CNMS4253)</td>
<td>348</td>
</tr>
<tr>
<td>Cprime (CNMS4254)</td>
<td>348</td>
</tr>
<tr>
<td>CHSNDMU (CNMS4256)</td>
<td>348</td>
</tr>
<tr>
<td>CRODMCON (CNMS4260)</td>
<td>349</td>
</tr>
<tr>
<td>CAUTOTB (CNMS4261)</td>
<td>349</td>
</tr>
<tr>
<td>CREGISTR (CNMS4262)</td>
<td>349</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263)</td>
<td>349</td>
</tr>
<tr>
<td>CHREGSTR (CNMS4266)</td>
<td>349</td>
</tr>
<tr>
<td>CPRSMB (CNMS4269)</td>
<td>350</td>
</tr>
<tr>
<td>CACTPI (CNMS4405)</td>
<td>350</td>
</tr>
</tbody>
</table>

### Index

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>351</td>
</tr>
</tbody>
</table>
Preface

This document describes how a system programmer can tailor or supplement Tivoli NetView for z/OS Version 5 Release 1 to satisfy unique requirements or operating procedures.

Who Should Read This Document

This document is intended for experienced system programmers who are knowledgeable in PL/I or C. The programmers should already be familiar with the functions that NetView® provides.

Secondary users of this document include operators, network planners, designers, and systems analysts, and Tivoli® marketing representatives and instructors.

What This Document Contains

This document is organized into the following sections that address the unique requirements or operating procedures for the NetView program.

“Chapter 1. High-Level Language Services” on page 3 describes the NetView services available to you for designing your command processor or installation exit routine.

“Chapter 2. HLL Installation Exit Routines” on page 11 describes how to write installation exit routines to view, delete, or replace data flowing to, from, or through the NetView program.

“Chapter 3. HLL Data Services Command Processors” on page 27 describes data services command processors written in an HLL.

“Chapter 4. Preinitialized and Non-Preinitialized Environments” on page 33 explains preinitialized and non-preinitialized environments. This chapter also describes the steps required to run your program in either environment.

“Chapter 5. Compiling, Link-Editing, and Running Your HLL Program” on page 43 explains how to modify the PL/I and C compile and link-edit job control language (JCL) for use with the NetView program. This chapter also provides several examples of compile and link-edit JCL.

“Chapter 6. Coding HLL Command Processors and Installation Exits” on page 49 provides the necessary information for coding HLL command processors and installation exits in PL/I. The appropriate interfaces and language-dependent restrictions are discussed in this chapter.

“Chapter 7. PL/I High-Level Language Services” on page 61 is an example-oriented discussion of commands and services the NetView program provides in PL/I.

“Chapter 8. Coding Your C Program-Interfaces and Restrictions” on page 113 provides the necessary information for coding HLL command processors and installation exits in C. The appropriate interfaces and language-dependent restrictions are described in this chapter.
Chapter 9. C High-Level Language Services on page 133 is an example-oriented discussion of commands and services the NetView program provides in C.

Chapter 10. Testing and Debugging on page 203 describes the NetView remote interactive debugger (RID) and how you can use it to debug HLL programs.

Chapter 11. Service Reference on page 209 describes HLL command and service routines, and their associated operands.

Appendix A. PL/I Control Blocks and Include Files on page 301 describes the PL/I control blocks and the include files needed to write command processors and installation exits in PL/I.

Appendix B. PL/I Samples on page 321 contains a table of the PL/I samples shipped with the NetView program, descriptions of these samples, and coded examples of an installation exit and two command processors.

Appendix C. C Language Control Blocks and Include Files on page 327 describes the C control blocks and the include files needed to write command processors and installation exits in C.

Appendix D. C Samples on page 345 contains a table of the C samples shipped with the NetView program, descriptions of these samples, and coded examples of an installation exit and two command processors.

Publications

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

Prerequisite and Related Documents

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

You can find additional product information on these Internet sites:

Table 1. Resource Web sites

<table>
<thead>
<tr>
<th>Web site</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** OS/390 and 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 xvii 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.

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.

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 xvii 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 xviii 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 xix, 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 xvii.

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.

\textbf{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\texttextsuperscript{®} 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.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
This element... & Looks like this... \\
\hline
Keyword & CCPLOADF \\
Variable & \textit{resname} \\
Operand & MEMBER=\texttt{membername} \\
Default & \textit{today} or INCL \\
\hline
\end{tabular}
\caption{Syntax Elements Examples}
\end{table}
Abbreviations

Command and keyword abbreviations are described in synonym tables after each command description.
Part 1. Overview

Chapter 1. High-Level Language Services.  3
  Synchronous Command Execution.  3
  Sending Commands (Asynchronous Command Execution).  3
  Client and Server Request Response Handling.  3
  Operator Interaction.  3
    Line Mode Input.  4
    Line Mode Output.  4
    Full-Screen Input/Output.  4
  Data Access.  4
    Message Trapping.  4
    NetView Message Automation.  4
    NetView MSU Automation.  4
    System Symbolic Access.  5
    Command List Variable Access.  5
    Querying NetView Information.  5
    VSAM Files.  5
    Data Queue Manipulation.  5
  Logical Unit 6.2 Transport.  6
  Operating Remote Systems.  6
  Communication Network Management Interface.  6
  NetView Partitioned Data Sets.  7
  Dynamic File Allocation/Deallocation.  7
  Storage Copying.  7
  Named Storage.  7
  User-Defined Lock Management.  7
  Parsing Character Strings.  7
  Command Authorization Checking.  7
  NetView Message Logging.  8
  NetView Bridge.  8
  NetView Bridge Remote Access.  8
  Debugging Support.  8
    Remote Interactive Debugger (RID).  8
    First Failure Data Capture Trace (FFDCT).  9

Chapter 2. HLL Installation Exit Routines  11
  Overview of Installation Exit Routines.  11
    Coding Restrictions.  11
    Performance Considerations.  12
    General Return Codes.  12
    Deleting Messages.  12
    Replacing Messages.  12
  Installation Exits.  14
    BNJPALLEX: Alert Generation Exit Routine.  14
    DSIEX01: Input from the Operator.  14
    DSIEX02A: Output to the Operator.  14
    DSIEX03: Input Before Command Processing.  15
    DSIEX04: Log Output.  15
    DSIEX05: Before VTAM Command Invocation.  16
    DSIEX06: Solicited VTAM Messages.  16
    DSIEX07: Cross-Domain Command Send.  17
    DSIEX09: Output to the System Console.  17
    DSIEX10: Input from the System Console.  17
    DSIEX11: Unsolicited VTAM Messages.  17
    DSIEX12: Logon Validation.  18
    DSIEX13: OST/NNT Message Receiver.  18
    DSIEX14: Before Logoff.  19
    DSIEX16: Post-Automation Table Installation Exit for Messages.  20
    DSIEX16B: Post-Automation Table Installation Exit for MSUs.  20
    DSIEX17: MVS Message and DOM Receive.  20
    DSIEX18: Log Browse Installation Exit.  20
    DSIEX19: RUNCMD Installation Exit.  20
    DSIEX20: SAW Exit.  21
    DSIEX21: Encryption Key for DSITCPRF Installation Exit.  21
      XITBN: BSAM Empty File.  21
      XITBO: BSAM Output.  21
      XITDI: CNM Interface Input.  22
      XITDO: CNM Interface Output.  24
      XITD: Data Services Task Initialization.  24
      XITVI: VSAM Input.  25
      XITVO: VSAM Output.  26
      XITXL: External Logging.  26

Chapter 3. HLL Data Services Command Processors  27
  Installing the Data Services Task.  27
  Data Services Command Processors.  28
    CNM Data Services.  28
    Unsolicited CNM Data Interface.  28
    VSAM Services.  29
    User-Defined Services.  30
  Scheduling Commands under the DST.  30
Chapter 1. High-Level Language Services

Tivoli NetView for z/OS V5R1 enables you to manage complex, multivendor networks and systems from a single point. This chapter describes NetView services available to you for designing your command processor or installation exit routine.

Note: Before reading this chapter, you should be familiar with the information in the Tivoli NetView for z/OS Customization Guide. You should also have NetView experience and programming experience in PL/I or C.

Synchronous Command Execution

High-level language (HLL) command processors can invoke any NetView command, including:
- Simple commands
- Command lists
- REXX command procedures
- Assembler command processors
- NetView applications such as the session monitor
- Other HLL command processors

The command must be executable in the calling environment. For example, data services commands can only be invoked from a data services command processor.

Sending Commands (Asynchronous Command Execution)

HLL installation exit routines cannot invoke NetView commands directly. However, all HLL command processors and installation exit routines can schedule NetView commands to be executed asynchronously under any NetView task.

Client and Server Request Response Handling

The NetView program supports server tasks that service and reply to requests from one or more operator tasks. This is accomplished by:
- Allowing the requesting command processor to wait for the reply, as follows:
  - The requesting command processor suspends during the wait.
  - The task continues processing other commands.
  - The suspended command processor resumes processing after receiving the reply.
- Allowing the requests and replies to be sent over NetView-NetView cross-domain operator sessions.
- Correlating the reply with the correct activation of the requesting HLL command processor. This correlation in turn allows multiple active instances of the requesting command processor under a single operator task.

Operator Interaction

The following sections describe operator interaction in line mode and full-screen mode.
High-Level Language Services

Line Mode Input

HLL command processors running under an operator station task (OST), NetView-NetView task (NNT), or primary program operator interface task (PPT) can accept line mode input from an operator. This function is similar to that provided by the NetView command list language &PAUSE statement, except that the HLL command processor can continue to run while waiting for operator input.

Operators do not need to know the language in which a command procedure is written. They can use the GO command to provide input to a command procedure written in the NetView command list language, REXX, or HLL. For more information, see “GO Command” on page 213.

Line Mode Output

HLL command processors and most installation exit routines can send line mode output to:
- A NetView operator
- The operating system console
- Another task
- The authorized receiver
- A group of operators defined by the NetView ASSIGN command

Multiline messages can be sent as a single unit using a multiline write-to-operator (MLWTO) so that an operator receives them in sequence, with no messages from other sources interspersed. This type of output appears on the command facility panel or on the operating system console.

Full-Screen Input/Output

HLL command processors can invoke the NetView VIEW command to provide full-screen interaction with an operator. This function is similar to the use of the VIEW command from a command list. You can also roll among NetView applications, including HLL command processors. HLL command processors are treated like command lists when determining roll groups. NetView also provides the capability to asynchronously update a panel while it is being displayed.

Data Access

The following sections describe data access techniques available to HLL command processors or installation exit routines.

Message Trapping

Frequently, command procedures need to intercept or trap and process messages that would ordinarily go to an operator. The NetView HLL application programming interface (API) provides this function for single and multiline messages.

NetView Message Automation

NetView message automation can pass both single and multiline messages and their resulting command strings to an HLL command processor. It also provides services to alter the contents of the messages.

NetView MSU Automation

The NetView automation table supports the automation of alerts and management services units (MSUs). NetView MSU automation allows the invocation of HLL
command processors from the automation table upon receipt of MSUs. NetView also provides the command processor with access to the command and the MSU that invoked the command.

**System Symbolic Access**

HLL command processors and installation exits can query the values of the MVS and user defined system symbols, as well as the local NetView defined &DOMAIN system symbol. The &DOMAIN value is unique across the NetView program in which the HLL command processor is executing, while the MVS and user defined system symbols are unique across the MVS system on which they are defined.

**Command List Variable Access**

NetView command lists and assembler command processors can store data in task global and common global command list variables that can be accessed by other command procedures. HLL command processors and installation exit routines can access and update these variables.

**Querying NetView Information**

HLL command processors and installation exit routines can query certain information (such as domain ID or message attributes) about the current NetView environment. This information is similar to that provided by control variables in the NetView command list language and control block fields in assembler language.

**VSAM Files**

HLL data services command processors can read, write, update, and delete records in VSAM files associated with the task under which the command processor is running. Other command processors are allowed to execute while the input/output (I/O) requests are executing.

The pipeline facility supports the use of the CNMCMD service to call DSIVSAM and DSIVSMX to access VSAM files. Refer to the Tivoli NetView for z/OS Command Reference or the NetView online help for information about DSIVSAM and DSIVSMX.

**Data Queue Manipulation**

An HLL command processor or installation exit routine can manipulate HLL data queues. Each HLL command processor and installation exit routine has a set of queues from which it can receive data. Each of these types of input data has a queue:

- Operator messages trapped for processing
- Input from a NetView operator
- Data from another HLL command processor or installation exit routine
- Initial data associated with:
  - The full message or MSU that caused an HLL command processor to be invoked
  - The full message that caused DSIEX02A to be invoked
  - The multiple domain support message unit (MDS-MU) for unsolicited requests or asynchronous replies
- Data solicited over the communication network management (CNM) interface
- MDS-MUs received synchronously
Logical Unit 6.2 Transport

The NetView logical unit 6.2 (LU 6.2) transport is a programming interface that implements architected protocols to enable applications in network nodes to communicate using conversations over LU 6.2 sessions.

The NetView LU 6.2 transport consists of two similar application program interfaces: the management services (MS) transport and the high performance transport. For applications that run in NetView, each transport provides a high-level programming interface to mask the LU 6.2 complexities. An application registered with the appropriate transport can send data in architected envelopes to a partner application and receive data in return.

Although both transports provide the same functions and mask the LU 6.2 complexities, each transport offers its own advantages.

High Performance Transport
- Uses different LU 6.2 protocols that are faster than the protocols used by the MS transport.
- Provides general error notification rather than specific error notification about data.
- Enables programmers to define session parameters such as RU size.

MS Transport
- Uses LU 6.2 conversation protocols that generate more network traffic than the high performance transport protocols to transport each piece of data.
- Guarantees delivery of data or specific error notification about the data.


Operating Remote Systems

Operations management support enables you to use NetView-provided and user-written operations management served applications to send architected operations management data to remote systems for execution. Operations management makes it possible to have a command routed to the appropriate command processor in the target system for execution, and to get the acceptance reports, completion reports, and other delayed replies from the target system back to the operator controlling that system. In this way, the system can be activated or deactivated by command, the system clock can be set, and other device specific commands can be set to operate the system remotely.

Communication Network Management Interface

HLL data services command processors can send and receive data over the communication network management (CNM) interface. The CNM interface is used to forward commands to and collect data from devices in the network. For example, response time monitor (RTM) data is collected from PU type 2 control units using the CNM interface. HLL command processors can also process unsolicited data received over the CNM interface. Solicited replies are pseudosynchronous.
NetView Partitioned Data Sets

HLL command processors and installation exit routines have read access to the NetView partitioned data sets. This enables you to write a program using the information in the NetView partitioned data sets. This function is completely synchronous.

Dynamic File Allocation/Deallocation

NetView provides facilities to dynamically allocate and deallocate files by using NetView ALLOCATE and FREE commands. Refer to the NetView online help for more information.

When allocated, these files can be accessed using the file I/O facilities present in the language being used.

Storage Copying

HLL command processors and installation exit routines can make a copy of any area of virtual storage in the NetView address space in which the HLL command processor or installation exit routine is running. A request to copy an area outside of this space results in a return code instead of an OC4 ABEND. This feature is useful for debugging in that it enables you to intercept and act upon a return code instead of failing with a hard ABEND.

Note: Ensure that the storage area to which the copy is made belongs to your program.

Named Storage

HLL command processors or installation exit routines can allocate and free an area of virtual storage and associate a name with it so that other HLL command processors and installation exit routines running under the same task can access this area of storage. Transaction-oriented applications can use this function to save data across transactions.

User-Defined Lock Management

HLL command processors and installation exit routines can obtain, release, and test the control of a named lock. The lock management scheme uses a simple alphanumeric hierarchy. Locking is useful when updating common global variables, or to serialize any other common resource.

Parsing Character Strings

A parsing service is provided as part of the HLL support for PL/I. This service is similar to the SSCANF function available in the C language and is intended to facilitate the parsing of commands and messages.

Command Authorization Checking

The NetView command authorization services can be invoked by HLL command processors and installation exit routines to determine whether a particular operator is authorized to issue a command with restricted operands or operand values.
NetView Message Logging

All HLL command processors and most installation exit routines can send message output to:
- The network log
- An external log such as system management facilities (SMF)
- A sequential log

NetView Bridge

The NetView Bridge option allows data to be transported between a NetView system running in MVS and a non-NetView database. This option is useful when you need to access a non-NetView database to create and update problem records, or to access and retrieve configuration data.

Note: The NetView Bridge is an effective means of connecting an active NetView installation to other databases by using user-written command procedures.

NetView Bridge Remote Access

The NetView Bridge remote access option enables you to access servers that reside on hosts other than the one in which the transaction was generated. You can collect data from the resources in your network, and create and update records in a database residing in another system. NetView Bridge remote access can operate from a NetView program running on an OS/390 operating system. Transactions are forwarded to the resident host by using the high performance transport API to access an MVS system running both NetView Bridge and NetView Bridge remote access.

Debugging Support

The NetView HLL API provides two debugging tools for users: an interactive debugger that displays the operands and results of all HLL API service routine invocations, and a continuous first failure data capture trace for ABEND debugging.

Note: In addition, you can use the NetView internal trace. Refer to “Diagnostic Tools for the NetView Program” in the Tivoli NetView for z/OS Diagnosis Guide for more information.

Remote Interactive Debugger (RID)

The RID enables NetView HLL service routine calls to be trapped and displayed to the programmer. You can implement RID using NetView commands and messages to create debugging procedures using NetView command list language, REXX, or HLL command procedures. In addition, because the NetView program provides facilities to route commands and messages to remote systems, use RID from one system to debug an HLL command processor or installation exit routine running on another system.

RID operates at the subtask level, so using RID to stop an HLL command processor or installation exit routine running under one subtask does not affect other subtasks in the same NetView address space.
First Failure Data Capture Trace (FFDCT)

Each HLL command processor or installation exit routine maintains an 8-entry continuously wrapping trace area. Trace entries are recorded at entry to, and exit from, HLL service routines and at other key points inside the HLL routines. In the event of an ABEND, this area gives some indication of what was occurring before the ABEND. Refer to “Diagnostic Tools for the NetView Program” in the *Tivoli NetView for z/OS Diagnosis Guide* for more information.
This chapter contains product-sensitive programming interfaces and associated guidance information provided by the NetView program.

You can write installation exit routines to view, delete, or replace data flowing to, from, or through the NetView program. For example, your code can examine the messages passing through the NetView program, record relevant data, and initiate work requests based on the data. In addition, your code can delete any unnecessary messages from further processing or substitute a modified message in place of the original message. Thus, installation exit routines can handle a specific event with nonstandard processing and automate processes based on message information.

Overview of Installation Exit Routines

The NetView program 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 the NetView program starts. See Table 4 on page 13 for a list of installation exits.
- DST installation exits (XITnn and BNJPALEX), which apply only to DSTs. The DST installation exit routines are loaded when their DST starts. Each DST can have its own set of installation exit routines.

Notes:

1. The NetView program supports the use of DST installation exits under the network product support (NPS) task named DSIGDS using the Programmable Network Access (PNA) support enhancement. The PNA support enables the operator to issue NetView commands with no change in command syntax, and with no additional knowledge of the configuration except for commands for PNA. However, knowledge of the configuration below a PNA is required.

2. Changes will not take effect until you recycle the NetView program.

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

For more information about input to the installation exit routines, see "Chapter 6. Coding HLL Command Processors and Installation Exits" on page 49 and "Chapter 8. Coding Your C Program-Interfaces and Restrictions" on page 113.

Coding Restrictions

The following HLL service routines cannot be invoked from any installation exit:

- CNMCMD
- CNMCNMI
- CNMKIO
HLL Installation Exit Routines

In addition, CNMSMSG cannot be issued from DSIEX04 and DSIEX09. Only CNMSMSG with a destination type of TASK can be issued from DSIEX02A. DSIEX02A, DSIEX04, and DSIEX09 can only be invoked in the mainline environment if written in HLL. If written in assembler, these installation exit routines can be invoked in both the mainline and the interruption request block (IRB) exit environments. Refer to Tivoli NetView for z/OS Customization: Using Assembler for additional information about exits running in the IRB exit environment.

Performance Considerations

Preinitialized environments can significantly improve the performance of installation exits written in PL/I and C. Instead of repeatedly allocating and deallocating environments for each invocation of an installation exit routine, the NetView program manages global pools of preinitialized environments. Exit routines can then execute in these environments, without initializing them, each time an exit program is run.

Preinitialized environments are supported in all NetView-supported releases of PL/I and in AD/Cycle® C/370™. Preinitialized environments are not supported for IBM C/370 V2R1M0. See “Chapter 4. Preinitialized and Non-Preinitialized Environments” on page 33 for more information about preinitialized environments.

Avoid coding installation exits for frequently executed functions, such as VSAM input/output (I/O), because performance can be degraded significantly.

General Return Codes

Unless otherwise noted, installation exit routines pass the following return codes to the NetView program in the return code field (HLBRC for PL/I or Hlbrc for C) to indicate that the messages are to be unchanged, deleted, or replaced:

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

USERDROP (4)
Delete the message from the terminal and from the network log, system log, and hardcopy log; stop processing before the message appears on the screen. For more information about how to delete messages, see “Deleting Messages.”

USERSWAP (8)
Replace the message with the modified CMDBUF (Cmdbuf). For more information, see “Replacing Messages.”

Deleting Messages

To delete a message entirely, use return code USERDROP.

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

Replacing Messages

To replace a message, use return code USERSWAP and set the input CMDBUF (Cmdbuf) contents to the desired data. For installation exits written in PL/I, the
replacement data must be less than or equal in length to the original CMDBUF (Cmdbuf) data; otherwise it is truncated to the original length of the CMDBUF (Cmdbuf) data.

For installation exits written in C, the replacement data must be less than or equal in length to the original CMDBUF (Cmdbuf) data. If the replacement data is longer than the original data, storage overlay can occur which causes abends and other unpredictable results.

Installation exit DSIEX02A provides a more flexible interface for replacing messages using CNMALTD. See "CNMALTD (CNMALTDATA): Alter Data on a Queue" on page 219.

You can concatenate DST installation exit routines when replacing messages. In this case, the buffer containing the replacement message becomes the input for the subsequent DST installation exit routine. Refer to Tivoli NetView for z/OS Customization: Using Assembler for message flows.

Table 4 lists all HLL installation exits and the task environments under which they can be called.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Description</th>
<th>Applicable Tasks</th>
<th>Associated Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNJPALEX</td>
<td>Not supported in HLL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIEX01</td>
<td>Not supported in HLL</td>
<td></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, CNMCSSIR</td>
<td>CNMS4213, CNMS4243</td>
</tr>
<tr>
<td>DSIEX03</td>
<td>Input before command processing or HDRTYPEPEX cross-domain return command receive</td>
<td>NNT, OST, PPT NNT</td>
<td>CNMS4210, CNMS4211, CNMS4212, CNMS4240, CNMS4241, CNMS4242</td>
</tr>
<tr>
<td>DSIEX04</td>
<td>Log output for buffers not processed by DSIEX02A</td>
<td>Main task or any subtask</td>
<td></td>
</tr>
<tr>
<td>DSIEX05</td>
<td>Before VTAM command invocation(1)</td>
<td>NNT, OST, PPT</td>
<td></td>
</tr>
<tr>
<td>DSIEX06</td>
<td>Solicited VTAM messages(1)</td>
<td>NNT, OST, PPT</td>
<td></td>
</tr>
<tr>
<td>DSIEX07</td>
<td>Cross-domain command send</td>
<td>NNT, OST</td>
<td></td>
</tr>
<tr>
<td>DSIEX09</td>
<td>Output to the system console</td>
<td>Main task or any subtask</td>
<td></td>
</tr>
<tr>
<td>DSIEX10</td>
<td>Input from the system console</td>
<td>DSIWTOMT</td>
<td></td>
</tr>
<tr>
<td>DSIEX11</td>
<td>Unsolicited VTAM messages(1)</td>
<td>PPT</td>
<td></td>
</tr>
<tr>
<td>DSIEX12</td>
<td>Logon validation</td>
<td>NNT, OST</td>
<td></td>
</tr>
<tr>
<td>DSIEX13</td>
<td>OST/NNT message receiver</td>
<td>NNT, OST, PPT</td>
<td></td>
</tr>
<tr>
<td>DSIEX14</td>
<td>Before logging off</td>
<td>NNT, OST</td>
<td></td>
</tr>
<tr>
<td>DSIEX16</td>
<td>Not supported in HLL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIEX16B</td>
<td>Not supported in HLL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIEX17</td>
<td>Not supported in HLL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIEX18</td>
<td>Not supported in HLL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIEX19</td>
<td>Service point application authorization checking during RUNCMD processing</td>
<td>NNT, OST, PPT</td>
<td></td>
</tr>
<tr>
<td>DSIEX20</td>
<td>Not supported in HLL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**HLL Installation Exit Routines**

*Table 4. Installation Exit Environments (continued)*

<table>
<thead>
<tr>
<th>Exit</th>
<th>Description</th>
<th>Applicable Tasks</th>
<th>Associated Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIEX21</td>
<td>Encryption key for DSITCPRF</td>
<td>CNMTAMEL, NMC-3270</td>
<td>DSIEX21</td>
</tr>
<tr>
<td>XITBN</td>
<td>BSAM empty file</td>
<td>DST</td>
<td></td>
</tr>
<tr>
<td>XITBO</td>
<td>BSAM output</td>
<td>DST</td>
<td></td>
</tr>
<tr>
<td>XITCI</td>
<td>CNM interface input</td>
<td>DST</td>
<td></td>
</tr>
<tr>
<td>XITCO</td>
<td>CNM interface output</td>
<td>DST</td>
<td></td>
</tr>
<tr>
<td>XITDI</td>
<td>DST initialization</td>
<td>DST</td>
<td>CNMS4220, CNMS4223, CNMS4250, CNMS4253</td>
</tr>
<tr>
<td>XITVI</td>
<td>VSAM input</td>
<td>DST</td>
<td></td>
</tr>
<tr>
<td>XITVN</td>
<td>VSAM empty file</td>
<td>DST</td>
<td>CNMS4221, CNMS4223, CNMS4224, CNMS4245, CNMS4251, CNMS4253, CNMS4254</td>
</tr>
<tr>
<td>XITVO</td>
<td>VSAM output</td>
<td>DST</td>
<td></td>
</tr>
<tr>
<td>XITXL</td>
<td>External logging</td>
<td>DST</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** (1) When using NetView POI only. Does not include VTAM messages from other sources; for example MVS/ESA™ SSI. You can process these messages in DSIEX02A.

### Installation Exits

The NetView program provides a number of installation exits that are described in the following sections. For a description of message flows and interception points in OSTs, NNTs, and PPTs, refer to *Tivoli NetView for z/OS Customization: Using Assembler*.

#### BNJPALEX: Alert Generation Exit Routine

This exit is available only through assembler language. Refer to *Tivoli NetView for z/OS Customization: Using Assembler* for a description of the exit.

#### DSIEX01: Input from the Operator

This exit is available only through assembler language. Refer to *Tivoli NetView for z/OS Customization: Using Assembler* for a description.

#### DSIEX02A: Output to the Operator

*Description:* The NetView program calls DSIEX02A just before &WAIT (or WAIT) message processing and before the automation table is scanned to determine processing actions. DSIEX02A is called for standard output to an operator’s terminal. DSIEX02A runs before the device-dependent output is inserted and the data is logged. If DSIEX02A is called, DSIEX04 is not called because logging options can be specified in either DSIEX02A or the NetView automation table.

*Example of Use:* Because the message has been formatted but not yet displayed or logged, you can use DSIEX02A to delete or replace the message before it is automated, logged, or displayed. Messages are reformatted by removing, changing, and adding new buffers to the original message. DSIEX02A prevents OVERRIDE command options from taking effect for messages.

Changes to the English message text will not be reflected in the translated messages.
**Coding Considerations:** 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.

CNMSMSG can be issued from DSIEX02A, but only with the destination type of TASK. The message resulting from the CNMSMSG call does not re-drive DSIEX02A.

**Return Codes:** DSIEX02A differs from other user exits in the NetView program, because it always sets register fifteen to zero on return.

**Notes:**
1. Do not use the USERSWAP return code to swap messages. Use the CNMALTD service. See **CNMALTD (CNMALTDATA): Alter Data on a Queue** on page 218.
2. Messages issued by DSIPSS with TYPE=FLASH are not exposed.

DSIEX02A is supported only in 31-bit addressing mode.

**DSIEX03: Input Before Command Processing**

*Description:* All regular commands call DSIEX03. 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
- Entered as simulated terminal input
- 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 using the EXCMD command

Before running, all commands are passed to either DSIEX01 or DSIEX03. Immediate commands are passed to DSIEX01. Regular commands entered from a command facility panel are passed to DSIEX01 and DSIEX03. The remaining command types previously listed are passed to DSIEX03.

**Example of Use:** If your conditions are more complex than those provided by the authority table for NetView commands, you can use DSIEX03 to restrict the use of particular regular commands.

**DSIEX04: Log Output**

*Description:* The NetView program calls DSIEX04 during the logging and tracing process. DSIEX04 is located within log services and applies to messages logged on the network log, the external trace log, the MVS system log, and the hardcopy log. It runs before the message is reformatted and sent to the log. DSIEX04 is not called if DSIEX02A is called, because logging options can be specified in either DSIEX02A or the NetView automation table.

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

*Coding Considerations:* DSIEX04 can run under any subtask that initiates message logging. Ensure that any HLL services you request are supported by the subtask under which the routine is running. To determine the subtask under which your routine is running, see the TASK option under *icname* in "CNMINFO(CNMINFOC): Query NetView Character Information" on page 253.

*Return Codes:* In addition to USERASIS, USERDROP, and USERSWAP, DSIEX04 can pass these codes:

**Return Code**
**Meaning**
USERLOG (12)  
Write the message to the network or MVS system log only.
USERLOGR (16)  
Write the substituted message to the network or MVS system log only.
USERHCL (20)  
Write the message to the hardcopy log only.
USERHCLR (24)  
Write the substituted message to the hardcopy log only.
USERNSL (28)  
Do not write to the MVS system log.
USERNSLR (32)  
Do not write to the MVS system log. Use the substituted message to write to the network log, external trace log, and the hardcopy log.

**DSIEX05: Before VTAM Command Invocation**

*Description:* The NetView program calls DSIEX05 when preparing to pass a command to VTAM through the program operator interface (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:* This exit applies only to commands entered directly, without using the MVS prefix, that are passed through NetView’s POI.

Commands passed to DSIEX05 have already been processed under DSIEX03 (and possibly DSIEX01).

**DSIEX06: Solicited VTAM Messages**

*Description:* The NetView program calls DSIEX06 when it receives a solicited VTAM message, which is generated in response to a VTAM command the user or the PPT issued. The message has not yet been 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:* This exit applies only to responses to commands entered directly, without using the MVS prefix, that are passed through NetView’s POI.

NetView automation is invoked after this installation exit routine has been called. Therefore, any changes made to 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.

Messages processed (and not dropped) in DSIEX06 are subsequently processed by DSIEX02A.

**DSIEX07: Cross-Domain Command Send**

*Description:* The NetView program calls DSIEX07 before commands are sent cross-domain to an NNT.

*Example of Use:* You can use DSIEX07 to monitor cross-domain traffic through the network.

**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. Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for a description of the DSIWCS macro.

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

*Coding Considerations:* DSIEX09 is called as a result of DSIWCS macro calls. The output of the MVS console OST is processed by DSIEX02A instead of DSIEX09.

**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 allow the system console operator to enter command abbreviations and synonyms. These can then be expanded in the installation exit routine.

*Coding Considerations:* DSIEX10 can only be called from task DSIWTOMT, not from a subtask.

DSIEX10 is not called for commands entered by an operator using an MVS OST. DSIEX03 is called instead of DSIEX10.

**DSIEX11: Unsolicited VTAM Messages**

*Description:* The NetView program calls DSIEX11 when an unsolicited VTAM message is received through the POI interface. When VTAM’s PPOLOG=YES modify or start option is used, copies of the messages 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 CNMSMSG to send a copy of the message buffer prior to processing by the NetView program.

*Coding Considerations:* 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 is deleted by this installation exit routine.
HLL Installation Exit Routines

DSIEX12: Logon Validation

Description: The NetView program calls DSIEX12 at the completion of the logon process, after the logon has been accepted by the NetView program.

Example of Use: You can use DSIEX12 to perform additional checking of 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 takeover operator ID name field.

For a non-takeover logon, the operator ID is listed in the Operator ID name field and the Takeover Operator ID field is blank.

For a takeover logon, the value "takeover" is listed in the Operator ID field and the Takeover Operator ID name field contains the operator ID that is taken over.

For a takeover operator task, DSIEX12 is invoked mainly to validate the operator ID and password. Therefore, first check the Operator ID field to determine if this exit was invoked for a takeover operator task.

Coding Considerations: 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 exit is called under all OSTs and NNTs, including unattended operator and MVS console operator tasks.

The accompanying structure maps the header information in the CMDBUF (Cmdbuf) passed to the DSIEX12 exit. OFFSET and LENGTH values are given in bytes.

Table 5. DSIEX12 Command Buffer Parameter Field Descriptions

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>Operator ID name or &quot;takeover&quot;</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Operator LU name</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>Password</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>Hardcopy device name</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>Profile name</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>New password</td>
</tr>
<tr>
<td>48</td>
<td>8</td>
<td>Takeover Operator ID. The operator ID that is taken over when operator ID above is &quot;takeover&quot;.</td>
</tr>
</tbody>
</table>

DSIEX13: OST/NNT Message Receiver

Description: When certain subtask-subtask communication buffers are received on a subtask resulting from the use of the DSIMQS macro on another task, the NetView program calls DSIEX13. DSIEX13 is called when:

- A user-defined internal function request (IFRCODCR) is received through the macro DSIMQS for an OST, a NetView-to-NetView task (NNT), or a primary POI task (PPT).
A message buffer (with HDRMTYPE=HDRTYPEM) is received through the macro DSIMQS on OSTs or NNTs.

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

**Example of Use:** You can use DSIEX13 in conjunction with IFRCODCR to initiate a user function with a buffer. Code DSIEX13 to perform the user function specified by IFRCODCR. The IFRCODCR message buffer can be sent to another task or queued to the task you are running using the DSIMQS macro. Refer to [NetView for z/OS Customization: Using Assembler](#) for more information about internal function requests.

You can also use DSIEX13 to monitor DSI039I messages received by OSTs or NNTs.

**Coding Considerations:** DSIEX13 should not free (DSIFRE) the IFRCODCR or HDRTYPEM buffers.

**Return Codes:** HDRTYPEM messages and IFRCODCR buffers are treated differently:

- The NetView program does not process IFRCODCR buffers after DSIEX13 is called. NetView frees the buffer using DSIFRE. This is consistent with other buffers that are received through DSIMQS.
- HDRTYPEM buffers are displayed unless the return code USERDROP is used. The following occur for displayed messages just as for typical NetView messages:
  - DSIEX02A and DSIEX16 exits are called.
  - Automation is invoked.
  - Automation table processing occurs.
  - The messages are logged.

**DSIEX14: Before Logoff**

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

- If LOGOFF is entered at the operator’s terminal.
- If the subtask LOSTERM exit is driven (VTAM).
- If the subtask is posted to end.

The exit cannot communicate with the operator’s terminal. However, you can write to the system console and write entries to the log.

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

**Non-Takeover and Takeover Operator Task Processing:** For a non-taking 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 a buffer is not associated with logoff processing, DSIEX14 does not receive an input buffer (the length of the command buffer is 0).

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

**DSIEX16: Post-Automation Table Installation Exit for Messages**

This exit is available only through an assembler interface. Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**DSIEX16B: Post-Automation Table Installation Exit for MSUs**

This exit is available only through an assembler interface. Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**DSIEX17: MVS Message and DOM Receive**

This exit is available only through an assembler interface. Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**DSIEX18: Log Browse Installation Exit**

This exit is available only through an assembler interface. Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**DSIEX19: RUNCMD Installation Exit**

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

**Example of Use:** You can use DSIEX19 to provide security checking at the service point command level. This security checking can be done by calling CNMSCOP or by using your own technique. To use CNMSCOP 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. For example, to define the service point command ADP to enable checking with CNMSCOP, define:

ADP CMDMDL MOD=DSISPCMD

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

- DSIEX19 is passed a read-only copy of a command buffer in the USERMSG field. Table 6 shows the exit parameters in the CMDBUF passed to the exit.

**Table 6. DSIEX19 Command Buffer Parameter Field Descriptions**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>User ID against which command authorization checking was done for the RUNCMD.</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
Table 6. DSIEX19 Command Buffer Parameter Field Descriptions (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>80</td>
<td>Security product token for the source issuer of the RUNCMD, if available.</td>
</tr>
<tr>
<td>96</td>
<td>8</td>
<td>The service point name entered on the RUNCMD.</td>
</tr>
<tr>
<td>104</td>
<td>8</td>
<td>The service point application entered on the RUNCMD.</td>
</tr>
<tr>
<td>112</td>
<td>8</td>
<td>The NETID entered on the RUNCMD.</td>
</tr>
<tr>
<td>120</td>
<td>2</td>
<td>Length of the service point command string.</td>
</tr>
<tr>
<td>122</td>
<td>Variable</td>
<td>The service point command string. The length of the string varies, dependent on the individual command string.</td>
</tr>
</tbody>
</table>

- Other DSIUSE fields passed are: USERLU, USEROPID, USERSWB and USERTVB. The value of the USERPDB field is set to zero (0).

Return Codes: DSIEX19 can pass these return codes:

USERASIS
Continue RUNCMD processing to send the command to the service point.

Any return code other than USERASIS
Discontinue RUNCMD processing. Message BNH192E is to be issued indicating that processing of the RUNCMD has stopped.

DSIEX20: SAW Exit

This exit is available only through an assembler interface. Refer to Tivoli NetView for z/OS Customization: Using Assembler for more information.

DSIEX21: Encryption Key for DSITCPRF Installation Exit

The DSIZKNYJ command is used to edit encrypted definition member DSITCPRF in DSIPRF. Refer to the Tivoli NetView for z/OS Security Reference for more information.

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.

Return Codes: To initialize the BSAM data set or file, return the USERSWAP return code and set the command buffer to the record to be used. A return code other than USERSWAP causes the DST to end.

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.
HLL Installation Exit Routines

**Coding Considerations:** XITBO can use only the service facilities available to the DST.

Avoid coding installation exits for frequently executed functions, such as BSAM I/O, because performance can be degraded significantly.

**XITCI: CNM Interface Input**

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

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 the NetView program routing.

Control point management services units (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.

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

**Table 7. Routing of Network Services Request Units**

<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>BNJDSERV</td>
</tr>
<tr>
<td>RECFMS</td>
<td>X'410384'</td>
<td>AAUTSKLP, BNJDSERV</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, BNJDSERV, DSIGDS</td>
</tr>
<tr>
<td>CP-MSU</td>
<td>X'1212'</td>
<td>BNJDSERV, DSIGDS</td>
</tr>
<tr>
<td>MDS-MU</td>
<td>X'1310'</td>
<td>BNJDSERV, DSIGDS</td>
</tr>
<tr>
<td>Cross-domain alert</td>
<td>X'1040'</td>
<td>BNJDSERV</td>
</tr>
</tbody>
</table>

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

**Table 8. Routing of RUs by Major Vector Key**

<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>BNJDSERV</td>
</tr>
<tr>
<td>X'0001'</td>
<td>Link event</td>
<td>BNJDSERV</td>
</tr>
<tr>
<td>X'0002'</td>
<td>Resolution</td>
<td>BNJDSERV</td>
</tr>
</tbody>
</table>
The focal point transfer RU header is part of the communication network management (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.

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

<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 NetView 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 NetView 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 network management vector transport (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 always X’1040’. The 16th byte contains the length of the 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.

For more information about the format of a specific RU, refer to the SNA library and NCP and EP Reference Summary and Data Areas.

Return Codes: XITCI can use two other return codes in addition to USERASIS, USERDROP, and USERSWAP.

USEREXLG (252)
The hardware monitor, executing under task BNJDSERV, records the
message only to system management facilities (SMF) 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 more information about 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 alert. 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 SNA-MDS forwarded alerts from non-NetView entry points, only event data is recorded to the database; this processing is the same as for local alerts that are not forwarded. For NV-UNIQ/LUC alert forwarding protocol forwarded alerts and 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 the 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.

Refer to the RATE and AUTORATE statements in the [Tivoli NetView for z/OS Administration Reference](#) for 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. If a substitute buffer is returned, the data must be a valid SNA RU.

Refer to the SNA library for a description of RU formats.

**XITDI: Data Services Task Initialization**

*Description:* The DST calls XITDI for each statement read by the DST during initialization. When the end of file is reached, this installation exit is entered and the length of the input command buffer is 0. You can code up to 10 module names for each user-written exit routine. See “Chapter 3. HLL Data Services Command Processors” on page 27 for more information about XITDI during DST initialization.
Example of Use: You can add XITDI to the DST initialization deck to provide user initialization values to DST initialization.

Coding Considerations: Do not replace NetView-provided DST XITDI exits.

XITDI can use only the service facilities available to the DST subtask.

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 prevent the DST from processing a definition statement by passing return code USERDROP.

When XITDI is called for an end-of-file situation, a nonzero return code indicates that the DST should be stopped.

**XITVI: VSAM Input**

Description: The DST calls XITVI after a CNMKIO call for input is issued. The record has been 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 has been retrieved from a VSAM data set or file.

Coding Considerations: XITVI can use only the service facilities available to the DST.

Avoid coding installation exits for frequently executed functions, such as VSAM I/O, because performance can be degraded significantly.

**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. The NetView program provides its own XITVN for VSAM logs generated under DST. You should code this installation exit only if you wish to write your own VSAM subtask using DST as a base.

Coding Considerations: XITVN can use only the service facilities available to the DST.

Notes:
1. Only VSAM key-sequenced data sets are supported.
2. Do not replace 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 set the command buffer to 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 through the CNMKIO service.

*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 text portion is mapped by DSILogDS when using this exit for the DSILog task.

Avoid coding installation exits for frequently executed functions, such as VSAM I/O, because performance can be degraded significantly.

XITXL: External Logging

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

*Example of Use:* You can use XITXL to write user-defined data to a user-defined log.

*Coding Considerations:* XITXL can use only the service facilities available to the DST.

You can use the following offsets (in byte values) to access the CMDBUF (Cmdbuf):

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>ELBLENG</td>
<td>Unsigned length of header</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ELBRLENG</td>
<td>Unsigned length of record</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>ELBTYPE</td>
<td>Log type</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>ELBLOG</td>
<td>EBCDIC log type</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Reserved</td>
<td>Reserved by NetView</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Start of record</td>
</tr>
</tbody>
</table>
Chapter 3. HLL Data Services Command Processors

HLL command processors that use the CNMCNMI and CNMKIO services must run under a DST. The DST provides the underlying interfaces required by both CNMCNMI and CNMKIO.

A DST is a set of NetView interfaces built on top of the NetView optional task base. The NetView optional task is described in [Tivoli NetView for z/OS Customization: Using Assembler](#). A DST provides a subtask processing module (DSIZDST) along with:

- An initialization exit interface
- A data services command processor (DSCP) interface that provides support for VSAM (through CNMKIO) and CNMI (through CNMCNMI)
- Various installation exit interfaces. See [Table 4 on page 13](#).

For more information about the TASK and DSTINIT statements referenced in this chapter, refer to [Tivoli NetView for z/OS Administration Reference](#).

Installing the Data Services Task

To install the data services task, you must code a TASK statement for the DST subtask in the CNMSTYLE member of the DSIPARM data set. The TASK statement is in the format:

```
TASK. taskname.MOD=DSIZDST
TASK. taskname.MEM=usermem
TASK. taskname.PRI=n
TASK. taskname.INIT=Y|N
```

Where:

- `taskname` Specifies the name of the task.
- `MOD` Specifies the subtask processing module. You must specify DSIZDST as the subtask processing module. DSIZDST is provided by the NetView program and provides the necessary initialization, processing, and termination routines to use the DSCP interfaces.
- `MEM` Specifies the user-defined initialization member found in DSIPARM to be used by this task. The initialization data set member must contain DSTINIT statements to provide initialization operands required by DSIZDST. The statements are described under their respective interfaces. The initialization exit can also process user-defined statements.
- `PRI` Specifies the relative task priority (1–9). 1 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 through the START command only (INIT=N).

When the DST is started, the initialization data set specified by the MEM keyword on the TASK statement is read, and the DSTINIT statements are processed.

The following DSTINIT keywords are related to initialization:
HLL Data Services Command Processors

- **FUNCT** - Specifies which DST services are required. In all cases, the ability to call HLL DSCPs is provided. The function choices are:
  - **OTHER** The DST does not require the CNMI or VSAM interfaces.
  - **BOTH** Both the VSAM and CNMI interfaces are required.
  - **CNMI** Only the CNMI 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 documented in “[Chapter 2. HLL Installation Exit Routines” on page 11]. This exit is called once for every statement in the specified initialization member (MEM keyword of TASK statement). When the end of file has been reached, the length of CMDBUF (Cmdbuf) is 0.

  For each statement (except end-of-file condition), the standard installation exit return codes cause these 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.

  - **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 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 fails.

---

Data Services Command Processors

Command processors running under DSTs are called data services command processors (DSCP) and must be defined as TYPE=D (DST only) or TYPE=RD (regular or DST). The next sections describe services that are available to DSCPs.

CNM Data Services

You must define an APPL definition with AUTH=CNM to VTAM for the DST (use the TSKID name as the APPL name). The DST provides access to both solicited and unsolicited CNM data. You can use CNMCNMI to solicit CNM data from the network. You can define an HLL DSCP to receive unsolicited CNM data from VTAM.

**Unsolicited CNM Data Interface**

VTAM provides a default table (ISTMGC01) that controls the routing of unsolicited CNM RUs. You can write a supplemental table (ISTMGC00) to override the default routing information provided by VTAM. 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 CNM services, an access method control block (ACB) is opened with an ACB name (the application name) equivalent to the task name as defined by the TSKID operand in the DST TASK definition statement. (The one 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.

The following DSTINIT keywords are related to the unsolicited CNM data interface:

- **UNSOL** - Specifies the command verb name of the module that is to serve as the unsolicited DSCP for this DST. The unsolicited DSCP should not issue the CNMCNMI macro, but can issue the CNMKIO macro.
- **DSRBU** - Specifies the number of unsolicited data services request blocks (DSRBs) that are to be allocated to this DST. If unsolicited CNM data will not be processed by this DST, set this value to 0; otherwise set it to 1.

When the unsolicited HLL DSCP receives control, CNMdbuf contains the unsolicited data RU.

**Solicited CNM Data Interface:** An HLL DSCP can use CNMCNMI to acquire communications network management data from the network.

The DSTINIT keyword DSRBO specifies the number of solicited DSRBs that are required by this task and limits the number of concurrent CNMCNMI or CNMKIO requests, or both. This value must be at least 1 and no greater than 862. (A DSCP is not invoked until a solicited DSRB is available.)

**VSAM Services**

A DSCP can invoke the CNMKIO service routine to perform input or output for a specified VSAM data set. The following DSTINIT keywords are related to CNMKIO service routine:

- **DSRBO** - Specifies the number of solicited DSRBs that are required by this task, and limits the number of concurrent CNMCNMI or CNMKIO service routines, or both. This value must be at least 1 and no greater than 862. (A DSCP is not invoked until a solicited DSRB is available.)
- **PDDNM** - Specifies the ddname of the primary data set to be used by VSAM services.
- **PPASS** - Specifies the VSAM password to be used when the primary data set ACB is opened.
- **SDDNM** - Specifies the ddname of the secondary data set to be used by VSAM services. Use the NetView SWITCH command to control which data set is 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.
- **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.
- **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.
- **XITVO** - Specifies an installation exit to receive control before output of a record to the VSAM data set.
HLL Data Services Command Processors

User-Defined Services

HLL command processors defined as TYPE=D or TYPE=RD can be invoked under the DST to perform user-defined functions in addition to CNMKIO or CNMCNMI functions.

Scheduling Commands under the DST

You use the CNMSMSG service routine to schedule a DSCP and, in conjunction with the WAIT command, to wait for the DSCP to send back the results of the scheduled work. For samples of DSCPs and installation exit routines, see "Appendix B. PL/I Samples" on page 321 for PL/I, and "Appendix D. C Samples" on page 345 for C.
Part 2. Enabling a High-Level Language Program

Chapter 4. Preinitialized and Non-Preinitialized Environments

Environments ................................................. 33
Languages Supported ................................. 33
Advantages and Disadvantages of Preinitialized Environments .......... 33
Steps for Implementing Command Processors and Installation Exits .... 34
Non-Preinitialized Environments ................. 34
Preinitialized Environments ......................... 35
HLL Definition Facilities ................................. 36
Interface Modules ............................................. 36
HLLENV Command ............................................. 36
Defining Preinitialized Environments .......... 37
Storage Keywords ............................................ 37
REGENVS and CRITENVS Keywords ............. 37
DEFAULT Keyword of the HLLENV Command ........ 38
HLL Run-Time Options ................................. 38
Using the HLL Run-Time Options for Preinitialized Environments .... 40
Examples ................................................. 41
Non-Preinitialized Example .......................... 41
Preinitialized Example ................................. 41

Chapter 5. Compiling, Link-Editing, and Running Your HLL Program .... 43
Compiling .................................................. 43
Link-Editing ................................................. 43
Running ...................................................... 45
Chapter 4. Preinitialized and Non-Preinitialized Environments

The NetView program supports preinitialized and non-preinitialized environments for PL/I and C programs.

If you use non-preinitialized environments, a unique environment is set up and freed each time a program is run.

Preinitialized environments significantly improve the performance of command processors and installation exits. Instead of repeatedly allocating and deallocating environments for each invocation of a command processor or installation exit, the NetView program manages a global pool of preinitialized environments. Command processors or installation exits are then able to enter and exit these environments more efficiently.

The NetView program supports regular and critical preinitialized environments. Critical preinitialized environments are reserved for programs that should always run in a preinitialized environment. Regular preinitialized environments are available to all preinitialization-enabled programs.

Languages Supported

Non-preinitialized environments are supported for:
• OS PL/I V2R3M0
• C/370 V2R1M0
• AD/Cycle PL/I
• AD/Cycle C/370

Preinitialized environments are supported for:
• AD/Cycle PL/I
• AD/Cycle C/370

Note: Non AD/Cycle PL/I and C compiled modules must be linked with LE/390.

Advantages and Disadvantages of Preinitialized Environments

The steps required to run your program in a non-preinitialized environment are fewer than those required to run with preinitialized environments. But, by running non-preinitialized you lose the performance enhancements available with preinitialized environments.

Preinitialized environments have a predefined set of options and are allocated a predefined initial amount of storage which is increased in fixed increments. If your program requires different run-time options, it cannot be run preinitialized. For information about run-time options, see “Chapter 6. Coding HLL Command Processors and Installation Exits” on page 49 and “Chapter 8. Coding Your C Program-Interfaces and Restrictions” on page 119. For information about defining the size of your preinitialized environments, see “HLLENV Command” on page 36.

Programs that are good candidates for running in a preinitialized environment include:
• Installation exits
Preinitialized and Non-Preinitialized Environments

- Frequently used command processors that run for a brief period of time such as those called from the NetView automation table.

Programs should run in a non-preinitialized environment if:
- Initialization overhead is significantly smaller than the overall run-time of the program.
- The program is used infrequently.
- The program needs to run with significantly more ISA, STACK, or HEAP storage than that defined for the preinitialized environment. When your program runs in a preinitialized environment, the HLLENV PISA, PSTACK, and PHEAP values for the preinitialized environment are used instead of the values defined in your program. See "HLLENV Command" on page 36 for more information.
- It is an IBM C/370 V2R1M0 program.
- It is an OS PL/I V2R3M0 program that:
  - Uses language specific file I/O, excluding stream-oriented output to SYSPRINT.
  - Uses CONTROLLED variables.
  - Uses FETCH and RELEASE statements.
- It is an AD/Cycle PL/I program that:
  - Uses the STOP statement.
  - Invokes an assembler routine containing SVC LINK.
- It is an AD/Cycle C program that:
  - Uses the exit() statement.
  - Invokes an assembler routine containing SVC LINK.

Note: Language and storage restrictions are required so that different programs can reuse the same environment. For more information about restrictions resulting from shared use of preinitialized environments, refer to the OS PL/I Version 2 library for PL/I V2R3M0 or the IBM Language Environment® for MVS and VM library for AD/Cycle PL/I and AD/Cycle C.

Steps for Implementing Command Processors and Installation Exits

The following are high-level steps to help you understand the process for creating programs to run in non-preinitialized and preinitialized environments. Information on the individual facilities and commands can be found in "HLL Definition Facilities" on page 36.

Non-Preinitialized Environments

The following steps create and run a program in a non-preinitialized environment:
1. Write the command processor or installation exit code using PL/I or C.
2. Add the appropriate HLOOPTS that specifies whether or not your program:
   - Accepts queued input.
   - Terminates on CANCEL/RESET.
3. Compile your program.
4. Link-edit your program with DSIHSTUB and the interface module appropriate to your language environment. See Table 11 on page 36.
5. Run your program.
Preinitialized Environments

The following steps create and run a program in a preinitialized environment:

1. Design and implement the preinitialized environments for your system:
   a. Define the size of each preinitialized environment using HLLENV PISA, PSTACK, and PHEAP.
   b. Define the appropriate number of regular preinitialized environments for your system using HLLENV REGENVS.
   c. Define the maximum number of critical preinitialized environments required for your system using HLLENV CRITENVs.
   d. Specify whether preinitialization-enabled programs will default to run in a preinitialized environment (HLLENV DEFAULT=PREINIT) or in a non-preinitialized environment (HLLENV DEFAULT=NOTPREINIT) unless the default is overridden within individual programs using HLLOPTS.

2. Code your command processor or installation exit using PL/I or C.

3. Add the appropriate HLLOPTS that specifies whether or not your program:
   - Accepts queued input.
   - Terminates on CANCEL/RESET.
   - Overrides the HLLENV DEFAULT=PREINIT or HLLENV DEFAULT=NOTPREINIT setting for your system.
   - Is critical to run in a preinitialized environment.

4. Compile your program.
   For preinitialization-enabled PL/I programs, ensure your program is compiled with the SYSTEM(MVS) compile option. You can do this in one of the following ways:
   - Compile on MVS and allow the compiler to default to SYSTEM(MVS).
   - Explicitly specify SYSTEM(MVS) with the PROCESS statement in your PL/I source. For example, include:
     ```pli
     *PROCESS SYSTEM(MVS);
     ```
   - Specify SYSTEM(MVS) in the EXEC statement of the compiler JCL.

   For preinitialization-enabled C programs, ensure that your program is compiled with the TARGET(MVS) compile option. You can do this in one of the following ways:
   - Compile on MVS and allow the compiler to default to TARGET(MVS).
   - Explicitly specify TARGET(MVS) with the #pragma options preprocessor directive in your C source. For example, include:
     ```c
     #pragma options (TARGET(MVS))
     ```
   - Specify TARGET(MVS) in the EXEC statement of the compiler JCL.

5. Link-edit your program with DSIHSTUB and the interface module appropriate to your language environment. (See Table 11 on page 36.) Programs linked with DSIHSTUB and one of DSIEXPLP, DSIEXAPP, or DSIEXAPC are considered preinitialization-enabled.

6. Run your program. Whether your program runs in a preinitialized mode, depends on your system settings for HLLENV DEFAULT, the HLLOPTS defined in the program, and the available preinitialized environments.
Preinitialized and Non-Preinitialized Environments

HLL Definition Facilities

The NetView program provides the facilities that allow you to define preinitialized environments. Combinations of the following facilities enable your programs to run as preinitialized command processors or installation exits. However, you can run the programs in either preinitialized or non-preinitialized environments:

- Interface Modules
- HLLENV
- HLLOPTS

Interface Modules

To identify your program to the NetView program as a command processor or installation exit, link-edit your program load-module with DSIHSTUB. DSIHSTUB must be the entry point.

Also, link-edit your program with the interface module appropriate to your program type.

Notes:

1. You can link your modules with preinitialization-enabled interface modules and then use HLLOPTS and HLLENV to control whether the modules run preinitialized or non-preinitialized. You can also explicitly link your modules with a non-preinitialized interface module if you want the module to always run non-preinitialized.

2. Preinitialization with LE/390 is only supported for AD/Cycle PL/I and AD/Cycle C/370.

Table 11 lists supported environments and their associated interface modules.

<table>
<thead>
<tr>
<th>Compiler and Program Type</th>
<th>Library</th>
<th>Interface Module</th>
<th>HLLENV TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I V2R3M0 Non-preinitialized</td>
<td>LE/390</td>
<td>DSIEXANP</td>
<td></td>
</tr>
<tr>
<td>AD/Cycle PL/I Non-preinitialized</td>
<td>LE/390</td>
<td>DSIEXANP</td>
<td></td>
</tr>
<tr>
<td>AD/Cycle PL/I Preinitialization-enabled</td>
<td>LE/390</td>
<td>DSIEXAPP IBMADPLI</td>
<td></td>
</tr>
<tr>
<td>C/370 V2R1M0 Non-preinitialized</td>
<td>LE/390</td>
<td>DSIEXANC</td>
<td></td>
</tr>
<tr>
<td>AD/Cycle C/370 Non-preinitialized</td>
<td>LE/390</td>
<td>DSIEXANC</td>
<td></td>
</tr>
<tr>
<td>AD/Cycle C/370 Preinitialization-enabled</td>
<td>LE/390</td>
<td>DSIEXAPC IBMADC</td>
<td></td>
</tr>
</tbody>
</table>

For an example of link-edit JCL, see “Chapter 5. Compiling, Link-Editing, and Running Your HLL Program” on page 43.

HLLENV Command

Preinitialized environments are defined and managed by the NetView HLLENV command. HLLENV creates global pools of preinitialized environments that can be used and then returned when programs using them terminate.
The HLLENV command defines and manages these types of preinitialized environments:
- AD/Cycle PL/I
- AD/Cycle C

For more information about the HLLENV command, refer to the NetView online help.

For each language, you can define preinitialized environments using the HLLENV command. The HLLENV TYPE keyword specifies definitions for the following languages:

- IBMADPLI: HLLENV specifications are for AD/Cycle PL/I programs.
- IBMADC: HLLENV specifications are for AD/Cycle C programs.

**Defining Preinitialized Environments**

When defining preinitialized environments, consider how many environments should be allocated, the ISA, STACK, and HEAP storage required, and how preinitialization is specified in individual programs.

**Storage Keywords**

HLLENV keywords enable you to customize the amount of initial storage allocated for preinitialized environments.

For AD/Cycle PL/I and AD/Cycle C, the HLLENV PSTACK and PHEAP keywords are used to specify the STACK and HEAP storage to be allocated for preinitialized environments. The HLLENV PSTACK and PHEAP values override STACK or HEAP values declared in the program.

See "Chapter 6. Coding HLL Command Processors and Installation Exits" on page 49 and "Chapter 8. Coding Your C Program-Interfaces and Restrictions" on page 119 for more information about PISA, STACK, PHEAP, and HEAP and how they apply to the PL/I and C environments.

**REGENVS and CRITENVS Keywords**

The REGENVS and CRITENVS keywords of the HLLENV command let you specify two pools of preinitialized environments for each language environment.

The REGENVS keyword specifies the number of preinitialized environments to be immediately allocated. Environments defined with REGENVS are retained by NetView and are available to all preinitialization-enabled programs of the applicable TYPE.

The CRITENVS keyword specifies the maximum number of preinitialized environments that can be allocated and made available to critical preinitialized programs. Critical preinitialized programs are defined using HLOPTS. See "Chapter 4. Preinitialized and Non-Preinitialized Environments" on page 38 for more information about HLOPTS.

The following sections describe how preinitialized environments created with the REGENVS and CRITENVS keyword are distributed.

**For Programs That Should Run in a Preinitialized Environment:** The program uses one of the preinitialized environments from the global pool set up with the REGENVS keyword of the HLLENV command. If an environment is not available, the program runs non-preinitialized and initializes its own environment resulting in no performance benefit.
For Critical Preinitialized Programs: The program first tries to use one of the preinitialized environments from the global pool allocated with the HLLEnv REGENVS command. If an environment is not available in the global pool, the program tries to use an environment allocated with the HLLEnv CRITENVS command.

If an environment is still not available, the program runs non-preinitialized. However, if the number of environments allocated for critical preinitialized programs does not exceed the value specified with the CRITENVS keyword of the HLLEnv command, a newly created preinitialized environment will be allocated for other critical programs to use. This makes it more likely that a preinitialized environment will be available the next time a critical program runs.

Note: Even when a program is defined as critical, there is no guarantee that it will run in a preinitialized environment. For example, if the number of preinitialized environments in use equals the total number of environments defined for both the REGENVS and CRITENVS keywords of HLLEnv, your critical program will not run in a preinitialized environment.

DEFAULT Keyword of the HLLEnv Command
The DEFAULT keyword specifies whether all programs link-edited with an interface module for a preinitialized environment (DSIEXAPP or DSIEXAPC) will run in a preinitialized environment.

Use the DEFAULT=PREINIT keyword on the HLLEnv command to specify that all preinitialization-enabled programs should run in a preinitialized environment. Use the DEFAULT=NOTPREINIT keyword on the HLLEnv command to specify none of your preinitialization-enabled programs should run in a preinitialized environment. Whether an individual program runs in a preinitialized environment also depends on the HLLOPTS settings for bits 2, 3, and 4.

You can override DEFAULT=PREINIT for individual programs by setting bit 2 in the HLL run-time options (HLLOPTS) to 1. When this bit is set to 1, the program will not run preinitialized even though DEFAULT=PREINIT is specified.

You can override DEFAULT=NOTPREINIT for individual programs by setting bit 3 in HLLOPTS to 1. When this bit is set to 1, the program will run preinitialized even though DEFAULT=NOTPREINIT is specified.

HLL Run-Time Options
You can specify HLL run-time options when coding your program by declaring and initializing the external variable named HLLOPTS. If you do not code HLL run-time options, the default of all bits being set to zero (0) is assumed. The bits defined in HLLOPTS are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HLL_QUEUED_INPUT</td>
<td>Determines whether an HLL program accepts queued input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refer to the QUEUE command in the NetView online help for more information.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>HLL program does not accept queued input.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>HLL program accepts queued input.</td>
</tr>
</tbody>
</table>
Table 12. Bits Defined in HLLOPTS (continued)

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HLL_NO_CANCEL</td>
<td>Determines whether an HLL program terminates on CANCEL/RESET. Refer to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESET command in the NetView online help for more information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Cancelable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Noncancelable.</td>
</tr>
<tr>
<td>2</td>
<td>OVER_DEFAULT_PREINIT</td>
<td>Overrides the DEFAULT=PREINIT setting on the HLLENV command. When</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEFAULT=PREINIT, bit 2 specifies:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Programs run in a preinitialized environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Programs run in a non-preinitialized environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Notes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. This bit is ignored if specified in a C/370 V2R1M0 program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. This bit is only checked when DEFAULT=PREINIT is specified on the HLENV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>command.</td>
</tr>
<tr>
<td>3</td>
<td>OVER_DEFAULT_NOTPREINIT</td>
<td>Overrides the DEFAULT=NOTPREINIT setting on the HLLENV command. When</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEFAULT=NOTPREINIT, bit 3 specifies:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Programs run in a non-preinitialized environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Programs run in a preinitialized environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Notes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. This bit is ignored if specified in a C/370 V2R1M0 program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. This bit is only checked when DEFAULT=NOTPREINIT is specified on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HLENV command.</td>
</tr>
<tr>
<td>4</td>
<td>HLL_CRIT_PREINIT</td>
<td>Specifies that a critical program should run in a preinitialized environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = It is not critical for the program to run in a preinitialized environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = It is critical for the program to run in a preinitialized environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Notes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. This bit is ignored if specified in a C/370 V2R1M0 program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. This bit is only checked if the program is to run preinitialized. That</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is, if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DEFAULT=PREINIT is specified on the HLLENV command and the HLLOPTS bit 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is set to zero (0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DEFAULT=NOTPREINIT is specified on the HLLENV command and the HLLOPTS bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 is set to 1.</td>
</tr>
<tr>
<td>5-31</td>
<td>Reserved for</td>
<td>Internal use. Do not assign any values to these fields.</td>
</tr>
</tbody>
</table>

For example, to override the default HLL run-time options in an HLL program and make the program noncancelable, include:

PL/I Program

DCL HLLOPTS BIT(32) STATIC EXTERNAL
  INIT('01000000000000000000000000000000'B);
#pragma variable(HLLOPTS,NORENT)
extern unsigned int HLLOPTS = 0x40000000;

For C, the #pragma variable preprocessor directive indicates that the variable named
HLLOPTS is to be used in a non-reentrant fashion. This directive does not affect
the reentrance of the HLL program.

Using the HLL Run-Time Options for Preinitialized
Environments

The HLL run-time options for preinitialized environments (bits 2, 3, and 4) are
used with the DEFAULT keyword of the HLLENV command. The DEFAULT
keyword specifies that either all or none of the preinitialization-enabled programs
run in a preinitialized environment. When you have chosen a DEFAULT value,
individual programs can set bit 2 or 3 to override the DEFAULT value.

Set bit 4 in HLLOPTS for preinitialization-enabled programs where it is critical that
the programs run in a preinitialized environment. Programs defined this way are
 termed critical preinitialized programs.

Table 13 summarizes the interaction of bits 2, 3, and 4 in HLLOPTS and the
DEFAULT values on the HLLENV command.

<table>
<thead>
<tr>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>HLLENV DEFAULT=PREINIT</th>
<th>HLLENV DEFAULT=NOTPREINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Preinitialized</td>
<td>Non-preinitialized</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Critical preinitialized</td>
<td>Non-preinitialized</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Preinitialized</td>
<td>Preinitialized</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Critical preinitialized</td>
<td>Critical preinitialized</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Non-preinitialized</td>
<td>Non-preinitialized</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Non-preinitialized</td>
<td>Non-preinitialized</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Non-preinitialized</td>
<td>Preinitialized</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Non-preinitialized</td>
<td>Critical preinitialized</td>
</tr>
</tbody>
</table>

The following combinations of settings for HLLOPTS bits 2, 3, and 4, are important
to highlight:

- If you want a preinitialization-enabled program to run in a preinitialized
  environment, regardless of the DEFAULT value on the HLLENV command, set
  bits 2, 3, and 4 to 010. For example, include:

  PL/I Program

  DCL HLLOPTS BIT(32) STATIC EXTERNAL
       INIT('00010000000000000000000000000000'B);

  C Program

  #pragma variable(HLLOPTS,NORENT)
  extern unsigned int HLLOPTS = 0x10000000;

- If you want a preinitialization-enabled program to run as a critical preinitialized
  program regardless of the DEFAULT value on the HLLENV command, set bits 2,
  3, and 4 to 011. For example, include:
Preinitialized and Non-Preinitialized Environments

**PL/I Program**

DCL HLLOPTS BIT(32) STATIC EXTERNAL
   INIT('00011000000000000000000000000000'B);

**C Program**

#pragma variable(HLLOPTS,NORENT)
extern unsigned int HLLOPTS = 0X18000000;

- If you want a preinitialization-enabled program to run in a non-preinitialized environment regardless of the DEFAULT value on the HLLENV command, set bits 2, 3, and 4 to 100. For example, include:

**PL/I Program**

DCL HLLOPTS BIT(32) STATIC EXTERNAL
   INIT('00100000000000000000000000000000'B);

**C Program**

#pragma variable(HLLOPTS,NORENT)
extern unsigned int HLLOPTS = 0X20000000;

---

**Examples**

The following sections show various examples of setting up programs to run in preinitialized and non-preinitialized environments.

**Non-Preinitialized Example**

In this example, a command processor is written in PL/I V2R3M0, which must accept queued input. Because you are using a non-AD/Cycle compiler, you must run it in a non-preinitialized environment. No other programs in your system require preinitialization, so it is not necessary to set up preinitialized environments. The following steps complete the example:

1. Write the command processor or installation exit code.
   Include in your PL/I code the following HLLOPTS external declaration specifying that your command processor will accept queued input:
   
   DCL HLLOPTS BIT(32) STATIC EXTERNAL
   INIT('10000000000000000000000000000000'B);

2. Compile the program.

3. Link-edit the program with DSIHSTUB and DSIEXANP.

4. Run your program.

**Preinitialized Example**

In the next example, all command processors are to run in preinitialized environments to improve performance. Most of your programs will require 8192 bytes of STACK and HEAP storage. You estimate that 5 preinitialized environments and 3 additional environments for critical programs will be required.

The first command processor you write is in AD/Cycle PL/I. You want this command processor to accept all defaults for HLLOPTS. Although you prefer that the program run in a preinitialized environment, you do not consider it to be critical. The following steps complete this example:

1. Define the preinitialized environments for AD/Cycle PL/I using the HLLENV command:
   
   HLLENV CHANGE,REGENVS=5,CRTENVS=3,PSTACK=8192,PHEAP=8192,
   DEFAULT=PREINIT,TYP=IBMADPLI
Preinitialized and Non-Preinitialized Environments

2. Code the command processor or installation exit.
3. Compile the program with the SYSTEM(MVS) compile option.
4. Link-edit the program with DSISHSTUB and DSIXAPP.
5. Run the program.

In the next example, the AD/Cycle PL/I command processor requires different STACK and HEAP storage from that defined using the PSTACK and PHEAP keywords on the HLLENV command. The new command processor cannot run in a preinitialized environment because it requires more initial storage than is defined for preinitialized programs. The following steps complete the example:

1. Compile the program with the SYSTEM(MVS) compile option.
2. Link-edit the program with DSISHSTUB and DSIXAPP.
3. Run the program.

In another example, you decide to start writing command processors using AD/Cycle C/370. For the AD/Cycle C/370 preinitialized environment, you estimate 3 preinitialized environments and 2 additional preinitialized environments for critical programs will be required. For AD/Cycle C/370 programs, you decide to set the default to run all programs in a non-preinitialized environment. However, your first command processor written in AD/Cycle C/370 is performance sensitive and you feel that it is critical that it run in a preinitialized environment. Take the following steps:

1. Define the preinitialized environments for AD/Cycle C/370 using the HLLENV command:
   
   HLLENV CHANGE,REGENVS=3,CRITENVS=2,DEFAULT=NOTPREINIT,TYPE=IBMADC

2. Write the command processor or installation exit code.
   a. Include in your C code the following HLLOPTS external declaration specifying that it is critical that your command processor run preinitialized. The following HLLOPTS specification overrides your HLLENV DEFAULT=NOTPREINIT command and specifies that your program is critical.
   
   #pragma variable(HLLOPTS,NORENT)
   extern unsigned int HLLOPTS = 0X18000000;

   b. Define the run-time options required for your program.

3. Compile the program with the TARGET(MVS) compile option.
4. Link-edit the program with DSISHSTUB and DSIXAPC.
5. Run the program.
Chapter 5. Compiling, Link-Editing, and Running Your HLL Program

If you have a PL/I or C compiler installed, you can modify the compile and link-edit JCL for use with the NetView program. This chapter describes how to make these modifications.

Use the examples provided in this chapter to modify the compile and link-edit JCL samples that you received with your compiler.

Note: You must have completed the installation steps for HLL before attempting to invoke programs in the NetView environment.

Compiling

To compile programs using NetView services, modify the compile step in the JCL to reference the NetView macro libraries and include a SYSLIB statement for SYS1.MACLIB. An example of modifications to the compile step JCL for PL/I follows:

```
//COMPILE EXEC PGM=IEL0AA,REGION=1000K,
// PARM='OBJECT,MACRO,LIST,RENT'

//SYSLIB DD DSN=NETVIEW.V1R4M0.MACLIB,DISP=SHR
DSN=SYS1.MACLIB,DISP=SHR
```

The following example shows how to modify the compile step JCL for C:

```
//COMPILE EXEC PGM=EDCCOMP,PARM=('RENT'),REGION=&CREGSIZ

//SYSLIB DD DSN=NETVIEW.V1R4M0.MACLIB,DISP=SHR
DSN=SYS1.MACLIB,DISP=SHR
```

Note: When you compile PL/I programs, you can ignore message IEL0548I.

Link-Editing

Consider the following when you link-edit modules:

- All load modules must be reentrant.
- Load modules can reside in 24- or 31-bit storage and can be entered in either addressing mode.
- You must link-edit all load modules with DSIHSTUB and the appropriate interface module; DSIHSTUB must be the entry point.
Compiling, Link-Editing, and Running Your HLL Program

To link-edit a module to run with the NetView program, modify the link-edit step in the JCL to reference the appropriate NetView libraries. This enables you to include DSIIHSTUB and the appropriate interface module when link-editing.

Add SYS1.CNMLINK to the list of automatic call libraries defined by SYSLIB in the link-edit step of the JCL. For C, add SYS1.CNMLINK and SYS1.NVULIB.

Figure 7 shows a sample JCL stream for link-editing a PL/I program:

```
//LKED EXEC PGM=IEWL,
// PARM='XREF,RENT,LET,LIST,AMODE=&AMODE,RMODE=&RMODE',
// REGION=4096K,COND=(8,LE,COMPILE)

//SYSLIB DD DSN=SYS1.CNMLINK,DISP=SHR
// DD DSN=Link-Edit Library Dataset,DISP=SHR

INCLUDE SYSLIB(DSIHSTUB)
INCLUDE SYSLIB(Interface Module)
ORDER DSIHSTUB
ENTRY DSIHSTUB
MODE AMODE(31),RMODE(ANY)
NAME LMODNAME(R)
```

Figure 8 shows a sample JCL stream for link-editing a C program:

```
//LKED EXEC PGM=IEWL,
// PARM='XREF,RENT,LET,LIST,AMODE=&AMODE,RMODE=&RMODE',
// REGION=4096K,COND=(8,LE,COMPILE)

//SYSLIB DD DSN=SYS1.CNMLINK,DISP=SHR
// DD DSN=SYS1.NVULIB,DISP=SHR
// DD DSN=Link-Edit Library Dataset,DISP=SHR

INCLUDE SYSLIB(DSIHSTUB)
INCLUDE SYSLIB(Interface Module)
ORDER DSIHSTUB
ENTRY DSIHSTUB
MODE AMODE(31),RMODE(ANY)
NAME LMODNAME(R)
```

Figure 8. Example of Link-Editing a C Program

Link-edit library datasets and interface modules are listed in Table 14.

<table>
<thead>
<tr>
<th>Compiler and Program Type</th>
<th>Library</th>
<th>Link-Edit Library Dataset</th>
<th>Interface Module</th>
<th>HLLENV TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I V2R3M0 Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEELKED</td>
<td>DSIEXANP</td>
<td></td>
</tr>
<tr>
<td>PL/I AD/Cycle Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEELKED</td>
<td>DSIEXANP</td>
<td></td>
</tr>
<tr>
<td>PL/I AD/Cycle Preinitialization-enabled</td>
<td>LE/390</td>
<td>SYS1.SCEELKED</td>
<td>DSIEXAPP</td>
<td>IBMADPLI</td>
</tr>
</tbody>
</table>
Table 14. Link-Edit Libraries and Interface Modules (continued)

<table>
<thead>
<tr>
<th>Compiler and Program Type</th>
<th>Library</th>
<th>Link-Edit Library Dataset</th>
<th>Interface Module</th>
<th>HLLENV TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/370 V2R1M0 Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEELKED</td>
<td>DSIEXANC</td>
<td></td>
</tr>
<tr>
<td>C/370 AD/Cycle Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEELKED</td>
<td>DSIEXANC</td>
<td></td>
</tr>
<tr>
<td>C/370 AD/Cycle reinitialization-enabled</td>
<td>LE/390</td>
<td>SYS1.SCEELKED</td>
<td>DSIEXAPC</td>
<td>IBMADC</td>
</tr>
</tbody>
</table>

All HLL modules must be reentrant.

- For PL/I, include the REENTRANT option on the PL/I procedure statement, then link-edit the resulting object decks with the RENT option.
- For C, invoke the PRE-LINKEDIT step, then link-edit the resulting object decks with the RENT option.

If you want to run your existing non-preinitialized load modules with LE/390, first relink them with the LE/390 run-time libraries.

Running

To invoke a command processor or installation exit in the NetView environment, you must modify your NetView startup procedure to reference the appropriate run-time libraries.

The run-time libraries appropriate for your command processor type are shown in Table 15. You must ensure that the required library is available at run-time. One way to do so is to specify the required run-time library in the STEPLIB of your NetView startup procedure.

Table 15. Run-Time Libraries

<table>
<thead>
<tr>
<th>Compiler and Program Type</th>
<th>Library</th>
<th>Run-Time Library Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I V2R3M0 Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEERUN</td>
</tr>
<tr>
<td>PL/I AD/Cycle Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEERUN</td>
</tr>
<tr>
<td>PL/I AD/Cycle Preinitialization-enabled</td>
<td>LE/390</td>
<td>SYS1.SCEERUN</td>
</tr>
<tr>
<td>C/370 V2R1M0 Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEERUN</td>
</tr>
<tr>
<td>C/370 AD/Cycle Non-preinitialized</td>
<td>LE/390</td>
<td>SYS1.SCEERUN</td>
</tr>
<tr>
<td>C/370 AD/Cycle Preinitialization-enabled</td>
<td>LE/390</td>
<td>SYS1.SCEERUN</td>
</tr>
</tbody>
</table>

HLL command processors require a CMDMDL statement in member DSICMD of the DSIPARM data set. Installation exits are loaded at initialization and should conform to installation exit naming conventions. For more information see “Chapter 2. HLL Installation Exit Routines” on page 11.
Part 3. Writing a PL/I Program

Chapter 6. Coding HLL Command Processors and Installation Exits ......................................................... 49
  Initial Parameters .................................................. 49
  Run-Time Options .................................................. 50
  CEEUOPT Run-Time Option CSECT ................................ 50
  LE/390 Preinitialization Defaults ................................ 50
  PSTACK and PHEAP ................................................ 51
    Example of Overriding PSTACK and PHEAP Values .......... 51
    LEOPTS Static External Variable ................................ 51
    Example of Specifying LEOPTS Values .......................... 51
  Parameters Passed to HLL Service Routines .................... 51
    Pointer Variables ............................................. 52
    Integer Variables ............................................ 53
    Fixed-Length Character Strings ................................ 53
    Varying Length Character Strings .............................. 54
  Control Blocks and Include Files ................................. 55
  PL/I Input and Output Considerations ............................ 56
    PL/I Input and Output Considerations in a Preinitialized Environment ........................................ 57
  PL/I Run-Time Considerations .................................... 57
  Considerations for HLL Command Processors ..................... 58
  Return Codes ..................................................... 58
  Restrictions for HLL Programs Written in PL/I .................. 58
    Restrictions for PL/I Programs Running in a Preinitialized Environment ........................................ 58

Chapter 7. PL/I High-Level Language Services ................................................................. 61
  PL/I Sample Template ............................................. 61
  Data Queue Management ............................................. 63
  Sending Messages .................................................. 64
  Parsing Input Strings Similar to NetView Command List Language ......................................................... 65
  Parsing Input String Similar to REXX ......................... 66
  Parsing Input String in CNMSCAN ............................... 67
  Invoking Synchronous Commands ................................. 70
  Sending Commands ................................................ 70
  Waiting and Trapping ............................................. 71
    Example of Using the TRAP and WAIT Commands (PL/I) ..... 71
    Example of Using the PIPE Command (PL/I) .................... 73
  Retrieving Information .......................................... 75
  Command List Variable Access .................................... 76
    Accessing Common Global Variables (PL/I) .................... 77
    Accessing Stem Variables Using the PIPE Command (PL/I) ........................................ 78
  Using Locks ..................................................... 79
  Operator Input .................................................. 80
  VIEW Command Processor ........................................ 80
  Message Processing .............................................. 82
  Command Authorization Checking ................................ 82
    Command Authorization Checking Using CMDAUTH=TABLE .................. 83
    In DISIPARM(CMDTABLE) ....................................... 83
    Command Authorization Checking Using CMDAUTH=SAF .................. 83

In RACF .......................................................... 83
  Command Authorization Checking Sample Code .................... 85
  Altering Data .................................................... 86
  Storage Access .................................................. 88
  Data Set Access ................................................ 89
  Communicating with Devices ....................................... 90
  Performing I/O on a VSAM File (Keyed File Access) ............. 93
  Coding a DST Installation Exit ................................... 95
  Coding an Installation Exit ....................................... 96
  Coding the WAIT FOR DATA Function ............................. 97
    Requesting Data (PL/I) ........................................ 98
    Sending Data (PL/I) .......................................... 100
  Automating MSUs .................................................. 101
  Translating Code Points ......................................... 102
  Registering Applications with MS Transport and Operations Management ........................................ 103
  Registering Applications with the High Performance Transport ........................................ 106
  Sending an Alert over the MS Transport ......................... 109
  Sending an Alert over the High Performance Transport .......... 110
  Sending an MDB to NetView for Processing ....................... 112
Chapter 6. Coding HLL Command Processors and Installation Exits

This chapter contains the following information for coding HLL command processors and installation exits in PL/I:

- Initial parameters
- HLL run-time options
- PL/I run-time options
- Parameters passed to HLL service routines
- Control blocks and include files
- Input and output considerations
- Restrictions

PL/I command processors and installation exits are supported for OS PL/I V2R3M0 and AD/Cycle PL/I.

Initial Parameters

The following initial parameters are passed to an HLL program upon invocation:

- CMDBUF
- HLBPTR
- ORIGBLCK

"Chapter 7. PL/I High-Level Language Services" on page 61 contains a sample template for coding the main procedure statement and the initial parameter declarations in PL/I.

CMDBUF

A varying length character field that contains the command or message that drives this program.

If this program is driven as an installation exit (other than DSIXEX02A), this field contains the message that drives this exit. If driven as DSIXEX02A, CMDBUF does not contain any useful information, and you must retrieve the message from the initial data queue (IDATAQ).

HLBPTR

A 4-byte pointer field containing the address of the HLB control block (DSIPHLB). The HLB control block is the HLL API interface control block that is used to communicate between the HLL service routines and HLL programs in the NetView environment. This pointer is required on all HLL service routine invocations. The HLB control block is unique for each invocation of an HLL program. NetView automatically inserts HLBPTR for the PL/I macro format.

See "DSIPHLB Control Block" on page 305 for more information.

ORIGBLCK

A 40-byte structure that describes the origin of the request that caused execution of this program. ORIGBLCK is mapped by DSIPORIG.

See "DSIPORIG Control Block" on page 306 for more information.
Run-Time Options

The method you use to specify PL/I run-time options depends on the type of PL/I support you use for your programs. The way you specify PL/I run-time options, therefore, depends on the following:

- Compiler (AD/Cycle)
- Library (LE/390)
- Preinitialization Support (Enabled or Disabled)

In addition, you need to consider information about:

- Running non-preinitialized, either intentionally or due to availability
- Type of run-time option, whether the option is a general option or a storage option.

Use Table 16 as a starting point to familiarize yourself with the specific methods of specifying the run-time options for your program. The table indicates the methods to use based on the interface module that is being link-edited with a program. Following the table, each method for specifying run-time options is described in more detail.

Table 16. Interface Modules and How to Specify PL/I General Run-Time Options

<table>
<thead>
<tr>
<th>Interface Module</th>
<th>Library and Program Type</th>
<th>General Run-Time Options</th>
<th>Storage Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIEXANP</td>
<td>LE/390 non-preinitialized</td>
<td>CEEUOPT</td>
<td>CEEUOPT</td>
</tr>
<tr>
<td>DSIEXAPP</td>
<td>LE/390 non-preinitialized</td>
<td>LE/390 preinitialized defaults or LEOPTS static external variable</td>
<td>PSTACK PHEAP</td>
</tr>
<tr>
<td></td>
<td>LE/390 preinitialization-enabled</td>
<td>LE/390 preinitialized defaults</td>
<td>PSTACK PHEAP</td>
</tr>
</tbody>
</table>

**CEEUOPT Run-Time Option CSECT**

When your program is compiled with either the IBM OS PL/I V2R3M0 or AD/Cycle PL/I compilers, and is used with the LE/390 libraries, the run-time options must be specified using the CEEUOPT CSECT. For more information about the CEEUOPT csect and how to use it to specify run-time options, refer to the LE/390 library.

**LE/390 Preinitialization Defaults**

The following PL/I run-time options are automatically set for AD/Cycle PL/I programs that request to run in a preinitialized environment. Except for the STACK and HEAP values, you cannot change the values. Values for `pstack` and `pheap` are taken from the values specified for PSTACK and PHEAP on the HLLENV command.

- `STACK(pstack,4K,ANY,FREE)`
- `LIBSTACK(8,0,FREE)`
- `ALL31(ON)`
- `NOTEST(ALL,*,*)`
- `BELOWHEAP(8,0,FREE)`
- `THREADHEAP(8,0,ANY,FREE) STORAGE(,,0)`
- `TRAP(OFF)`
- `INTERRUPT(OFF)`
- `MSGQ(1)`
- `PLITASKCOUNT(1)`

```plaintext
STACK(pstack,4K,ANY,FREE)  BELOWHEAP(8,0,FREE)  HEAP(pheap,4K,ANY,FREE,8,0)
LIBSTACK(8,0,FREE)          THREADHEAP(8,0,ANY,FREE) STORAGE(,,0)
ALL31(ON)                   TRAP(OFF)            INTERRUPT(OFF)
NOTEST(ALL,*,*)             MSGQ(1)              PLITASKCOUNT(1)
```
Refer to the Language Environment for MVS & VM library for more information about these options.

**PSTACK and PHEAP**

The AD/Cycle STACK and HEAP storage values are set using the PSTACK and PHEAP keywords of the HLLENV command. The initial values for PSTACK and PHEAP are 4096 bytes each. These values can be changed dynamically with the HLLENV command.

**Example of Overriding PSTACK and PHEAP Values**

The following example shows how to override the initial values for PSTACK and PHEAP for an AD/Cycle PL/I program that requests to run in a preinitialized environment:

```
HLLENV CHANGE,PSTACK=2048,PHEAP=2048,TYPE=IBMADPLI
```

**LEOPTS Static External Variable**

When your program uses the DSIEXAPP interface module and explicitly forces non-preinitialization through the combination of the HLLENV DEFAULT value and the HLLOPTS bits, you may specify the run-time options using the LEOPTS static external variable.

When you define LEOPTS in your program, the run-time options specified are used instead of those described in "LE/390 Preinitialization Defaults" on page 50 and "PSTACK and PHEAP".

**Example of Specifying LEOPTS Values**

You can override the default LE/390 preinitialization, PSTACK, and PHEAP run-time options in a PL/I program as shown in the following LEOPTS declaration.

```
DCL LEOPTS CHAR(255) STATIC EXTERNAL INIT((
  'STACK(1K,2K,ANY,FREE) HEAP(3K,4K,ANY,FREE,8,0) TRAP(OFF) MSGQ(1)'
  ' BELOWHEAP(8,0,FREE) LIBSTACK(8,0,FREE) THREADHEAP(8,0,ANY,FREE)'
  ' INTERRUPT(OFF) NOTEAST(ALL,r,* ) STORAGE(,,0) ALL31(ON)'
  ' PLITASKCOUNT(1)');
```

---

**Parameters Passed to HLL Service Routines**

Four types of parameters can be passed to HLL service routines. Each parameter, as described in "Chapter 11. Service Reference" on page 209, falls into one of four categories:

- Pointer variables
- Integer variables
- Fixed-length character strings
- Varying length character strings

A discussion of each parameter type follows. The sections describe how you can declare, initialize, and pass each parameter type to the HLL service routines. This chapter provides examples and recommendations for writing HLL programs in PL/I.

**Note:** These examples are not complete. They are included to emphasize how you should declare, initialize, and pass the HLL service routine parameters. For complete examples of user-written HLL programs, refer to the HLL samples shipped with the NetView program, or see "Appendix B. PL/I Samples" on page 321.
Pointer Variables

A pointer variable is a 4-byte pointer field containing an address. All HLL service
routines require at least one argument called HLBPTR. The NetView program
calculates the value of HLBPTR and passes it to a PL/I HLL command processor
or installation exit. Therefore, HLBPTR only needs to be declared in PL/I. Do not
assign a value to the HLBPTR variable. This is the only parameter of this type that
you do not have to assign.

Note: You do not need to specify the HLBPTR parameter when coding the HLL
service routine invocation in the PL/I macro format. HLBPTR is inserted for
you before the HLL service routine is invoked.

If an HLL service routine expects an address in a pointer field, you are responsible
for assigning a value to that pointer field before invoking the HLL service routine.
In PL/I, you can use the ADDR function when passing pointer variables to HLL
service routines rather than creating a separate pointer variable for this purpose.
This ensures that the pointer variable has been assigned a value before invoking
the HLL service routine.

Figure 9 shows how to use pointer variables in PL/I. More information for the
numbered steps follow the figure.

DCL VARTOVAR CHAR(8) INIT('VARTOVAR'); /* VARTOVAR constant */

1 DCL HLBPTR PTR; /* HLBPTR MUST BE DECLARED! */
2 DCL SRCPTR PTR; /* Source pointer */
3 DCL DSTPTR PTR; /* Destination pointer */
4 DCL DSTLEN FIXED BINARY(31,0); /* Length of Destination */
5 DCL SRCBUF CHAR(255) VARYING; /* Source buffer */
6 DCL DSTBUF CHAR(255) VARYING; /* Destination buffer */

3 SRCPTR = ADDR(SRCBUF); /* Address of source buffer */
4 DSTPTR = ADDR(DSTBUF); /* Address of destination buffer */
5 DSTLEN = LENGTH(DSTBUF); /* Length of destination buffer */
6 SRCBUF = (255)'A'; /* Initialize source buffer */
7 DSTBUF = (255)' '; /* Initialize destination buffer */

4 CALL CNMCPYS(HLBPTR, SRCPTR, DSTPTR, DSTLEN, VARTOVAR); /* Copy buffer */

Figure 9. Using Pointer Variables in PL/I

The descriptions for the steps shown in Figure 9 are:

1 HLBPTR is declared as a pointer (PTR) variable to be used in the
CNMCPYS invocation. You must not assign a value to HLBPTR. HLBPTR
is specified for this invocation because you invoked CNMCPYS using the
PL/I call format, rather than the PL/I macro format. Chapter 11, Service
Reference" on page 209 contains examples of how to invoke HLL service
routines using the PL/I macro format.

2 SRCPTR is declared as a pointer (PTR) variable.

3 SRCPTR is assigned the address of the source buffer (SRCBUF) to be used
in the CNMCPYS invocation.

4 Both HLBPTR and SRCPTR have been passed as parameters to CNMCPYS.
Using the ADDR function eliminates the need to declare pointer (PTR) variables. Note here the use of a character constant instead of the VARTOVAR variable:

```
CALL CNMCPYS(HLBPTR,ADDR(SRCBUF),ADDR(DSTBUF),DSTLEN,'VARTOVAR');
```

**Note:** If you use the ADDR function to represent a pointer to a varying length character string, message IEL0548I is generated at compile time.

### Integer Variables

Several of the HLL service routines require you to pass a 4-byte integer value to be used as a length, count, queue number, and so on. Figure 10 illustrates the use of integer variables in the PL/I environment. (Descriptions for other statements in this figure are described in "Fixed-Length Character Strings" and "Varying Length Character Strings" on page 54.)

```
%INCLUDE DSIPLI; /* Include the HLL macros */
DCL HLBPTR PTR; /* HLB pointer MUST BE DECLARED */
DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
DCL ORIGBLCK CHAR(40);
DCL SPNAME CHAR(8) VARYING INIT('POOLNAME'); /* Subpool name */
DCL SPFUNC CHAR(8); /* Subpool function */
DCL SPTOKEN FIXED BIN(31,0); /* Subpool token (returned) */
DCL SPLENG FIXED BIN(31,0); /* Cell size */
DCL SPPRICNT FIXED BIN(31,0); /* Number of cells in primary */
DCL SPSECCNT FIXED BIN(31,0); /* Number of cells in secondary */
DCL SPCLASS FIXED BIN(31,0); /* Class of storage */
SPFUNC = 'ALLOC'; /* Function is ALLOCATE */
SPTOKEN = 0; /* Initialize subpool token */
SPLENG = 256; /* Cell size = 256 bytes */
SPPRICNT = 3; /* Primary count = 3 */
SPSECCNT = 2; /* Secondary count = 2 */
SPCLASS = 1; /* Class = 31 bit addressable */
CALL CNMPOOL(HLBPTR,SPFUNC,SPTOKEN,SPNAME,SPLENG,SPPRICNT,SPSECCNT,SPCLASS); /* Allocate subpool */
```

**Figure 10. Using Integer Variables in PL/I**

The descriptions for the steps shown in Figure 10 are:

1. SPTOKEN is declared as a 4-byte integer (FIXED BIN(31,0)).
2. SPLENG is declared as a 4-byte integer (FIXED BIN(31,0)).
3. SPTOKEN is initialized to zero (0). A value is returned in SPTOKEN upon successful completion of the CNMPOOL invocation.
4. SPLENG is assigned a value of 256 to be used in the call to CNMPOOL.
5. SPTOKEN and SPLENG are passed to CNMPOOL. The value of SPTOKEN is returned to you upon successful completion of the call to CNMPOOL.

### Fixed-Length Character Strings

Most HLL service routines require you to pass one or more fixed-length character strings as arguments. Figure 11 on page 54 highlights fixed-length variables in the PL/I environment.
Coding PL/I Programs

1. %INCLUDE DSIPLI; /* Include the HLL macros */
   DCL HLBPTR PTR; /* HLB pointer MUST BE DECLARED! */
   DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
   DCL ORIGBLCK CHAR(40);
   DCL SPNAME CHAR(8) VARYING INIT('POOLNAME'); /* Subpool name */
   DCL SPFUNC CHAR(8); /* Subpool function */
   DCL SPTOKEN FIXED BIN(31,0); /* Subpool token (returned) */
   DCL SPLENG FIXED BIN(31,0); /* Cell size */
   DCL SPPRICNT FIXED BIN(31,0); /* Number of cells in primary */
   DCL SPSECCNT FIXED BIN(31,0); /* Number of cells in secondary */
   DCL SPCLASS FIXED BIN(31,0); /* Class of storage */

2. SPFUNC = 'ALLOC'; /* Function is ALLOCATE */
   SPTOKEN = 0; /* Initialize subpool token */
   SPLENG = 256; /* Cell size = 256 bytes */
   SPPRICNT = 3; /* Primary count = 3 */
   SPSECCNT = 2; /* Secondary count = 2 */
   SPCLASS = 1; /* Class = 31 bit addressable */

3. CALL CNMPOOL(HLBPTR,SPFUNC,SPTOKEN,SPNAME,SPLENG,SPPRICNT,SPSECCNT,SPCLASS); /* Allocate subpool */

Figure 11. Using Fixed-Length Variables in PL/I

PL/I constants for most fixed-length character strings are in the
DSIPCONS include file. DSIPCONS is optional and can be tailored to your
specific needs. See "DSIPCONS Control Block" on page 302 for more
information about DSIPCONS.

HLL service routines use two types of origin blocks: ORIGBLCK and an
origin block defined by you. The first type of origin block, ORIGBLCK, is a
40-byte structure that you must declare. This is a required initial parameter
described in "Initial Parameters" on page 49. You should never alter this
structure.

You define the second type of origin block (adorigin, gdorigin). The adorigin
and gdorigin must be at least 38 bytes long and must map to the first 38
bytes of the origin block structure (DSIPORIG). You must declare these
origin blocks separately from ORIGBLCK.

SPFUNC is declared as an 8-byte character field (CHAR(8)).

Character string 'ALLOC' is assigned to SPFUNC to be used in the call to
CNMPOOL.

SPFUNC is passed to CNMPOOL. SPFUNC could have been initialized or
passed to CNMPOOL as a character constant as shown below. In all cases,
PL/I automatically pads fixed-length character fields with blanks.

CALL CNMPOOL(HLBPTR,'ALLOC',SPTOKEN,SPNAME,SPLENG,SPPRICNT,SPSECCNT,SPCLASS); /* Allocate subpool */

Varying Length Character Strings

Several HLL service routines require you to pass a varying length character string
as an argument. Figure 12 on page 55 highlights the use of fixed-length variables in
the PL/I environment.
Control Blocks and Include Files

A number of control blocks and include files are required to run a PL/I program in the NetView environment. The DSIPLI include file, which contains INCLUDE statements for other PL/I control blocks, is the main file necessary to compile NetView programs written in PL/I. Optional include files are provided to assist you in coding and maintaining HLL programs. You can tailor DSIPLI, DSIPCNM, and DSIPCONS to fit your needs.

Note: Tailoring files can lead to better system performance in many cases. This is especially helpful in performance-sensitive environments such as installation exits.

Appendix A. PL/I Control Blocks and Include Files on page 301 contains these include files:

DSIPLI
(Required) Must be included by all HLL programs written in PL/I. DSIPLI contains INCLUDE statements for all of the external HLL control blocks and include files needed to compile and run PL/I programs in the NetView environment. See the PL/I coding template in Chapter 7. PL/I High-Level Language Services on page 63 for usage.

DSIPCONS
(Optional) Declares constants that are helpful when coding HLL programs in PL/I.

DSIPHLB
(Required) Specifies PL/I mapping of the NetView HLB control block.
DSIPORIG
(Required) Specifies PL/I mapping of the origin block of the request that caused the execution of the program currently running.

DSIPHLLS
(Required) Specifies PL/I definitions for HLL service routines.

DSIPCNM
(Optional) Declares HLL return code constants for PL/I.

DSIPPRM
(Required) Specifies PL/I mapping of the NetView Bridge parameter control block.

PL/I Input and Output Considerations

PL/I provides several input and output statements that allow you to transmit data between main storage and auxiliary storage of a computer. PL/I programs using such file I/O capabilities can run in the NetView environment. However, there are some important things to consider when doing file I/O in PL/I.

Note: See "PL/I Input and Output Considerations in a Preinitialized Environment" on page 57 for I/O considerations in a preinitialized environment.

Each file referenced from your PL/I program correlates to a physical data set in auxiliary storage. Before opening a file for I/O, you must ensure that the appropriate data set has been allocated. You can perform allocation under the time-sharing option (TSO) or by using the NetView ALLOCATE command described in the NetView online help. NetView also provides a FREE command to deallocate a data set.

If the data set is allocated from TSO, you must also add a corresponding data definition (DD) statement to the NetView startup procedure. The data definition name (ddname) must match the name of the PL/I file. The DD statement specifies a physical data set name (dsname) and gives its characteristics, as follows:

```
//OUTFILE DD DSN=MYPROG.OUTFILE, ...
```

A DD statement is not necessary if the data set is allocated using the NetView ALLOCATE command.

The next example illustrates the use of file I/O in an HLL program written in PL/I. Note the use of the ON UNDEFINEDFILE statement to protect against an OPEN failure. Check for this condition before opening a file for I/O.

```
DCL OUTFILE FILE STREAM; /* Declare output file */

/****************************Nguồn: See "PL/I Input and Output Considerations in a Preinitialized Environment" on page 57 for I/O considerations in a preinitialized environment. /

OPEN FILE(OUTFILE) OUTPUT; /* Open file for output */
```
If you want to write to a common output file from two or more PL/I programs, the programs must coordinate access to the common file. You can accomplish this using the HLL service routine CNMLK. If access is not coordinated, you might experience a system ABEND 213.

**Note:** PL/I and C cannot share an open file. However, a C program can read a file created by PL/I.

If you choose to code a GET or PUT statement without the FILE option, the compiler inserts the file names SYSIN and SYSPRINT. By default, SYSIN and SYSPRINT are directed to the terminal. These defaults are not valid, and cause undetermined results if used in the NetView environment. Terminal I/O can be done using WAIT FOR OPINPUT and CNMSMSG as described in "Chapter 11 Service Reference" on page 209. For more information about files and PL/I I/O, refer to the AD/Cycle PL/I library.

**PL/I Input and Output Considerations in a Preinitialized Environment**

Programs that you want to run in a preinitialized environment cannot perform file I/O with PL/I language specific functions. These programs can only perform stream-oriented output to SYSPRINT and SYSPRINT must be declared as an external file.

I/O services provided by the HLL service routines are not affected by preinitialized environments.

**PL/I Run-Time Considerations**

All errors detected at run time are associated with PL/I conditions that can be handled by ON-units written by the programmer. An ON-unit is a user-written statement that establishes an action to be invoked when a particular PL/I error condition is raised. PL/I error conditions can be detected by the operating system or by PL/I.

Because PL/I programs running in a production NetView environment must run with the NOSTAE and NOSPIE options for PL/I V2R3M0 or the TRAP(OFF) option for AD/Cycle PL/I, you cannot code ON-units for operating system detected conditions.

While debugging a PL/I program in a test NetView environment you can use the STAE and SPIE options for PL/I V2R3M0 or the TRAP(ON) option for AD/Cycle PL/I until run-time problems are resolved. If you plan to use programs in a preinitialized environment, you may want to run them non-preinitialized with STAE and SPIE or TRAP(ON) until run-time problems have been resolved. Most run-time errors are represented by diagnostic messages written to the SYSPRINT file. Refer to the OS PL/I Version 2 library or the AD/Cycle PL/I library for more information.
Considerations for HLL Command Processors

You must code a CMDMDL statement in DSICMD for each HLL command processor that you write. The CMDMDL TYPE depends on the functions that your command processor performs. Keep in mind that some of the HLL services are useful only when executed under a DST. The CMDMDL statement is described in the "NetView Definition Statement Reference" in the Tivoli NetView for z/OS Administration Reference.

There is no support for HLL command processors running as immediate commands (TYPE=I) or being pushed (with the macro DSIPUSH) as ABEND, LOGOFF, or RESUME routines.

Return Codes

When an HLL service routine is done, its completion code is stored in the return code field (HLBRC) of the HLB control block. Check this field after each HLL service routine invocation. Also, use this field when passing return codes between HLL programs.

For a complete list of HLL API return codes, see DSIPCNM in "Appendix A. PL/I Control Blocks and Include Files" on page 301. "Chapter 11. Service Reference" on page 209 describes the return codes that apply to each HLL service routine.

Do not use PLIRETV and PLIRETC when passing return codes between HLL programs written in PL/I. Both of these routines could yield unpredictable results in the NetView environment. To end a PL/I program normally, assign a value to HLBRC and issue a RETURN statement as illustrated below:

```
HLBRC = CNM_GOOD; /* Successful completion */
RETURN; /* Return to caller */
```

Restrictions for HLL Programs Written in PL/I

Do not use these commands when coding PL/I programs to run in the NetView environment:

Table 17. Restrictions for HLL Programs Written in PL/I

<table>
<thead>
<tr>
<th>Command</th>
<th>Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>Use NetView’s CNMSMSG service routine.</td>
</tr>
<tr>
<td>WAIT, DELAY</td>
<td>Use NetView’s WAIT command.</td>
</tr>
<tr>
<td>ON FIXEDOVERFLOW</td>
<td>These condition codes do not work in the NetView program because they require SPIE and STAE.</td>
</tr>
<tr>
<td>ON OVERFLOW</td>
<td></td>
</tr>
<tr>
<td>ON UNDERFLOW</td>
<td></td>
</tr>
<tr>
<td>ON ZERODIVIDE</td>
<td></td>
</tr>
<tr>
<td>PLIRETV PLIRETC</td>
<td>Pass return codes through HLBRC.</td>
</tr>
</tbody>
</table>

Restrictions for PL/I Programs Running in a Preinitialized Environment

The following restrictions apply to PL/I programs that run in a preinitialized environment:

- For OS PL/I V2R3M0, these restrictions apply:
– PL/I language specific file I/O is not valid. This excludes stream-oriented output to SYSPRINT.
– CONTROLLED variables are not valid.
– FETCH/RELEASE statements are not valid.

• For AD/Cycle PL/I, these restrictions apply:
  – STOP statements are not valid.
  – Invocations of assembler routines that contain an SVC LINK are not valid.

The previous restrictions are needed so that different programs can reuse the same environment. For more information about restrictions resulting from shared use of preinitialized environments, refer to the Language Environment for MVS & VM library.
Chapter 7. PL/I High-Level Language Services

This chapter describes and provides examples of NetView commands and services that support PL/I. See "Chapter 11. Service Reference" on page 203 for more information.

Note: When you are compiling PL/I programs for the NetView program, you will receive message IEL0548I. You can ignore this message.

PL/I Sample Template

Figure 13 on page 62 is a coding template sample you can use when coding HLL programs in PL/I. Use this template, with your enhancements, to utilize NetView functions and commands. Use the examples in this chapter with this template. Fully functional samples of NetView command procedures written in PL/I are in "Appendix B. PL/I Samples" on page 321."
PL/I High-Level Language Services

*PROCESS SYSTEM(MVS);
PTMPPLT: PROC(HLBPTR,CMBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/*******************************************************************************/
/* 5697-B82 (C) COPYRIGHT IBM CORPORATION 1989, 1995 */
/* ALL RIGHTS RESERVED. */
/* IEBCOPY SELECT MEMBER=((CNMS4200,PTMPPLT,R)) */
/* (Explanations included in parentheses should be deleted) */
/* (after the pertinent information has been filled in. ) */
/* Descriptive Name: High-Level Language PL/I Template */
/* (This is the more descriptive name or title of the module.) */
/* Function: */
/* Template for writing HLL modules in PL/I. */
/* (This is the description of what the module does.) */
/* (It may be paragraph or pseudocode form. ) */
/* Dependencies: */
/* (List conditions that must be met in order for this) */
/* (module to perform. An example of this might be a ) */
/* (key data area that must already have been built. ) */
/* Restrictions: */
/* (List any limitations this module may have.) */
/* Language: PL/I */
/* Input: */
/* 1) A pointer to a 4-byte field containing the address of */
/* the HLB control block. */
/* 2) A varying length character string containing the */
/* command or message which invoked this program. */
/* If this program was invoked as a command processor, */
/* this will be a command string. */
/* If this program was invoked as an installation exit */
/* (other than DSIEX02A), this will be a message string. */
/* When driven as DSIEX02A- */
/* this string will be empty and the message must */
/* be retrieved from the Initial Data Queue (IDATAQ). */
/* 3) A 40-byte structure which describes the origin of the */
/* request that caused execution of this program. */

Figure 13. PL/I Sample Template (Part 1 of 2)
Data Queue Management

The NetView program uses several data and message queues to work in conjunction with HLL service routines. Use the CNMGETD function to manipulate information retrieved from these queues to enhance your network manageability. Table 18 on page 64 describes how queues are defined for data and message.
### Table 18. Data and Message Management Queues

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAPQ</td>
<td>1</td>
<td>Message queue. Contains trapped messages. See <strong>TRAP Command</strong> on page 218.</td>
</tr>
<tr>
<td>OPERQ</td>
<td>2</td>
<td>Operator input queue. See <strong>GO Command</strong> on page 213 and <strong>QUEUE Command</strong> on page 214.</td>
</tr>
<tr>
<td>DATAQ</td>
<td>3</td>
<td>Data queue. Contains data sent from another HLL command processor or installation exit routine. See <strong>CNMSMSG (CNMSENDMSG): Send Message or Command</strong> on page 282.</td>
</tr>
<tr>
<td>IDATAQ</td>
<td>4</td>
<td>Initial data queue. Contains the full message or MSU that invokes the HLL command processor through NetView automation. It also contains messages that drive DSIEX02A and DSIEX17. This is also the queue where an application command processor receives an MDS-MU from the NetView high performance transport, the MS transport, or operations management for an unsolicited request or asynchronous reply. For invocations from pipes, the IDATAQ contains the data flowing into the HLL command processor.</td>
</tr>
<tr>
<td>CNMIQ</td>
<td>5</td>
<td>CNMI solicited data queue. Contains RUs solicited through the CNMI service routine. Chained RUs are treated like multiline messages. See <strong>CNMCNMI (CNMI): CNMI Access Under a DST</strong> on page 224.</td>
</tr>
<tr>
<td>MDSMUQ</td>
<td>6</td>
<td>MDS-MU data queue. Contains message units (MUs) received as synchronous replies. The MDS-MUs are received from operations management, the MS transport, and the high performance transport. Refer to the <em>Tivoli NetView for z/OS Application Programmer’s Guide</em> for more information.</td>
</tr>
</tbody>
</table>

The examples in the following sections show how the preceding queues are used with HLL command procedures.

### Sending Messages

Figure 14 on page 65 is an example of sending messages:
Parsing Input Strings Similar to NetView Command List Language

Figure 15 on page 66 is an example of parsing the input string that is similar to parsing with the NetView command list language. The example parses the first ten tokens individually, just as the NetView command list language would parse them into &1, &2, and so on. This example also sets an equivalent variable to &PARMSTR.

```pli
 PL/I High-Level Language Services

/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'Line 1 of 3 ','MSG_C','OPER','');
CALL CNMSMSG(HLBPTR,'Line 2 of 3 ','MSG_D','OPER','');
CALL CNMSMSG(HLBPTR,'Line 3 of 3 ','MSG_F','OPER','');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'Line 1 of 3 ','MSG_C','TASK','OPER2');
CALL CNMSMSG(HLBPTR,'Line 2 of 3 ','MSG_D','TASK','OPER2');
CALL CNMSMSG(HLBPTR,'Line 3 of 3 ','MSG_F','TASK','OPER2');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'Line 1 of 3 ','MSG_C','TASK','OPER2');
CALL CNMSMSG(HLBPTR,'Line 2 of 3 ','MSG_D','TASK','OPER2');
CALL CNMSMSG(HLBPTR,'Line 3 of 3 ','MSG_F','TASK','OPER2');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'Line 1 of 3 ','MSG_C','TASK','OPER2');
CALL CNMSMSG(HLBPTR,'Line 2 of 3 ','MSG_D','TASK','OPER2');
CALL CNMSMSG(HLBPTR,'Line 3 of 3 ','MSG_F','TASK','OPER2');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'Hello Sysop ','MSG','SYSOP','');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'Hello Authrcvr ','MSG','AUTHRCV','');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'This should only be in log','MSG','NETVLOG','');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'test msg','MSG','SEQLOG','SQLOGTSK');
/*****************************/
/*                  */
/*****************************/
CALL CNMSMSG(HLBPTR,'hello group','MSG','OPCLASS','+GROUP1');

Figure 14. Sending Messages (PL/I)

Parsing Input Strings Similar to NetView Command List Language

Figure 15 on page 66 is an example of parsing the input string that is similar to parsing with the NetView command list language. The example parses the first ten tokens individually, just as the NetView command list language would parse them into &1, &2, and so on. This example also sets an equivalent variable to &PARMSTR.
Parsing Input String Similar to REXX

Figure 16 on page 67 parses an input string in a manner similar to that used in REXX. REXX parses the first four tokens individually and puts the rest of the input into the fifth token:

```arg token1 token2 token3 token4 token5```

PL/I parsing is similar.
Figure 16. Parsing an Input String Similar to REXX (PL/I)

Parsing Input String in CNMSCAN

Figure 17 on page 68 shows how the C SSCANF function, CNMSCAN, parses and converts in a single step and accepts any input character as a delimiter.
PL/I High-Level Language Services

/***************************************************************/
/* */
/* Other Declarations */
/* */
/***************************************************************/
DCL CNT FIXED BINARY(31,0); /* Number of strings parsed */
DCL INPUT_STR CHAR(256) VARYING; /* Sscan input string */
DCL FORMAT CHAR(256) VARYING; /* Sscan format string */
DCL MSGBUF CHAR(256) VARYING; /* Message buffer */
DCL (CH1,CH2,CH3,CH4,CH5,CH6) CHAR(8) VARYING; /* Char vars */
DCL (FX1,FX2,FX3) FIXED BINARY (31,0); /* Fixed vars */
/***************************************************************/
/* */
/* Execution */
/* */
/***************************************************************/
/* Parse out a string */
/***************************************************************/

Figure 17. Parsing an Input String in CNMSCAN (PL/I) (Part 1 of 3)
INPUT_STR='parm1 '|| /* Set input string up */
'parm2 '|| 'parm3 '|| '10000 '|| '200 '|| 'FFFFF '|| '01XYZ2 '|| ' parm4';

FORMAT= /* The format string says to: */
'(%S)' || /* (1) Find a character string */
'(%4S)' || /* (2) Find a 4-byte character string */
'(%S)' || /* (3) Find a character string */
'(%*S)' || /* (4) Skip over a character string */
'(%D)' || /* (5) Find a decimal string */
'(%2D)' || /* (6) Find a 2-byte decimal string */
'(%*S)' || /* (7) Skip over a character string */
'(%X)' || /* (8) Find a hex string */
'(%{ Q10})'|| /* (9) Find a string that contains one of the bracketed characters, stop scanning when a non-bracketed character is found */
'(%{2ZYX})'|| /* (10) Find a string that contains one of the bracketed characters, stop scanning when a non-bracketed character is found */
'(%{¬4})'; /* (11) Find a string that does NOT contain a 4, stop scanning when a 4 is found */

CALL CNMSCAN(HLBPTR, /* Scan the input string... */
INPUT_STR, /* ...input is in here */
FORMAT, /* ...format string */
CNT, /* ...number of string parsed */
CH1, /* ...character string */
CH2, /* ...character string */
CH3, /* ...character string */
FX1, /* ...decimal string */
FX2, /* ...decimal string */
FX3, /* ...hex string */
CH4, /* ...character string */
CH5, /* ...character string */
CH6, /* ...character string */
' '); /* ...not used */

Figure 17. Parsing an Input String in CNMSCAN (PL/I) (Part 2 of 3)
Invoking Synchronous Commands

Figure 18 is an example of an HLL command processor invoking another command. The command could be another HLL command, a command list, a VTAM command, or a NetView command.

```pli
/* After executing, the variables have these values: */
/* */
/* CH1 = "parm1" */
/* CH2 = "parm" */
/* CH3 = "2" */
/* FX1 = 10000 Decimal, 2710 Hex */
/* FX2 = 20 Decimal, 14 Hex */
/* FX3 = 1048575 Decimal, FFFFF Hex */
/* CH4 = "01" */
/* CH5 = "XYZ2" */
/* CH6 = "parm" */
/* */
/* */
/********************************************************************/
```

Figure 17. Parsing an Input String in CNMSCAN (PL/I) (Part 3 of 3)

Sending Commands

Figure 19 on page 71 is an example of sending a command to execute under another task. The command to be run under the other task could be another HLL command, a command list, a VTAM command, or a NetView command.

Use this process to execute commands under DSTs, other OSTs, or the PPT.
Waiting and Trapping

**Figure 20 on page 72** uses the TRAP and WAIT commands to issue a command, trap the output, and respond depending on the output that is encountered. **Figure 21 on page 74** provides similar function using the PIPE command. Both examples activate the given logical unit (LU) and issue an appropriate message.

**Example of Using the TRAP and WAIT Commands (PL/I)**

The syntax for the command shown in **Figure 20 on page 72** follows:

```
PACTLU
```

**Where:**

- `luname`
  
  Is the name of the LU to be activated.

---

**Figure 19. Sending a Command to Execute Another Task (PL/I)**
PL/I High-Level Language Services

********************************************************************************
/*
/* Other Declarations
/*
********************************************************************************
DCL GETBLOCK CHAR(40); /* Area for the Orig Block */
DCL GETPTR PTR; /* Pointer to the Orig Block */
DCL INBUF CHAR(256) VAR; /* Buffer area for messages */
DCL NODENAME CHAR(8) VAR; /* Nodename to be activated */
DCL STATUS CHAR(8) VAR; /* Status of the resource */
DCL CNT FIXED BIN(31,0); /* Number of elements parsed */
********************************************************************************
/*
/* Execution
/*
********************************************************************************
GETPTR=ADDR(GETBLOCK); /* Address the Orig Block */

CALL CNMSCAN(HLBPTR,/* Parse the input ...
      CMDBUF,/* ...command line is the input */
      '%*S%8S',/* ...skip over command name */
      CNT,/* ...returned */
      NODENAME);/* ...nodename */
IF CNT=1 THEN /* Nodename specified? */
  DO; /* Yes... */
    CALL CNMCMD(HLBPTR,/* Trap the following VTAM msgs */
      ' TRAP AND SUPPRESS ONLY MESSAGES IST*');
    CALL CNMCMD(HLBPTR,' V NET,ACT,ID='||NODENAME); /* Activate node*/
    CALL CNMGETD(HLBPTR,/* Get the first trapped msg... */
      'GETMSG',/* ...function is get a msg */
      INBUF,/* ...result goes here */
      256,/* ...max input length */
      GETBLOCK,/* ...must provide a work area */
      TRAPQ,/* ...message is trapped */
      1);/* ...get the first one */

Figure 20. Waiting and Trapping Using the TRAP and WAIT Commands (PL/I) (Part 1 of 2)
Example of Using the PIPE Command (PL/I)

The syntax for the command shown in Figure 21 on page 74 is as follows:

**PACTPIP**

```
PACTPIP luname
```

Where:

- **luname**
  
  Is the name of the LU to be activated.
PL/I High-Level Language Services

```pli
/** Other Declarations */
DCL GETBLOCK CHAR(40);  /* Area for the Orig Block */
DCL GETPTR PTR;        /* Pointer to the Orig Block */
DCL INBUF CHAR(256) VAR; /* Buffer area for messages */
DCL NODENAME CHAR(8) VAR; /* Nodename to be activated */
DCL STATUS CHAR(8) VAR; /* Status of the resource */
DCL CNT FIXED BIN(31,0); /* Number of elements parsed */
DCL VARNM CHAR(16) VAR /* Varname for PIPE result */
   INIT('VNETRESULT'); /* should be VNETRESULT */

/** Execution */
GETPTR=ADDR(GETBLOCK);  /* Address the Orig Block */

/** Scan the input for the nodename to activate */
CALL CNMSCAN(HLBPTR, /* Parse the input ... */
   CMDBUF, /* ...command line is the input */
   '%%S%8S', /* ...skip over command name */
   CNT, /* ...returned */
   NODENAME); /* ...nodename */
IF CNT=1 THEN /* Nodename specified? */
   DO; /* Yes... */
      CALL CNMVARS(HLBPTR, /* Declare a variable ... */
         DCL, /* ...declare function */
         ', /* ...not needed */
         0, /* ...not needed */
         VARNM, /* ...specifies variable name */
         LOCAL); /* ...local variable pool */
```

Figure 21. Waiting and Trapping Using the PIPE Command (PL/I) (Part 1 of 2)
For more information about the PIPE command, refer to Tivoli NetView for z/OS Customization: Using Pipes.

Retrieving Information

Figure 22 on page 76 is an example of how an HLL command processor or installation exit routine can retrieve information from the NetView program.

For more information, see "CNMINFC (CNMINFOC): Query NetView Character Information" on page 253 and "CNMINFI (CNMINFOI): Query NetView Integer Information" on page 256 for a list of the values supported.
PL/I High-Level Language Services

/* Other Declarations */
DCL CDATA CHAR(18) VAR; /* Character information holder */
DCL IDATA FIXED BIN(31,0); /* Integer information holder */

CALL CNMINFC(HLBPTR, /* Retrieve the date & time... */
             'DATETIME', /* ...specify the variable */
             CDATA, /* ...result goes here */
             18); /* ...at most 18 bytes */
CALL CNMINFI(HLBPTR, /* Retrieve the number of colors */
             'COLORS', /* ...specify the variable */
             IDATA); /* ...result goes here */

Figure 22. Retrieving Information (PL/I)

Command List Variable Access

Figure 23 on page 77 shows the capability of updating common global variables. This example increments a global variable named GVARIABLE by 1. Task global variables are updated and read the same way. The only difference is the pool name TGLOBAL is specified instead of CGLOBAL.

Figure 24 on page 78 shows how to retrieve values from a stem variable after issuing a PIPE command with the STEM stage command.
Accessing Common Global Variables (PL/I)

```pli
DCL DATA_IN CHAR(24) VAR; /* Holds the input data */
DCL DATA_IN_LEN FIXED BIN(31,0) INIT(24); /* Max length of input*/

CALL CNMVARS(HLBPTR, /* Read the global variable... */
  'GET', /* ...function is read */
  DATA_IN, /* ...result goes here */
  DATA_IN_LEN, /* ...truncate after 24-bytes */
  'GVARIABLE', /* ...variable name is GVARIABLE */
  'CGLOBAL'); /* ...variable pool is CGLOBAL */

DATA_IN=DATA_IN+1; /* Increment Variable */

CALL CNMVARS(HLBPTR, /* Update the global variable... */
  'PUT', /* ...function is write */
  DATA_IN, /* ...data is here */
  '', /* ...not used */
  'GVARIABLE', /* ...variable name is GVARIABLE */
  'CGLOBAL'); /* ...variable pool is CGLOBAL */
```

Figure 23. Accessing Common Global Variables (PL/I)
Accessing Stem Variables Using the PIPE Command (PL/I)

Other Declarations

DCL NUMVARS CHAR(4) VARYING; /* Number of variables in stem */
DCL PIPECMD CHAR(256) VARYING; /* Buffer to hold PIPE command */
DCL STEMLINE CHAR(80) VARYING; /* Store value of stem variable */
DCL VARNAME CHAR(15) VARYING; /* Name of stem variable */

Execution

Build a PIPE command.

PIPECMD = 'PIPE < MYINFILE'||
  ' | NLOCATE 1.1 &*& 1.2 &/*&'||
  ' | STEM PIPELINE'; /* Build PIPE command */

CNMCOMMAND DATA(PIPECMD); /* Issue PIPE command */

Local variable PIPELINE0 contains the character representation of the total number of PIPELINEx stem variables.

VARNAME = 'PIPELINE0';

Get number of stem variables and store value in NUMVARS.

CNMVARPOOL FUNC('GET')
  NAME(VARNAME)
  POOL('LOCAL')
  DATA(NUMVARS)
  LENG(4);

Local variable PIPELINE1 contains the first non-comment line in MYINFILE.

VARNAME = 'PIPELINE1';

Get first non-comment line and store value in STEMLINE.

CNMVARPOOL FUNC('GET')
  NAME(VARNAME)
  POOL('LOCAL')
  DATA(STEMLINE)
  LENG(80);

Figure 24. Accessing Stem Variables Using the PIPE Command (PL/I)

For more information about the PIPE command, refer to Tivoli NetView for z/OS Customization: Using Pipes.

Customization: Using PL/I and C
Figure 23 on page 77 shows how you can update common global variables. However, the example does not provide for protecting the updating of the variable named GVARIABLE by using a lock. The need for protecting the updating should be assessed on a case-by-case basis.

Figure 25 shows how to modify Figure 23 on page 77 to obtain a lock before attempting the update. The lock name can be the same as the global variable, or it can be different.

If you decide that it is important to synchronize the updating of a variable, you can use the following lock method, or you can run all the updates on a given task. Because only one process can occur on a task at a time, the updates are serialized. This could be any task, including the PPT.

DCL DATA_IN CHAR(24) VAR; /* Holds the input data */
DCL DATA_IN_LEN FIXED BIN(31,0) INIT(24); /* Max length of input */
/***************************************************************************/
/* Obtain the lock to secure the accuracy of the update */
/***************************************************************************/
CALL CNMLK(HLBPTR, /* Obtain the Lock ... */
  'LOCK', /* ...function is obtain lock */
  'GVARIABLE', /* ...name of the lock */
  '', /* ...not used */
  'WAIT'); /* ...wait if not available */
/***************************************************************************/
/* Find out the value of the variable */
/***************************************************************************/
CALL CNMVARS(HLBPTR, /* Read the global variable... */
  'GET', /* ...function is read */
  DATA_IN, /* ...result goes here */
  DATA_IN_LEN, /* ...truncate after 24-bytes */
  'GVARIABLE', /* ...variable name is GVARIABLE */
  'CGLOBAL'); /* ...variable pool is CGLOBAL */

DATA_IN=DATA_IN+1; /* Increment Variable */
/***************************************************************************/
/* Set the global variable */
/***************************************************************************/
CALL CNMVARS(HLBPTR, /* Update the global variable... */
  'PUT', /* ...function is write */
  DATA_IN, /* ...data is here */
  '', /* ...not used */
  'GVARIABLE', /* ...variable name is GVARIABLE */
  'CGLOBAL'); /* ...variable pool is CGLOBAL */
/***************************************************************************/
/* Release the lock to let other tasks update GVARIABLE */
/***************************************************************************/
CALL CNMLK(HLBPTR, /* Free the Lock ... */
  'UNLOCK', /* ...function is free lock */
  'GVARIABLE', /* ...name of the lock */
  '', /* ...not used */
  ''); /* ...not used */
Figure 26 shows how to code an HLL command processor to accept operator input in single-line mode. The interface is similar to the &PAUSE function of the NetView command list language. Input is requested by the application using the WAIT FOR OPINPUT command, and retrieved by the application using the CNMGETD service routine. The operator can respond by using the GO command. See “GO Command” on page 213 for more information.

```pli
/**************************************************************************/
/* */
/* Other Declarations */
/* */
/**************************************************************************/
DCL GETBLOCK CHAR(40); /* Area for the Orig Block */
DCL GETPTR PTR; /* Pointer to the Orig Block */
DCL DATA_INCHAR(256) VAR; /* Buffer area for messages */
/**************************************************************************/
/**/
/* Execution */
/**/
/**************************************************************************/
GETPTR=ADDR(GETBLOCK); /* Address the Orig Block */
CALL CNMSMSG(HLBPTR, /* Send a message... */
    'ENTER OPERATOR INPUT DATA', /* ...text of message */
    'MSG', /* ...single line message */
    'OPER', /* ...to the invoking operator */
    ''); /* ...not used */
CALL CNMCMD(HLBPTR, ' WAIT 10 SECONDS FOR OPINPUT'); /* Wait... */
IF HLBRC=CNM_OPINPUT_ON_WAIT THEN
    DO; /* Operator input supplied... */
        CALL CNMGETD(HLBPTR, /* Get the first trapped msg... */
            'GETMSG', /* ...function is get a msg */
            DATA_IN, /* ...result goes here */
            256, /* ...max input length */
            GETBLOCK, /* ...must provide a work area */
            OPERQ, /* ...message is on OPINPUT QUEUE*/
            1); /* ...get the first one */
        CALL CNMSMSG(HLBPTR, /* Send a message... */
            'OPERATOR INPUT IS:', /* ...text of message */
            'MSG', /* ...single line message */
            'OPER', /* ...to the invoking operator */
            ''); /* ...not used */
    END;
ELSE /* No operator input supplied */
    CALL CNMSMSG(HLBPTR, /* Send a message... */
        'NO OPERATOR INPUT SUPPLIED', /* ...text of message */
        'MSG', /* ...single line message */
        'OPER', /* ...to the invoking operator */
        ''); /* ...not used */
ENDIF;
/**************************************************************************/
/* */
/* Other Declarations */
/* */
/**************************************************************************/
```

Figure 26. Operator Input (PL/I)

**VIEW Command Processor**

Figure 27 on page 81 uses the full-screen VIEW command processor to create and initialize a local variable called PARM1. The VIEW command processor is invoked, displaying a full-screen panel. For more information, refer to the Tivoli NetView for z/OS Customization Guide.
Figure 27 shows the panel definition that is invoked by Figure 26.

Figure 28 shows the panel definition that is invoked by Figure 27.

```
/+TESTHLL
$ %TEST THE VIEW COMMAND WITH HLL
$ +==================================================================
$ + Example of using the VIEW command with HLL:
$ +
$ + Notes: The field below, PARM1, is defined as a variable
$ + by preceding the string PARM1 with an ampersand (&).
$ +
$ + INPUT======> &PARM1
$ +
$ +
%Action===>˜&CUR
$ +
$ +

Figure 28. Sample Panel Definition
```
Message Processing

Figure 29 lists the message attributes of a message. The invocation must be a result of an entry in the NetView automation table. This example applies to both single-line and multiline messages. Refer to Tivoli NetView for z/OS Automation Guide for more information.

![Code example](image-url)

Command Authorization Checking

This section describes an example of the command authorization checking capabilities provided by the NetView program. You define the operator, the operator’s authority, and the restrictions on invoking the command, keyword, and value.
The command gives the return code that the command authorization checking service routine returned to the operator.

The syntax for which this command checks is as follows:

```
PSPCCKO
```

Where:

- **PARM**\textsubscript{x}:
  - Specifies the keyword variable used to perform the authorization check.

- **VAL**\textsubscript{x}:
  - Indicates the value variable used to perform the authorization check.

Before using the example in Figure 30 on page 85, perform the following NetView set up as it applies to your method of command authorization checking.

**Command Authorization Checking Using CMDAUTH=TABLE**

If you are using CMDAUTH=TABLE with TBLNAME=CMDTABLE in NetView domain CNM01 in network NETA, perform the following set up prior to using the sample code.

**In DSIPARM(CMDTABLE)**

Restrict access to the command, keywords, and values as well as the operators and classes that can access them.

The command PSPCCKO can be executed by operators in GRP1 and GRP2. Operators in GRP1 can issue any keyword or keyword value, but operators in GRP2 cannot use the value of VAL1 with keyword PARM2 and cannot issue PARM3.

```
PROTECT NETA.CNM01.PSPCCKO
PROTECT NETA.CNM01.PSPCCKO.PARM2
PROTECT NETA.CNM01.PSPCCKO.PARM2.VAL1
PROTECT NETA.CNM01.PSPCCKO.PARM3
PROTECT NETA.CNM01.PSPCCKO.PARM3.VAL1
GROUP GRP1 OPER1,OPER2,FRED,BILL,JENNY
GROUP GRP2 OPER3,OPER4,FRED,LISA,BETH
GROUP GRP3 JOE
PERMIT GRP1 NETA.CNM01.PSPCCKO
PERMIT GRP1 NETA.CNM01.PSPCCKO.PARM2
PERMIT GRP1 NETA.CNM01.PSPCCKO.PARM2.VAL1
PERMIT GRP1 NETA.CNM01.PSPCCKO.PARM3
PERMIT GRP1 NETA.CNM01.PSPCCKO.PARM3.VAL1
PERMIT GRP2 NETA.CNM01.PSPCCKO
PERMIT GRP2 NETA.CNM01.PSPCCKO.PARM2
```

**Command Authorization Checking Using CMDAUTH=SAF**

If you are using CMDAUTH=SAF and OPERSEC=SAFDEF in NetView domain CNM01 in network NETA, perform the following set up prior to using the sample code. RACF is used as the SAF product in this example.

**In RACF**

Restrict access to the command, keywords, and values as well as the operators and classes that can access them.
The command PSPCCKO can be executed by operators in GRP1 and GRP2. Operators in GRP1 can issue any keyword or keyword value, but operators in GRP2 cannot use the value of VAL1 with keyword PARM2 and cannot issue PARM3.

RDEFINE NETCMDs NETA.CNM01.PSPCCKO UACC(NONE)
RDEFINE NETCMDs NETA.CNM01.PSPCCKO.PARM2 UACC(NONE)
RDEFINE NETCMDs NETA.CNM01.PSPCCKO.PARM2.VAL1 UACC(NONE)
RDEFINE NETCMDs NETA.CNM01.PSPCCKO.PARM3 UACC(NONE)
RDEFINE NETCMDs NETA.CNM01.PSPCCKO.PARM3.VAL1 UACC(NONE)

ADDGROUP GRP1
ADDGROUP GRP2
ADDGROUP GRP3
CONNECT OPER1 GROUP(GRP1)
CONNECT OPER2 GROUP(GRP1)
CONNECT FRANK GROUP(GRP1)
CONNECT BILL GROUP(GRP1)
CONNECT JENNY GROUP(GRP1)
CONNECT OPER3 GROUP(GRP2)
CONNECT OPER4 GROUP(GRP2)
CONNECT FRED GROUP(GRP2)
CONNECT LISA GROUP(GRP2)
CONNECT BETH GROUP(GRP2)
CONNECT JOE GROUP(GRP3)

PERMIT NETA.CNM01.PSPCCKO CLASS(NETCMDs) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.PSPCCKO.PARM2 CLASS(NETCMDs) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.PSPCCKO.PARM2.VAL1 CLASS(NETCMDs) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.PSPCCKO.PARM3 CLASS(NETCMDs) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.PSPCCKO.PARM3.VAL1 CLASS(NETCMDs) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.PSPCCKO.PARM3.VAL1 CLASS(NETCMDs) ID(GRP2) ACCESS(READ)
PERMIT NETA.CNM01.PSPCCKO.PARM2 CLASS(NETCMDs) ID(GRP2) ACCESS(READ)
Command Authorization Checking Sample Code

```pli
/* --------- Other Declarations --------- */
DCL INBUF CHAR(80) VAR; /* Buffer area for messages */
DCL CMDNAMEV CHAR(8) VAR; /* Command that invoked us */
DCL KEYWORDV CHAR(8) VAR; /* Keyword of invocation */
DCL KEYVALUEV CHAR(8) VAR; /* KeyValue of invocation */
DCL CMDNAME CHAR(8); /* Command that invoked us */
DCL KEYWORD CHAR(8); /* Keyword of invocation */
DCL KEYVALUE CHAR(8); /* KeyValue of invocation */
DCL CNT FIXED BIN(31,0); /* Number of elements parsed */
/* --------- Execution --------- */
CALL CNMSCAN(HLBPTR, /* Parse the input ... */
               CMDBUF, /* ...command line is the input */
               /* SYNTAX OF COMMAND IS: */
               /* CMDNAME KEYWORD(KEYVALUE) */
               CNT, /* ...number strings parsed */
               CMDNAMEV, /* ...command goes here */
               KEYWORDV, /* ...keyword goes here */
               KEYVALUEV); /* ...keyvalue goes in here */
```

Figure 30. Command Authorization Checking (PL/I) (Part 1 of 2)
Altering Data

The DSIEX02A installation exit routine changes the echoed command message (MSGTYPE=*) to be more informative by giving the time that the command was entered.

Without the exit, the output is:

WHO

With the exit, the output is:

Command entered was: "WHO" at 12:00:00

Figure 30. Command Authorization Checking (PL/I) (Part 2 of 2)
**PL/I High-Level Language Services**

DSIEX02A: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);

/*****************************/
/*
  
  
  */
/* Change Activity: */
/* date,author: description of changes */
/*****************************/
/*
  */
/* Operand Declarations */
/*
  */
DCL HLBPTR PTR; /* Pointer to the HLB */
%INCLUDE DSIPLI; /* Include the HLL macros */
DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
DCL ORIGBLCK CHAR(40); /* Area for the Orig Block */
DCL ORIGIN PTR; /* Pointer to the Orig Block */
DCL ADDR BUILTIN; /* Built-in function */
ORIGIN=ADDR(ORIGBLCK); /* Address the Orig Block */
/*****************************/
/*
  */
/* Other Declarations */
/*
  */
DCL GETBLOCK CHAR(40); /* Area for the Orig Block */
DCL DATAIN CHAR(255) VAR; /* Old command text */
DCL TIME CHAR(256) VAR; /* Area for time */
/*****************************/
/*
  */
/* Execution */
/*
  */
GETPTR=ADDR(GETBLOCK); /* Address the Orig block */
CNMINFOC /* Retrieve the time... */
ITEM('TIME') /* ...variable is time of day */
DATA(TIME) /* ...the result goes here */
LENG(256); /* ...max length of 256 */
CMGETDATA /* Peek the msg before altering */
FUNC('PEEKLINE') /* ...subfunction is PEEK */
QUEUE(IDATAQ) /* ...initial data queue */
DATA(DATAIN) /* ...result goes here */
LENG(256) /* ...max length is 256 */
ORIGIN(GETBLOCK) /* ...use new Orig block */
LINE(1); /* ...check the first line */
IF GETPTR->ORIG_MSG_TYPE ='*' THEN /* Echo'ed message? */
  CNMALTDATA /* Replace the text ... */
  FUNC('REPLINE') /* ...function is replace */
  QUEUE(IDATAQ) /* ...initial data queue */
  DATA('Command entered was: "'||DATAIN||'" at '||TIME)
  /* ...text of new message */
  ORIGIN(GETBLOCK) /* ...use Peeked Orig block */
  LINE(1); /* ...replace the first line */
  HLBC=CNM_GOOD; /* Clear RC */
END DSIEX02A;

**Figure 31. Altering Data (PL/I)**
Figure 32 shows how to display the character representation of the contents of the storage that the NetView program can access. For example, after locating the address of the main vector table using DISPMOD DSIMNTEX, you can display the first 4 bytes of the DSIMVT control block.

```pli
DCL NUM_PARMS FIXED BIN(31); /* Number of parms passed */
DCL XADDR FIXED BIN(31); /* Hex value of source_ptr */
DCL NUM_BYTES FIXED BIN(31); /* Number of bytes to display */
DCL INPUT_BFR CHAR(4096); /* Buffer where data is copied */
DCL SOURCE_PTR PTR; /* Address to copy from */
DCL I FIXED BIN(31); /* Work counter */

ibrariesextru罟

CNMSSCAN DATA(CMDBUF) /* Scan the command for: */
  FORMAT('%*S'||/* ...skip the command */
       '%X'||/* ...save the source address */
       '%X') /* ...save the length */
  COUNT(NUM_PARMS) /* ...number of parms scanned */
  P1(XADDR) /* ...the address to display */
  P2(NUM_BYTES); /* ...for this number of bytes */
SELECT;
  WHEN(NUM_PARMS=2) /* Did they give an address and */
    WHEN(NUM_BYTES<=0) /* ...a length? */
      CNMSENDMSG /* No, give error message... */
      DATA('INVALID NUMBER OF OPERANDS') /*...text */
      MSGTYPE('MSG') /* ... message */
      DESTTYPE('OPER'); /* ... to the operator */
      WHEN(NUM_BYTES=4096) /* Did they give a valid length? */
        CNMSENDMSG /* No, give error message... */
        DATA('INVALID LENGTH GIVEN, MUST BE LESS THAN '||
            'OR EQUAL TO FFF') /*...text of message */
        MSGTYPE('MSG') /* ... message */
        DESTTYPE('OPER'); /* ... to the operator */
  OTHERWISE
    DO;
      UNSPEC(SOURCE_PTR) = UNSPEC(XADDR); /* assign value into a ptr */
      CNMCOPYSTR /* Copy storage */

Figure 32. Storage Access (PL/I) (Part 1 of 2)
Data Set Access

Figure 33 contains the opening (CNMEMO), reading (CNMMEMR), and closing (CNMMEMC) of NetView partitioned data sets. This example reads the CNMSTYLE member of DSIPARM and displays the contents to the operator.

DCL MEMBER CHAR(8); /* Member name to read */
DCL DDNAME CHAR(8); /* DDNAME to read */
DCL TOKEN FIXED BIN(31,0); /* Token used to match open to ...read and close */
DCL MRDATA CHAR(80) VAR; /* Line that is read */

/***************************************************************************/
/** */
/* Execution */
/** */
/***************************************************************************/
DDNAME='DSIPARM';
MEMBER='CNMSTYLE';

/***************************************************************************/
/* OPEN THE MEMBER */
/***************************************************************************/
CALL CNMEMO(HLBPTR, /* Open the data set member ... */
TOKEN,
DDNAME,
MEMBER);

IF HLBC=CNM_GOOD THEN
CALL CNMSMSG(HLBPTR, /* OPEN failed... */
'OPEN FOR DATA SET FAILED RC='||CHAR(HLBC),
'MSG', /* ...single line message */
'OPER', /* ...to the operator */
''); /* ...taskname ignored */
ELSE
DO; /* Open was successful... */

Figure 33. Data Set Access (PL/I) (Part 1 of 2)
Communicating with Devices

The NetView program provides the CNMCNMI service routine for use in communicating with devices in the network through the CNMI. You can access any data that is returned using the CNMGETD service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

Figure 34 on page 91 uses the CNMCNMI service routine to send a request for product set ID data to a specified PU. Any data returned is sent as a message to the operator.

The syntax of the command is:

**PNMVTPU**

<table>
<thead>
<tr>
<th>PNMVTPU</th>
<th>pname</th>
<th>OWN</th>
<th>ALL</th>
</tr>
</thead>
</table>

Where:

**ALL**

Specifies that vital product data is to be retrieved for the PU and its attached ports.

**OWN**

Specifies that vital product data is to be retrieved for the PU only. This keyword is the default.
**pname**

Is the name of the PU from which vital product data is received. This parameter is required.

```pli
/* */
/* Other Declarations */
/* */
DCL RCODE FIXED BIN(31,0); /* Return code */
DCL COUNT FIXED BIN(31,0); /* Count of Scanned args */
DCL PUNAMEV CHAR(8) VAR; /* PUNAME varying length */
DCL PUNAME CHAR(8); /* PUNAME fixed length */
DCL GETBLOCK CHAR(40); /* Area for the work orig block */
DCL GETPTR PTR; /* Pointer to the work Orig Block*/
DCL DATAIN CHAR(1024) VAR; /* Buffer for the RU */
DCL OWNORALL CHAR(8) VAR; /* Own or all placeholder */

/* Vital Product Data RU definitions */
/* From the VTAM Programming Manual, a forward RU is defined below */
/* */
/* Byte Value Description */
/* 0 81 Network services, logical services */
/* 1 08 Management services */
/* 2 10 Request code */
/* 3 00 Format 0 */
/* 4 00 Ignore target names, */
/* Solicit a reply, and */
/* No CMM header contained */
/* 5 00 Reserved */
/* 6-7 000E Length of NS RU */
/* 8-15 NS RU -- NMVT -- documented in SNA Ref Sum */
/* 8-A 41038D NS Header for NMVT */
/* B-C 0000 Retired */
/* D-E 0111 PRID */
/* F 00 unsolicited NMVT, */
/* only NMVT for this PRID */
/* 10-16 One MS major vector */
/* 10-11 0006 Length field of PSID (Product Set ID) vector */
/* 12-13 8090 Code point for PSID */
/* 14-15 Length of subvector */
/* 14 02 Length of subvector */
/* 15 81 Request information on control unit only */
/* 15 83 Request information on control unit and its */
/* attached devices */
/* 16 F1 From VTAM programming, PU */
/* 17 08 Length of PU name */
/* 18 PUNAME Eight byte PUNAME, left justified */
/* 20 00 End of RU */
```

**Figure 34. CNMCNMI Service Routine (PL/I) (Part 1 of 3)**
DCL FORWARD_RU CHAR(100) VAR INIT('810810000000000E41038D00000111000006809002'X);
DCL OWN CHAR(1) VAR INIT('81'X);
DCL ALL CHAR(1) VAR INIT('83'X);
DCL PUNAME_HDR CHAR(2) VAR INIT('F108'X);
DCL ENDOFRU CHAR(1) VAR INIT('00'X);

/***************************************************************************/
/**/ /* Execution */
/**/ /* */
/***************************************************************************/
RCODE=0; /* Initialize return code */
GETPTR=ADDR(GETBLOCK); /* Address the work Orig Block */

CALL CNMSCAN(HLBPTR, /* Scan the command line... */
            CMDBUF, /* ...input in command line */
            ' %*S0S', /* ...skip over the command */
            COUNT, /* ...number of args parsed */
            PUNAMEV, /* ...pname */
            OWNORALL); /* ...own or all specified */
PUNAME=PUNAMEV; /* Get fixed length PU name */

SELECT;
WHEN(COUNT=1) /* Own or All not specified */
    FORWARD_RU=FORWARD_RU||OWN||PUNAME_HDR||PUNAME||ENDOFRU;
        /* Default is OWN */
WHEN(OWNORALL='OWN') /* Own or All not specified */
    FORWARD_RU=FORWARD_RU||OWN||PUNAME_HDR||PUNAME||ENDOFRU;
        /* Process OWN */
WHEN(OWNORALL='ALL') /* Own or All not specified */
    FORWARD_RU=FORWARD_RU||ALL||PUNAME_HDR||PUNAME||ENDOFRU;
        /* Invalid parm... tell user */
DO;
    CALL CNMSMSG(HLBPTR,'INVALID COMMAND SYNTAX', /* wrong... */
                 'MSG','TASK',ORIGIN ->ORIG_TASK); /* ...syntax */
        RCODE=8; /* Bad syntax */
END;
/* Of Select */

END; /* Of Select */

Figure 34. CNMCNMI Service Routine (PL/I) (Part 2 of 3)
Performing I/O on a VSAM File (Keyed File Access)

Figure 35 on page 94 shows coding a NetView HLL command processor that allows I/O to a VSAM file through the CNMKIO service routine.

The command processor must execute on a DST. Use either the CNMSMSG service routine (with a type of COMMAND) or the EXCMD command.
Figure 35 creates a database with five records and the following keys and data:

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>A</td>
</tr>
<tr>
<td>02</td>
<td>B</td>
</tr>
<tr>
<td>03</td>
<td>C</td>
</tr>
<tr>
<td>04</td>
<td>D</td>
</tr>
<tr>
<td>05</td>
<td>E</td>
</tr>
</tbody>
</table>

DCL REC CHAR(10) VAR; /* Record that is output */
DCL INREC CHAR(10) VAR; /* Input record */
DCL KEY CHAR(2) VAR; /* Key to record */
DCL OUTDATA(5) CHAR(8) VAR INIT('A','B','C','D','E');
DCL KEYDATA(5) CHAR(2) VAR INIT('01','02','03','04','05');

/********************************************************************
/* Execution -- WRITE OUT 5 RECORDS... */
/**/ /* PUT DIRECT must be used for new records, and PUT UPDATE must */
/* be used for existing records. Therefore, we use GET EQUAL */
/* to determine if the record is new or not. If new, then a PUT */
/* DIRECT will follow...if not, then a PUT UPDATE follows. */
/**/ /*
********************************************************************/
DO I = 1 TO HBOUND(OUTDATA,1); /* For 5 records */
  KEY=KEYDATA(I); /* Set key portion of record */
  REC=KEY||OUTDATA(I); /* Record must have key first */
  CALL CNMKIO(HLBPTR, /* Provide HLB pointer... */
              'GET_EQ', /* ... requesting a get */
              INREC, /* ... data is in inrec */
              10, /* ... 10 bytes max input */
              KEY, /* ... key is in key */
              'UPDATE'); /* ... this is an update */
  IF HLBRC=CNM_NOT_FOUND THEN DO;
    CALL CNMKIO(HLBPTR, /* Provide HLB pointer... */
                'PUT', /* ... requesting a put */
                REC, /* ... data is in rec */
                0, /* ... not used */
                KEY, /* ... key is in key */
                'DIRECT'); /* ... this is not an update */
    IF HLBRC=CNM_GOOD THEN CALL CNMSMSG(HLBPTR, /* Issue error message... */
                  'CNMKIO PUT REQUEST FAILED, RC='||CHAR(HLBRC), /* ... text of message */
                  'MSG', /* ... single line message */
                  'TASK', /* ... to the task */
                  ORIGIN->ORIG_BLOCK.ORIG_TASK); /* ...that invoked */
  END;
END;

Figure 35. Keyed File Access (PL/I) (Part 1 of 2)
ELSE
    CALL CNMKIO(HLBPTR, /* Provide HLB pointer... */
        'PUT', /* ... requesting a put */
        REC, /* ... data is in rec */
        0, /* ... not used */
        KEY, /* ... key is in key */
        'UPDATE'); /* ... this is an update */
    IF HLBRC=CNM_GOOD THEN
        CALL CNMSMSG(HLBPTR, /* Issue error message... */
            'CNMKKEYIO PUT REQUEST FAILED, RC='||CHAR(HLBRC), /* ... text of message */
            'MSG', /* ... single line message */
            'TASK', /* ... to the task */
            ORIGIN->ORIG_BLOCK.ORIG_TASK); /* ...that invoked */
    END;

END;

Figure 35. Keyed File Access (PL/I) (Part 2 of 2)

Coding a DST Installation Exit

Figure 36 on page 96 shows coding a NetView HLL installation exit routine that primes an empty VSAM database for a DST. If a VSAM database has not been primed (has at least one record), subsequent I/O requests fail.
PL/I High-Level Language Services

Figure 37 on page 97 shows coding an installation exit routine, DSIEX03, that sets a task global variable equal to the last time a command was entered on the system. If the last command was the PSNDDAT command, the task global variable is not set. The PSNDDAT command is used to receive the variable value. See "Sending Data (PL/I)" on page 100.

```pli
PPRMVDB: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/***********************************************************
/* Descriptive Name: High Level Language PL/I DSIXITVN Example */
/* */
/* Change Activity: */
/* date,author: description of changes */
/* */
/***********************************************************/

/***********************************************************
/* */
/* * Operand Declarations */
/* */
/***********************************************************/
DCL HLBPTR PTR; /* Pointer to the HLB */
%INCLUDE DSIPLI; /* Include the HLL macros */
DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
DCL ORIGBLCK CHAR(40); /* Area for the Orig Block */
DCL ORIGIN PTR; /* Pointer to the Orig Block */
DCL ADDR BUILTIN; /* Built-in function */
ORIGIN=ADDR(ORIGBLCK); /* Address the Orig Block */
/***********************************************************/
/* */
/* * Other Declarations */
/* */
/***********************************************************/
DCL KEY CHAR(2) VAR; /* 2-byte key of the record */
/***********************************************************/
/* */
/* * Execution - */
/* */
/* /* Create the record to initialize the VSAM database. The */
/* /* record will have a key of 0000 and a value of "Low rec". */
/* /* Setting the HLBRC to USERSWAP (8) will cause the contents */
/* /* of CMDBUF to be swapped into the database, thereby giving */
/* /* it an initial value, and enabling the subsequent VSAM I/O. */
/* /* */
/***********************************************************/
KEY='0000'X; /* Set key to low values */
CMDBUF=KEY||'Low rec'; /* Build the data record */
HLBRC=USERSWAP; /* Set USERSWAP rc */
END PPRMVDB;
```

Figure 36. Coding a DST Installation Exit (PL/I)

Coding an Installation Exit
Coding the WAIT FOR DATA Function

PWATDAT and PSNDDAT are standard command processors, which can be invoked from an installation exit. The following sections describes sending messages with a type of request, waiting on the response, and parsing the results. The code in this example is the PWATDAT command.

The syntax of the command is:

```pli
PSETTG: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);

/* Descriptive Name: High Level Language PL/I DSIEX03 Example */

/* Change Activity: */
/* date,author: description of changes */
/* */

/* Operand Declarations */
/* */

DCL HLBPTR PTR; /* Pointer to the HLB */
%INCLUDE DSIPLI; /* Include the HLL macros */
DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
DCL ORIGBLCK CHAR(40); /* Area for the Orig Block */
DCL ORIGIN PTR; /* Pointer to the Orig Block */
DCL ADDR BUILTIN; /* Built-in function */
ORIGIN=ADDR(ORIGBLCK); /* Address the Orig Block */

/* Other Declarations */
/* */

DCL TIME CHAR(256) VAR; /* Time last command entered */

/* Execution */
/* */

IF INDEX(CMDBUF,'PSNDDAT')¬=1 THEN /* Command other than PSNDDAT? */
DO; /* Yes... */
   CNMINFOC /* Gather NetView information... */
   ITEM('TIME') /* ...what time is it? */
   DATA(TIME) /* ...answer goes here */
   LENG(256); /* ...length of time */
   CNMVARPOOL FUNC('PUT') /* Put answer in task global... */
   NAME('LAST_COMMAND_TIME') /* ...by the name of... */
   POOL('TGLOBAL') /* ...task global pool */
   DATA(TIME); /* ...information in TIME */
END;

HLBRC=USERASIS; /* Clear RC */
END PSETTG;
```

Figure 37. Coding an Installation Exit (PL/I)
**PL/I High-Level Language Services**

**PWATDAT**

```
PWATDAT taskname
```

*Figure 38. Syntax for PWATDAT*

**Where:**

*taskname*

Specifies the task global variable set by the installation exit and retrieved by the PSNDDAT command.

*Figure 39* shows the flow of the WAIT FOR DATA function.

**Requesting Data**

- The requesting OST invokes PWATDAT and specifies the target OST to which to send the request.
- PWATDAT uses CNMSMSG to send the request to the target OST.
- The requesting OST issues a WAIT FOR DATA.
- The WAIT FOR DATA is satisfied and a message is issued.

**Sending Data**

- The PSNDDAT command is invoked on the target OST and finds the task global variable set by DSIEX03.
- CNMSMSG is invoked to send the value retrieved. The type used in the CNMSMSG is DATA.

*Figure 39. Flow of the WAIT FOR DATA Function (PL/I)*

**Requesting Data (PL/I)**

The declaration [*Figure 40 on page 99*] shows how to request data. The example finds the last time that a command was entered on the given operator station task (OST). A task global variable, LAST_COMMAND_TIME, is set by DSIEX03. See "Coding an Installation Exit" on page 94.
This indefinite value is retrieved by the PSNDDAT command that is invoked on the target task. See "Sending Data (PL/I)" on page 100.

```
DCL GETBLOCK CHAR(40), /* Area for the Orig Block */
    NEWMSG CHAR(256) VAR, /* Message sent from PSNDDAT */
    TARGTASK CHAR(8) VAR, /* Task of inquiry */
    TARGTASKF CHAR(8); /* Task of inquiry */
DCL PARMCNT FIXED BIN(31); /* Number of parms scanned */
CNMSSCAN DATA(CMDBUF) /* Scan the input command... */
    FORMAT('%*S%S') /* ...skip the command */
    COUNT(PARMCNT) /* ...number of parms */
    PI(TARGTASK); /* ...target task */
IF PARMCNT=1 THEN /* Was the target task entered? */
    DO; /* Syntax ok... */
        TARGTASKF=TARGTASK; /* Put into fixed length string */
        CNMSENDMSG DATA('PSNDDAT') /* Invoke PSNDDAT command */
            MSGTYPE('REQUEST') /* ...type is request */
            DESTTYPE('TASK') /* ...on a task */
            DEST(TARGTASKF); /* ...specified by input */
    CNMCOMMAND DATA('WAIT 120 SECONDS FOR DATA');
    IF HLBC ¬= CNM_DATA_ON_WAIT THEN /* Wait successful ? */
        CNMSENDMSG /* No... */
            DATA('Wait for data abnormally ended') /*...text */
            MSGTYPE('MSG') /* ...message */
            DESTTYPE('OPER'); /* ...to the operator */
    ELSE /* Wait was successful */
        DO; /* Process the results */
            CNMGTDATA FUNC('GETMSG') /* Read in the response... */
                QUEUE(DATQ) /* ...on the data queue */
                DATA(NEWMSG) /* ...read into NEWMSG variable */
                LENG(256) /* ...give plenty of room */
                ORIGIN(GETBLOCK); /* ...provide a different org blk*/
        CNMSENDMSG /* Inform user... */
            DATA(NEWMSG) /* ...message is in NEWMSG */
            MSGTYPE('MSG') /* ...message */
            DESTTYPE('OPER'); /* ...to the operator */
        END; /* of process the results */
    END; /* of Syntax ok */
ELSE /* Target task not entered... */
    CNMSENDMSG /* Inform user... */
        DATA('Target task required') /*...Syntax error */
        MSGTYPE('MSG') /* ...message */
        DESTTYPE('OPER'); /* ...to the operator */
```

Figure 40. Requesting Data (PL/I)
The purpose of the code in Figure 41 on page 101 is to find the last time that a
c command was entered on the given task. A task global variable,
LAST_COMMAND_TIME, is set by DSIEX03. See "Coding an Installation Exit" on
page 96.

This value is retrieved by the PSNDDAT command that is invoked by the
PWATDAT command on the target task. See "Coding the WAIT FOR DATA
Function" on page 97. This command processor is executed when the PSNDDAT
command is invoked at the requesting OST.
Automating MSUs

Figure 42 on page 103 shows a routine to send an MSU directly to the automation table for evaluation.
Translating Code Points

Figure 43 on page 103 shows how to translate a numeric code point value into its corresponding EBCDIC text.
Registering Applications with MS Transport and Operations Management

Figure 44 on page 104 shows a routine to register an application with both the MS transport and the operations management application.
PL/I High-Level Language Services

/***************************************************************************/
/* Declare variables for the program. */
/***************************************************************************/
DCL MSGTOOP CHAR(80) VARYING; /* CNMSENDMSG buffer */
DCL CMD_RETCODE FIXED BINARY(31,0); /* Return code */
DCL ANAME CHAR(B) VARYING; /* appl name entered */
DCL CNAME CHAR(B) VARYING; /* command name entered */
DCL FPCATE CHAR(B) VARYING; /* focal pt category entered */
DCL REPL CHAR(B) VARYING; /* replace value entered */
DCL RTYPE CHAR(B) VARYING; /* registration type entered */
DCL FP CHAR(B) VARYING; /* focal point value entered */
DCL NOTI CHAR(B) VARYING; /* notify value entered */
DCL APPLNAME CHAR(B); /* appl name used in CNMRGS */
DCL CMDNAME CHAR(B); /* command name used in CNMRGS */
DCL FPCATVAL CHAR(B); /* focal pt category used in RGS */
DCL REPLACE CHAR(B); /* replace value used in CNMRGS */
DCL FOCPT CHAR(B); /* focal pt value used in RGS */
DCL NOTI VAL CHAR(B); /* notify value used in CNMRGS */
DCL REGTYP CHAR(B); /* registration type */
DCL TOKENS FIXED BINARY(31,0); /* no of input entered */

/***************************************************************************/
/* Misc. declares - BUILT-IN functions, labels, etc. */
/***************************************************************************/
DCL CHAR BUILTIN; /* Binary to char. function */
DCL LENGTH BUILTIN;
DCL SUBSTR BUILTIN;

/***************************************************************************/
/* Initialization */
/***************************************************************************/
APPLNAME = '
CMDNAME = '
FPCATVAL = '
REPLACE = '
REGTYP = 'BOTH'
FOCPT = 'NO'

Figure 44. Registering Applications with MS Transport and Operations Management (PL/I)
(Part 1 of 3)
/*******************************************************************************/
/* Scan input command buffer and get all parameters. */
/*******************************************************************************/
CMMSSCAN DATA(CMDBUF)
 FORMAT('%*S' ||
 'S' ||
 'S' ||
 'S' ||
 'S' ||
 'S' ||
 'S' ||
 'S')
 COUNT(TOKENS) /* Parse out the appl name */
 P1 (ANAME) /* Parse out the command name */
 P2 (CNAME) /* Parse out the FP category */
 P3 (FPCATE) /* Parse out the Focal Pt value */
 P4 (FP ) /* Parse out the Replace value */
 P5 (REPL) /* Parse out the NOTIFY value */
 P6 (NOTIFY) /* Parse out the Reg. type */
 P7 (RTYPE ); /* Parse out the Reg. type */
/*******************************************************************************/
/* Make sure application name and command name are given */
/*******************************************************************************/
IF TOKENS < 2 THEN
   DO;
      MSGTOOP = 'APPLICATION NAME AND COMMAND NAME ARE REQUIRED';
      CNMSENDMSG DATA (MSGTOOP)
      MSGTYPE ('MSG ')
      DESTTYPE('OPER ');
      HLBRC = CNM_BAD_INVOCATION; /* registration not done */
   END;
 ELSE
   DO;
      APPLNAME = ANAME; /* save appl name */
      CMDNAME = CNAME; /* save command name */
      IF LENGTH(FPCATE) > 0 & SUBSTR(FPCATE, 1, 1) ¬= '.' THEN
         FPCATVAL = FPCATE; /* save FP category if entered */
      IF LENGTH(FP) > 0 & SUBSTR(1, 1) ¬= '.' THEN
         FOCPT = FP; /* save focal pt value if entered */
      IF LENGTH(REPL) > 0 & SUBSTR(REPL,1, 1) ¬= '.' THEN
         REPLACE = REPL; /* save replace value if entered */
      IF LENGTH(NOTIFY) > 0 & SUBSTR(NOTIFY, 1, 1) ¬= '.' THEN
         NOTI_VAL = NOTIFY; /* save notify value if entered */
      IF LENGTH(RTYPE) > 0 & SUBSTR(RTYPE, 1, 1) ¬= '.' THEN
         REGTYPE = RTYPE; /* save reg type if entered */
   END;
*******************************************************************************/
/* Save parsed value and invoke CNMREGIST service */
/*******************************************************************************/
ELSE
   DO;
      APPLNAME = ANAME; /* save appl name */
      CMDNAME = CNAME; /* save command name */
      IF LENGTH(FPCATE) > 0 & SUBSTR(FPCATE, 1, 1) ¬= '.' THEN
         FPCATVAL = FPCATE; /* save FP category if entered */
      IF LENGTH(FP) > 0 & SUBSTR(1, 1) ¬= '.' THEN
         FOCPT = FP; /* save focal pt value if entered */
      IF LENGTH(REPL) > 0 & SUBSTR(REPL,1, 1) ¬= '.' THEN
         REPLACE = REPL; /* save replace value if entered */
      IF LENGTH(NOTIFY) > 0 & SUBSTR(NOTIFY, 1, 1) ¬= '.' THEN
         NOTI_VAL = NOTIFY; /* save notify value if entered */
      IF LENGTH(RTYPE) > 0 & SUBSTR(RTYPE, 1, 1) ¬= '.' THEN
         REGTYPE = RTYPE; /* save reg type if entered */
   END;
*******************************************************************************/
/* Make sure application name and command name are given */
*******************************************************************************/
*/ Figure 44. Registering Applications with MS Transport and Operations Management (PL/I) (Part 2 of 3)
Registering Applications with the High Performance Transport

Figure 45 on page 107 shows how to register an application with the high performance transport.
DECLARE variables for the program:

DCL MSGTOOP CHAR(80) VARYING; /* CNMSSENDMSG buffer */
DCL CMD_RETCODE FIXED BINARY(31,0); /* Return code */
DCL ANAME CHAR(8) VARYING; /* appl name entered */
DCL CNAME CHAR(8) VARYING; /* command name entered */
DCL LGMOD CHAR(8) VARYING; /* logmode entered */
DCL REPL CHAR(8) VARYING; /* replace value entered */
DCL RTYPE CHAR(8) VARYING; /* registration type entered */
DCL APPLNAME CHAR(8); /* appl name used */
DCL CMDNAME CHAR(8); /* command name used */
DCL LOGMOD CHAR(8); /* focpt category used */
DCL REPLACE CHAR(8); /* replace value used */
DCL REGTYPE CHAR(8); /* registration type used */
DCL TOKENS FIXED BINARY(31,0); /* no of input entered */

Misc. declares - BUILT-IN functions, labels, etc.

DCL CHAR BUILTIN; /* Binary to char. function */
DCL LENGTH BUILTIN;
DCL SUBSTR BUILTIN;

Initialization

APPLNAME = ' ';
CMDNAME = ' ';
LOGMOD = ' ';
REPLACE = ' ';
REGTYPE = 'REGAPPL ';

Figure 45. Registering Applications with the High Performance Transport (PL/I) (Part 1 of 3)
PL/I High-Level Language Services

/******************************************************************
/* Scan input command buffer and get all parameters. */
/******************************************************************
CNMSSCAN DATA(CMDBUF)
FORMAT('%*S' ' %S' ' %S' ' %S' ' %S' ' %S')
COUNT(TOKENS) /* Parse out the appl name */
P1 (ANAME) /* Parse out the command name */
P2 (CNAME) /* Parse out the Logmode */
P3 (LNAME) /* Parse out the REPLACE value */
P4 (REPL) /* Parse out the Reg. type */
P5 (RTYPE ); /* Parse out the */

/******************************************************************
/* Make sure application name and command name are given */
/******************************************************************
IF TOKENS < 2 THEN
  DO;
    MSGTOOP = 'APPLICATION NAME AND COMMAND NAME ARE REQUIRED';
    CNMSENDMSG DATA (MSGTOOP)
      MSGTYPE ('MSG ')
      DESTTYPE('OPER ')
      HLBRC = CNM_BAD_INVOCATION; /* registration not done */
  END;

/******************************************************************
/* Save parsed value and invoke CNMHRGS service */
/******************************************************************
ELSE
  DO;
    APPLNAME = ANAME; /* save appl name */
    CMDNAME = CNAME; /* save command name */
    IF (LENGTH(LNAME) > 0 &
        SUBSTR(LNAME, 1, 1) == '.') THEN
      LOGMOD = LNAME; /* save Logmode if entered */
    IF (LENGTH(REPL) > 0 &
        SUBSTR(REPL, 1, 1) == '.') THEN
      REPLACE = REPL; /* save replace value if entered*/
    IF (LENGTH(RTYPE) > 0 &
        SUBSTR(RTYPE, 1, 1) == '.') THEN
      REGTYPE = RTYPE; /* save reg type if entered */
  END;

Figure 45. Registering Applications with the High Performance Transport (PL/I) (Part 2 of 3)
Sending an Alert over the MS Transport

Figure 46 on page 110 shows how to send a software alert over the MS transport.
**Sending an Alert over the High Performance Transport**

Figure 47 on page 111 shows how to send a software alert over the high performance transport.
%INCLUDE DSIPL1; /* Include the HLL macros */

DCL HLBPTR PTR; /* Pointer to the HLB */
DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
DCL ORIGBLCK CHAR(40); /* Area for the Origin Block */

DCL CMD_RETCODE FIXED BIN(31,0); /* return code from CNMSENDMU */
DCL MSGTOOP CHAR(80) VARYING; /* CNMSENMDBG buffer */
DCL ALERT_MSU CHAR(256) VARYING; /* alert to be sent */
DCL CORR_AREA CHAR(53) VARYING; /* area for correlator retm */
DCL OAPPL CHAR(8); /* origin appl - USERAPPL */
DCL DNET CHAR(8); /* dest net - NETA */
DCL DLU CHAR(8); /* dest LU - CNM01 */
DCL RMT_CMD CHAR(8); /* dest appl - RMT_CMD */

DCL CHAR BUILTIN; /* Binary to char. function */
DCL LENGTH BUILTIN;
DCL SUBSTR BUILTIN;

Figure 47. Sending an Alert over the High Performance Transport (PL/I) (Part 1 of 2)
Sending an MDB to NetView for Processing

Figure 48 on page 113 shows how to send an message data block (MDB) to NetView for processing.

PL/I High-Level Language Services

```pli
/* Initialization */

ALERT_MSU = '0046121200420000'X ||
              '0B920000121010000001'X ||
              '10100000110E00040F1F2F4F54040'X ||
              '110303010905C14C5F1404040E3E0D7F1'X ||
              '069310011023'X ||
              '0C9606011022102304813110'X;

OAPPL = 'USERAPPL';
DNET = 'NETA ';
DLU = 'CNM01 ';
RMT_CMD = '30F5F5F5F5F5F5F5'X;

CNMMSMU DATATYPE (NONMDSMU)
          DATA (ALERT_MSU)
          CORRELAREA (CORR_AREA)
          ORIGAPPL (OAPPL)
          DESTNET (DNET)
          DESTLU (DLU)
          DESTAPPL (RMT_CMD)
          MUTYPE (REQUEST_WITHOUT_REPLY);

CMD_RETCODE = HLBRC;
MSGTTOOP = 'RETURN CODE FROM CNMMSMU IS' ||
            CHAR(CMD_RETCODE);

CNMSNDMSG DATA (MSGTTOOP)
          MSGTYPE ('MSG ')
          DESTTYPE ('OPER ');

HLBRC = CNM_GOOD;  /* Successful completion */

END PHSENDMU;
```

Figure 47. Sending an Alert over the High Performance Transport (PL/I) (Part 2 of 2)
/* NetView High-Level Language include files */
/* Include the HLL macros */
DCL HLBPTR PTR; /* Pointer to the HLB */
DCL CMDBUF CHAR(*) VARYING; /* Buffer for the command */
DCL ORIGBLCK CHAR(40); /* Area for the Origin Block */
DCL MDBPTR PTR;
DCL SOPTR PTR;
DCL MYCORR CHAR(16);
DCL MYMDB CHAR(300);
DCL MYSRC CHAR(50);
DCL INFOMSG CHAR(120) VARYING;

Figure 48. Sending an MDB to NetView for Processing (PL/I) (Part 1 of 2)
PL/I High-Level Language Services

Figure 48. Sending an MDB to NetView for Processing (PL/I) (Part 2 of 2)
Part 4. Writing a C Program

Chapter 8. Coding Your C Program-Interfaces and Restrictions  ........................................ 119
  Initial Parameters ........................................ 119
  C Run-Time Options .................................... 119
    #pragma runopts .................................. 120
    LE/390 Preinitialization Defaults .................. 120
    PSSTACK and PHEAP .................................. 121
    LEOPTS Static External Variable ................... 121
  Parameters Passed to HLL Service Routines ........ 121
    Pointer Variables ................................ 122
    Integer Variables ................................ 123
    Fixed-Length Character Strings .................... 124
    Varying Length Character Strings ................. 124
  Cnmvlc: Converting a String to a Varying Length Character String .................. 126
  Cnmnvlc: Converting a String to a Varying Length Character String Using Length .... 127
  Control Blocks and Include Files .................... 128
  C Input/Output Considerations ....................... 129
  C Run-Time Considerations .......................... 130
  Considerations for HLL Command Processors ........ 131
  Return Codes ...................................... 131
  Restrictions for HLL Programs Written in C Language ..................................................... 131
    Restrictions for C Programs Running in a Preinitialized Environment ..................... 132

Chapter 9. C High-Level Language Services .................................................. 133
  C Sample Template ................................... 133
  Varying Length Character Strings ................... 136
    Cnmvlc .......................................... 136
    Cnmnvlc .......................................... 137
    Defining Varying Length Character Strings .......... 138
  Data Queue Management ................................ 138
  Sending Messages .................................... 139
  Invoking Synchronous Commands ..................... 140
  Sending Commands ................................... 141
  Waiting and Trapping ................................ 143
    Example of Using the TRAP and WAIT Commands (C) ..................... 143
    Example of Using the PIPE Command (C) .................. 146
  Retrieving Information ................................ 149
  Command List Variable Access ....................... 151
    Accessing Common Global Variables (C) ............. 152
    Accessing Stem Variables Using the PIPE Command (C) .................. 154
  Using Locks ........................................ 155
  Operator Input ...................................... 158
  VIEW Command Processor ................................ 159
  Message Processing .................................. 161
  Command Authorization Checking ...................... 163
    Command Authorization Checking Using
      CMDAUTH=TABLE .................................. 164
      In DSIPARM(CMDTABLE) ......................... 164
    Command Authorization Checking Using
      CMDAUTH=SAF ................................... 164

In RACF ............................................. 164
  Command Authorization Checking Sample
    Code ............................................. 166
  Altering Data ....................................... 167
  Storage Access ..................................... 169
  Data Set Access .................................... 171
  Communicating with Devices ......................... 173
  Performing I/O on a VSAM File (Keyed File Access) ........................................... 178
  Coding a DST Installation Exit ....................... 181
  Coding an Installation Exit ........................ 183
  Coding the WAIT FOR DATA Function ................ 185
    Requesting Data .................................. 186
    Sending Data ..................................... 188
    Automating Management Services Units (MSUs) ........ 190
    Translating Code Points ........................... 191
    Registering Applications with MS Transport and Operations Management ........... 192
    Registering Applications with the High Performance Transport ..................... 193
    Sending an Alert Over the MS Transport ............ 193
    Sending Data Over the High Performance Transport ........................................ 195
    Sending an MDB to NetView for Processing ........ 197
Chapter 8. Coding Your C Program-Interfaces and Restrictions

This chapter describes how to write high level language (HLL) command processors and installation exits in the C language. This chapter also describes the appropriate interfaces and language dependent restrictions.

C language command processors and installation exits are supported for IBM C/370 V2R1M0 (no preinitialization support) and AD/Cycle C/370.

Initial Parameters

Every HLL program written in C must have exactly one function main that declares the parameters argc and argv. The first parameter, argc, is an integer value that indicates the number of pointers in the array argv. NetView does not use argv.

argv is an array of pointers. In the normal C environment, each element in argv points to an argument in the command line. In the NetView environment, elements 1 through 3 point to the EBCDIC representation of the initial parameters (Hlbptr, Cmdbuf and Origblk) passed to main from the NetView program. The initial parameters must be converted to hexadecimal using sscanf. The original command line is passed to the user’s program in Cmdbuf. Chapter 9, C High-Level Language Services” on page 133 contains a sample template for coding the main function and defining and converting the initial parameters. The descriptions of the initial parameters are:

Cmdbuf
A varying length character field that contains the command or message that drives this program.

If this program is driven as an installation exit (other than DSIEX02A), this field contains the message that drives this exit. If driven as DSIEX02A, Cmdbuf does not contain any useful information, and you must retrieve the message from the initial data queue (IDATAQ).

Hlbptr
A 4-byte pointer field containing the address of the HLB control block (DSICHLB). The HLB control block is the HLL API interface block that is used to communicate between the HLL service routines and HLL programs in the NetView environment. This pointer is required on all HLL service routine invocations. The HLB control block is unique for each invocation of an HLL program. NetView automatically inserts HLBPTR for the C invocation format.

Origblk
A 40-byte structure that describes the origin of the request that caused the execution of this program. Origblk is mapped by DSICORIG.

C Run-Time Options

The method you use to specify C/370 run-time options depends on the type of C/370 support you use for your programs. The way you specify C run-time options, therefore, depends on the following:

- Compiler (IBM C/370 or AD/Cycle C/370)
- Libraries (IBM C/370 or LE/390)
- Preinitization Support (Enabled or Disabled)

In addition, you need to consider information about:
Running Non-Preinitialized, either intentionally or due to availability. The type of run-time option, whether the option is for a general option or a storage option.

Use Table 19 as a starting point to familiarize yourself with the specific methods of specifying the run-time options for your program. The table indicates the methods to use based on the interface module that is being link-edited with a program. Following the table, each method for specifying run-time options is described in more detail.

Table 19. Interface Modules and How to Specify C/370 Run-Time Options

<table>
<thead>
<tr>
<th>Interface Module</th>
<th>Library and Program Type</th>
<th>General Run-Time Options</th>
<th>Storage Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIEXANC</td>
<td>LE/390 Non-Preinitialized</td>
<td>#pragma runopts</td>
<td>#pragma runopts</td>
</tr>
<tr>
<td>DSIEXAPC</td>
<td>LE/390 Non-Preinitialized</td>
<td>LE/390 Preinitialized Defaults or LEOPTS static external variable</td>
<td>LEOPTS static external variable</td>
</tr>
<tr>
<td>DSIEXAPC</td>
<td>LE/390 Preinitialization-enabled</td>
<td>LE/390 Preinitialized Defaults</td>
<td>PSTACK, PHEAP</td>
</tr>
</tbody>
</table>

#pragma runopts

Run-time options can be specified with #pragma runopts preprocessor directive for both the IBM C/370 and AD/Cycle C/370 compilers. However, the specific options that can be specified with each compiler are different.

For example, for IBM C/370, include:
#pragma runopts (NOEXECOPS,NOSTAE,NOSPIE,ISASIZE(4K),ISAINC(4K))

In contrast, for AD/Cycle C/370, include:
#pragma runopts( STACK(4K,4K,ANY,FREE) HEAP(4K,4K,ANY,FREE,8,0) )
#pragma runopts( BELOWHEAP(8,0,FREE) LIBSTACK(8,0,FREE) )
#pragma runopts( INTERRUPT(OFF) NOTEST(ALL,*,* ) STORAGE(,,0) )
#pragma runopts( TRAP(OFF) MSGQ(1) ALL31(ON) )

The run-time options for executing C/370 programs in the NetView environment are illustrated in the C language coding template in "Chapter 9. C High-Level Language Services" on page 133.

IBM C/370 programs must run with the NOSTAE and NOSPIE options when running in the NetView environment. AD/Cycle C/370 programs must run with the TRAP(OFF) option. Running with STAE, SPIE, or TRAP(ON) options causes unpredictable results if error recovery is necessary.

To achieve optimum performance, use the REPORT or RPTSTG option until accurate ISA, STACK, and HEAP sizes are determined.

Refer to the run-time storage section of the C/370 library or the LE/390 library for more information.

LE/390 Preinitialization Defaults

The following C run-time options are automatically set for programs that request to run in a preinitialized environment. Except for the STACK and HEAP values, you cannot change the values. Values for pstack and pheap are taken from the values specified for PSTACK and PHEAP on the HLLENV command.
PSTACK and PHEAP
STACK and HEAP storage values are set using the PSTACK and PHEAP keywords of the HLLENV command. The initial value for PSTACK and PHEAP are both 4096 bytes. These values can be changed dynamically with the HLLENV command.

Example of Overriding PSTACK and PHEAP Values: The following example shows how to override the initial values for PSTACK and PHEAP for a program that requests to run in a preinitialized environment:

HLLENV CHANGE,PSTACK=2048,PHEAP=2048,TYPE=IBMADC

LEOPTS Static External Variable
When your program uses the DSIEXAPC interface module and explicitly forces non-preinitialization through the combination of the HLLENV DEFAULT value and the HLOPTS bits, you may specify the run-time options using the LEOPTS static external variable.

When you define LEOPTS in your program, the run-time options specified are used instead of the defaults described in “LE/390 Preinitialization Defaults” on page 120 and “PSTACK and PHEAP”.

Example of Specifying LEOPTS Values: You can override the default LE/390 Preinitialization, PSTACK, and PHEAP run-time options in a C/370 program as shown in the following LEOPTS declaration.

```
#pragma variable(LEOPTS,NORENT)
extern char LEOPTS??(255??) = "STACK(4K,4K,ANY,FREE) HEAP(4K,4K,ANY,FREE,8,0) TRAP(OFF) MSGQ(1)"
"BELOWHEAP(8,0,FREE) LIBSTACK(8,0,FREE) THREADHEAP(8,0,ANY,FREE)"
" INTERRUPT(OFF) NOTEST(ALL,*,* ) STORAGE(,,0) ALL31(ON)"
" PLITASKCOUNT(1)";
```

Parameters Passed to HLL Service Routines
Four types of parameters can be passed to HLL service routines. Each of the parameters, described throughout “Chapter 11. Service Reference” on page 209, falls into one of these categories:
- Pointer variables
- Integer variables
- Fixed-length character strings
- Varying length character strings

A description of each of these parameter types follows. The sections describe how you can declare, initialize, and pass each of these parameter types to the HLL service routines. This chapter also provides examples and recommendations for writing HLL programs in C. These examples are not complete, they are included to emphasize how you should declare, initialize, and pass service routine parameters.
For complete examples of user-written HLL programs, refer to the HLL samples shipped with NetView, or see "Appendix D. C Samples" on page 345.

**Pointer Variables**

A pointer variable is a 4-byte pointer field containing an address. All HLL service routines require one pointer variable called *Hlbptr*. NetView calculates the value of *Hlbptr* and passes it to the HLL command processor or installation exit. Once *Hlbptr* is received as an initial parameter, do not assign a value to *Hlbptr*. This is the only parameter of this type that you do not have to assign.

**Note:** You do not need to specify the *Hlbptr* parameter when coding HLL service routine invocations in C. *Hlbptr* is inserted for you before the HLL service routine is invoked.

If an HLL service routine is expecting an address in a pointer field, you are responsible for assigning a value to that pointer field before invoking the HLL service routine. For C, use the & (address) operator when passing pointer variables to HLL service routines rather than creating a separate pointer variable for this purpose. This ensures that the pointer variable is assigned a value before invoking the HLL service routine.

**Figure 49** shows an example of using pointer variables in C. Some of the steps are explained in more detail following the figure.

```c
#define VARTOVAR "VARTOVAR" /* VARTOVAR constant */
Dshlb *Hlbptr; /* HLB pointer MUST BE DEFINED */
Dsivarch *srcptr; /* Pointer to source buffer */
Dsivarch *dstptr; /* Pointer to destination buffer */

int dstlen; /* Length of destination buffer */
Dsivarch srcbuf; /* Source buffer */
Dsivarch dstbuf; /* Destination buffer */

srcptr = &srcbuf; /* Address of source buffer */
dstptr = &dstbuf; /* Address of destination buffer */
dstlen = 255;

Cnmcps(srcptr,dstptr,dstlen,VARTOVAR); /* Copy buffer */
```

**Figure 49. Using Pointer Variables in C**

The descriptions for the steps shown in **Figure 49** are:

2. *Hlbptr* is defined as a pointer variable. You must not assign a value to *Hlbptr* or include it in the Cnmcps invocation. NetView inserts *Hlbptr* in the parameter list for you during the compilation’s preprocessor phase.

3. *srcptr* is defined as a pointer to a structure of type *Dsivarch* (varying length character string).

4. *srcptr* is assigned the address of the source buffer (*srcbuf*) to be used as an operand to Cnmcps.

5. *srcptr* is passed as a parameter to Cnmcps.

Using the & (address) operator in C eliminates the need to define pointer variables. Note the use of a string constant instead of the VARTOVAR constant in this example:
Integer Variables

Several of the HLL service routines require the user to pass a 4-byte integer value to be used as a length, count, queue number, and so on. Figure 50 illustrates the use of integer variables in the HLL environment. Some of the steps are explained in more detail following the figure.

The descriptions for the steps shown in Figure 50 are:

2. `sptoken` is defined as a 4-byte integer (int).
3. `spleng` is defined as a 4-byte integer (int).
4. `sptoken` is initialized to 0. A value is returned in `sptoken` upon successful completion of the `Cnmpool` invocation.
5. `spleng` is assigned a value of 256 to be used in the call to `Cnmpool`
6. `Cnmpool` is invoked using `&sptoken` and `spleng` as operands. The value of `sptoken` is returned to the user upon successful completion of the `Cnmpool` service.

**Note:** All of the integer variables are specified by name except for `sptoken`. `sptoken` is specified using the `&` (address) operator. In C, all operand fields that return values to the user's program must specify a pointer to that operand field when invoking an HLL service routine. Otherwise, you do not see the changes made to that variable upon successful completion of the HLL service routine. Using the `&` (address) operator ensures that the value is returned to the calling program.

The `&` (address) operator is also used for `spname`. This is explained in detail in "Varying Length Character Strings" on page 124.
C Code Interfaces and Restrictions

Fixed-Length Character Strings

The majority of the HLL service routines require you to pass one or more fixed-length character strings as arguments.

C string constants for most of the fixed-length character strings are provided in DSICCONS. DSICCONS is optional and can be tailored to your specific needs. The following steps correlate to Figure 50 on page 123.

1. `spfunc` is defined as a 9-byte character array (8 bytes + \0 null character).
2. `spfunc` is assigned a value of "ALLOC " to be used in the `Cnmpool` invocation.
3. `Cnmpool` is invoked using the `spfunc` parameter. `spfunc` can be assigned a value, defined with a preprocessor define statement (see `VARTOVAR` in Figure 49 step 1), or passed as a string constant as shown in the next example:
   ```c
   Cnmpool("ALLOC ",&sptoken,&spname,sppleng,sppricnt,spsecnt,spclass); /* Allocate Subpool */
   ```

C does not pad character strings with blanks. You must pad the fixed-length character strings with blanks, if necessary.

Also, all character strings must be delimited by the null character (\0) which is the end-of-string character in C. Enclosing text in double quotes ensures that the null character is appended to the last byte of the character string. As a result, when using a character array to represent a fixed-length character string, you must define the character array’s length to be one character greater than the length expected by the HLL service routine. Avoid using character arrays to represent fixed-length character strings whenever possible. You can use any of the alternative methods mentioned previously to ensure that the fixed-length character string is padded with blanks and delimited by the null character (\0). For further explanation, refer to the C documentation.

HLL service routines use two types of origin blocks. The first type of origin block (`Origblck`) is a 40-byte structure that you must declare. This is a required initial parameter that was previously described in Initial Parameters on page 119. You should never alter this structure. See the C coding template in Chapter 9, C High-Level Language Services on page 133 for an example of how to define `Origblck`.

The user specifies the second type of origin block (`adorigin`, `gdorigin`). The `adorigin` and `gdorigin` origin blocks must be at least 38 bytes long and must map to the first 38 bytes of the origin block structure (DSICORIG). You must define these origin blocks separately from the origin block that is required as an initial operand.

Varying Length Character Strings

Several of the HLL service routines require you to pass a varying length character string as an argument. Varying length character strings are currently supported in the PL/I environment, but not in C. As a result, a structure must be defined as follows to map the internal representation of a varying length character string:
Where:

**LL**  Is the 2-byte integer field containing the length of **TEXT**.

**TEXT**  Is the character string. 

**Dsivarch** is an example of a structure that maps a varying length character string. The maximum size of the buffer portion of this particular structure is 256 bytes; 255 bytes (maximum) of text plus 1 byte for the end-of-string character (\0). **Dsivarch** is in DSICVARC and is included by DSIC. The structure consists of two parts:

- **size**  A 2-byte (short) integer field that contains the size of the character array represented by **buffer**

  **Note:** The end-of-string character (\0) is not included in this size but should delimit the character array.

- **buffer**  A 255-byte null-terminated character array, delimited by the end-of-string character (\0). A 256-byte field contains a 255-byte buffer plus a 1-byte end-of-string character.

You are responsible for creating structures like **Dsivarch** to represent varying length character strings passed to HLL service routines. You can use the size portion of the buffer to manipulate buffers that contain hexadecimal data. The presence of the null character at the end of the buffer enables you to use the buffer portion of the structure in other C functions that require the end-of-string character as a delimiter. You are responsible for ensuring that the end-of-string character delimits the buffer portion of the structure when necessary.

**Note:** HLL service routines that return data in varying length character strings do not delimit the data with the end-of-string character.

The NetView program provides two functions that let you manipulate varying length character strings in the C environment. **Cnmvlc** and **Cnmnvlc** calculate the value of the 2-byte size field and ensure that the buffer portion of the varying length character string is delimited by the end-of-string character (\0). These are the only functions you should use when initializing or altering the buffer portion of a varying length character string. If you choose to alter the contents of a varying length character string without using **Cnmvlc** or **Cnmnvlc**, you must update the size field and ensure that the buffer is null-terminated.

**Note:** **Cnmvlc** and **Cnmnvlc** are specific to C support only.

**Figure 51 on page 12d** shows how to use varying length character strings in C. Some steps are explained in more detail following the figure.
The descriptions for the steps shown in Figure 51 are:

1. `spname` is defined as a varying length character string.
2. `spname` is assigned the value `POOLNAME` using the `Cnmvlc` function.
3. `Cnmpool` is invoked with the operand `&spname`. When passing a structure as an operand, you must pass a pointer to the structure rather than the structure itself. In this example, the `&` (address) operator is used when passing `spname` to `Cnmpool`.

You could have defined a pointer variable, assigned the address of the structure to that pointer variable, and passed the pointer variable to `Cnmpool`. However, in that case you could forget to assign the pointer variable before passing it to the HLL service routine.

### Cnvlc: Converting a String to a Varying Length Character String

`Cnvlc` enables you to convert a C string to a varying length character string to be used in the NetView HLL environment. You can choose to provide a simple null-terminated string or a format string as input. If you specify formatting, you must provide a valid argument list. `Cnvlc` also converts the input string to hexadecimal, if desired. This option can be helpful when you code command processors that invoke `Cnmcnmi` and `Cnmkio`.

`Cnvlc` calculates the length of the converted string (not including the null terminator) and stores it in the 2-byte size field of the varying length character string structure. The null terminator is actually copied into the buffer portion of the structure even though it is not included in the size calculation. This ensures that the structure’s buffer portion is null-terminated so that it can be used by other C library routines.

A pointer to the converted varying length character string structure is returned to the caller on successful completion of this routine. If an error occurs, `Cnvlc`...
returns a null pointer. See "Chapter 9. C High-Level Language Services" on page 133 for examples showing the use of Cnmvlc.

The syntax for this function is:

```c
void *Cnmvlc(void *vstring, short convert, char *istring);
```

**Where:**

- **convert**
  - Is a 2-byte integer field containing the value 0 or 1, indicating whether the input string should be converted to hexadecimal. Constants NOHEXCNV and CNVTOHEX have been defined in DSICCONS, as follows:
  - 0 (NOHEXCNV)
    - Do not convert input string to hexadecimal.
  - 1 (CNVTOHEX)
    - Convert input string to hexadecimal.

- **istring**
  - An input string that follows the conventions specified for the format-string operand of the printf function in C. The istring variable must be null-terminated and can contain format specifications (designated by %). Provide an argument list if formatting is desired. Refer to the SAA® Common Programming library for more information.

- **vstring**
  - Is a varying length character string to receive the converted string. See "Varying Length Character Strings" on page 124 for more details.

**Notes:**

1. If you specify conversion to hexadecimal, all of the characters in the input string must represent valid hexadecimal data. Cnmvlc returns a void pointer if it encounters a character that cannot be converted to hexadecimal.
2. A null pointer is also returned if you do not specify a valid value for convert or if the format specifications cannot be resolved.
3. Some of the HLL services routines (such as Cnmcmni and Cnmkio) require you to move hexadecimal data into varying length character strings. This can often create a problem for the C programmer because of the probability that the null terminator (represented as hexadecimal zeros) will be interspersed throughout the hexadecimal data stream. In this situation, you should use CNMNVLC.
4. If istring contains a format string, you must provide a vstring that can contain the entire resolved format string, irrespective of hexadecimal conversion.

---

**Cnmnvlc: Converting a String to a Varying Length Character String Using Length**

Cnmnvlc enables you to convert a C string to a varying length character string to be used in the NetView HLL environment. This function is primarily used for moving hexadecimal data into varying length character strings and is particularly useful when coding HLL command processors that invoke Cnmcmni or Cnmkio.

The function provided by Cnmnvlc is very similar to that of Cnmvlc, except that you must pass a length field. Cnmnvlc uses the length field to determine how many characters to copy from the input string. Also, Cnmnvlc does not accept format specifications or an argument list. You can choose to convert an input string to hexadecimal if desired. If you specify hexadecimal conversion, the length operand should represent the length of the input string before it is converted.
C Code Interfaces and Restrictions

Once the copy function is complete, Cnmv1c stores the value of the length operand in the structure. The null terminator is copied into the buffer portion of the structure even though it is not included in the size calculation.

A pointer to the converted varying length character string structure is returned to the caller on successful completion of this routine. If an error occurs, Cnmv1c returns a null pointer. See [Chapter 9. C High-Level Language Services” on page 133] for examples of Cnmv1c.

The syntax for this function is:

```c
void *Cnmv1c(void *vstring, short convert, int length, char *istring);
```

**Where:**

- **convert**
  - Is a 2-byte integer field containing the value 0 or 1, indicating whether the input string should be converted to hexadecimal. Constants NOHEXCNV and CNVTOHEX are defined in DSICCONS, as follows:
    - 0 (NOHEXCNV)
      - Do not convert input string to hexadecimal.
    - 1 (CNVTOHEX)
      - Convert input string to hexadecimal.

- **istring**
  - Is an input string. If hexadecimal conversion is required, all of the characters in the input string must represent valid hexadecimal data.

- **length**
  - Is a 4-byte integer field specifying the number of bytes to copy from the input string. If hexadecimal conversion is required, length is the length of the input string before it is converted. length must be greater than zero.

- **vstring**
  - Is a varying length character string to receive the converted string. See [“Varying Length Character Strings” on page 124] for more details.

**Notes:**

1. Cnmv1c returns a void pointer if it encounters a character that cannot be converted to hexadecimal.
2. A null pointer is also returned if you do not specify a valid value for convert or length.

Control Blocks and Include Files

A number of control blocks and include files are required to invoke an HLL program written in C in the NetView environment. DSIC, which includes the rest of the files, is the main file necessary to compile HLL programs written in C. Optional include files are provided to assist you in coding and maintaining HLL programs. You can tailor DSIC, DSICCNM and DSICCONS to your needs.

**Note:** Tailoring files can lead to better performance in many cases. This is helpful in performance-sensitive environments such as installation exits.

The following include files are listed in [“Appendix C. C Language Control Blocks and Include Files” on page 327]:

- **DSIC** (Required) Must be included for all HLL programs written in C. DSIC includes all of the external HLL control blocks and include files needed to
C Code Interfaces and Restrictions

compile and run C programs in the NetView environment. See the C coding template in “Chapter 9. C High-Level Language Services” on page 133 for usage.

DSICCONS
(Optional) Defines constants that are helpful when coding HLL programs in C.

DSICVARC
(Required) Specifies C mapping of a varying length character string.

DSICHLB
(Required) Specifies C mapping of the NetView HLB control block.

DSICORIG
(Required) Specifies C mapping of the origin block of the request that caused the execution of the program currently running.

DSICPRM
(Required) Specifies C mapping of the NetView Bridge parameter control block.

DSICCALL
(Required) Specifies C definitions for HLL service routines.

DSICCNM
(Optional) Defines HLL return code constants for C.

C Input/Output Considerations

C provides several input and output routines that allow you to transmit data between main storage and auxiliary storage of a computer. C programs using such file input/output capabilities can run in the NetView environment. However, there are some important things to consider when doing file I/O in C.

Each file referenced from your C program correlates to a physical data set in auxiliary storage. You can specify the file name or a data definition name (ddname) when opening a file using fopen. If you specify a ddname, you must ensure that the appropriate data set is allocated before attempting to open the file. You can allocate under TSO or by using the NetView ALLOCATE command described in the NetView online help. NetView also provides a FREE command to deallocate a data set.

If the data set is allocated from TSO, you must also add a corresponding DD statement to the NetView startup procedure. The data definition name (ddname) in the DD statement must match the ddname specified in the call to the fopen library routine. The DD statement specifies a physical data set name (dsname) and gives its characteristics as shown in this example:

```
//OUTFILE DD DSN=MYPROG.OUTFILE, ...
```

A DD statement is not necessary if the data set is allocated using the NetView ALLOCATE command.

This example shows the use of file I/O in an HLL program written in C. The check for a null pointer is added to protect against a failure in fopen. This check is recommended when opening a file for I/O.
**C Code Interfaces and Restrictions**

```c
FILE *Outfd; /* Define file */

/*********************/
/* Check for error opening file for I/O. If fopen error occurs, */
/* issue an error message and exit program. */
/*********************/
if ((Outfd = fopen("dd:OUTFILE","w")) == NULL)
{
    Cnmvlc(&msgbuf,NOHEXCNV,"ERROR OPENING DATA FILE.");
    Cnmsmsg(&msgbuf,MSG,SYSOP,NULLCHAR);
    Hlbptr->Hlbrc = CNM_GOOD;
    exit();
}

fprintf(Outfd, ... /* Write to output file */
fclose(Outfd); /* Close output file */
```

*Figure 52. Example of File I/O in an HLL Program Written in C Language*

If you choose to write to a common output file from two or more C programs, the programs must coordinate access to the common file. You can do this by using NetView’s CNMLK service routine, if desired. If access is not coordinated, you can get a system ABEND 213.

Take care when attempting to share open files between two or more HLL programs. Sharing of open files must be coordinated between the sharing programs.

**Note:** PL/I and C cannot share an open file. However, a C program can read a file created by PL/I.

Certain C routines (such as `getchar` and `putchar`) are designed to perform functions on `stdin` and `stdout`. By default, `stdin` and `stdout` are directed to the terminal. These defaults are not valid and cause undetermined results if used in the NetView environment. You can perform terminal I/O using `WAIT FOR OPINPUT` and `CNMSMSG` as described in "Chapter 11. Service Reference" on page 209.

Refer to the SAA Common Programming library, the C/370 library, and the AD/Cycle C/370 library for more information about files and C language I/O.

**C Run-Time Considerations**

Most run-time errors detected in the C environment are handled by passing a return code or a null pointer back to the caller. Run-time errors that are detected by the operating system generate an interrupt signal that could normally be handled by coding a `signal` function in your program. However, because C language programs must run with the NOSTAE and NOSPIE or TRAP(OFF) options in a production NetView environment, the operating system is unable to generate such interrupts.

While debugging a C language program in a test NetView environment, you can use the STAE and SPIE or TRAP(ON) options until run-time problems are resolved. Run-time errors that are not detected by the operating system cause a diagnostic message to be written to `stderr`.
Considerations for HLL Command Processors

You must code a CMDMDL statement in DSICMD for each HLL command processor that you write. The CMDMDL type depends on the functions that your command processor performs. Keep in mind that some of the HLL services are useful only when executed under a DST.

The CMDMDL statement is described in the Tivoli NetView for z/OS Administration Reference.

There is no support for HLL command processors running as immediate commands (TYPE=I) or being pushed (with the macro DSIPUSH) as ABEND, LOGOFF, or RESUME routines.

Return Codes

Upon completion of an HLL service routine, the completion code from that service routine is stored in the return code field (Hlbrc) of the HLB control block. Check this field after each HLL service routine invocation. Also, use this field when passing return codes between HLL programs.

A return type of void is specified for each of the HLL service routines defined in DSICCALL. This indicates that the HLL service routines do not return values to you. You can check the completion code only by evaluating the return code field (Hlbrc) in the HLB.

For a complete list of HLL API return codes, see DSICCNM in Appendix C. See “Chapter 11. Service Reference” on page 203 for a list of return codes that apply to each HLL service routine.

In C, you can achieve normal termination by assigning a value to Hlbrc and issuing a return() statement as follows:

```c
Hlbptr->Hlbrc = CNM_GOOD; /* Successful completion */
return(); /* Return to caller */
```

Restrictions for HLL Programs Written in C Language

C functions not supported in the NetView environment are:

```c
system  Cnmcmd - recommended HLL alternative
```

You cannot use the following functions without redirecting stdin, stdout, and stderr as follows:

```c
getchar  Cnmgetd - recommended HLL alternative
getenv  Cnminfi, Cnminfc, Origblk, Cnmvars - recommended HLL alternatives
gets  Cnmgetd - recommended HLL alternative
printf  Cnmsmsg - recommended HLL alternative
putchar  Cnmsmsg - recommended HLL alternative
puts  Cnmsmsg - recommended HLL alternative
scanf  Use Cnmgetd to fetch data and sscanf to parse data
perror  Check return code and use Cnmsmsg
```

Special Considerations:
C Code Interfaces and Restrictions

1. The C signal function does not work for errors detected by the operating system.

2. You cannot use Cnmsmsg to display wide character strings. If you want to process wide character strings, redirect stdout and use the printf function.

Restrictions for C Programs Running in a Preinitialized Environment

Other restrictions for AD/Cycle C programs that run in a preinitialized environment include:

- Exit() statements are not valid.
- Invocations of assembler routines that contain an SVC LINK command are not valid.
Chapter 9. C High-Level Language Services

This chapter provides an example-oriented description of commands and services provided by the NetView program in support of the C language. Refer to the Tivoli NetView for z/OS Command Reference and "Chapter 11. Service Reference" on page 209 for more information.

C Sample Template

Figure 53 on page 134 is a coding template sample to be used when coding HLL programs in C. You can use this template, with your enhancements, to utilize NetView functions and commands. Use the additional examples in this chapter in conjunction with this template. You can find fully functional samples of NetView command procedures written in C in "Appendix D. C Samples" on page 343.
C High-Level Language Services

/********************************************************************/
/* */
/* 5697-B82 (C) COPYRIGHT IBM CORPORATION 1989, 1997 */
/* ALL RIGHTS RESERVED. */
/* */
/* IEBCOPY SELECT MEMBER=((CNMS4201,CTMPPLT,R)) */
/* */
/* (Explanations included in parentheses should be deleted) */
/* (after the pertinent information has been filled in. ) */
/* */
/* Descriptive Name: High-Level Language C Template */
/* (This is the more descriptive name or title of the module.) */
/* */
/* Function: */
/* Template for writing HLL modules in C. */
/* (This is the description of what the module does.) */
/* (It may be paragraph or pseudocode form. ) */
/* */
/* Dependencies: */
/* (List conditions that must be met in order for this) */
/* (module to perform. An example of this might be a ) */
/* (key data area that must already have been built. ) */
/* */
/* Restrictions: */
/* (List any limitations this module may have.) */
/* */
/* Language: C */
/* */
/* Input: */
/* 1) A pointer to a 4-byte field containing the address of */
/* the HLB control block. */
/* 2) A varying length character string containing the */
/* command or message which invoked this program. */
/* If this program was invoked as a command processor, */
/* this will be a command string. */
/* If this program was invoked as an installation exit */
/* (other than DSIEX02A), this will be a message string. */
/* When driven as DSIEX02A- */
/* this string will be empty and the message must */
/* be retrieved from the Initial Data Queue (IDATAQ). */
/* 3) A 40-byte structure which describes the origin of the */
/* request that caused execution of this program. */

Figure 53. C Sample Template (Part 1 of 3)
# C High-Level Language Services

/* Output: */ /* (Describe any output from this module.) */ /* */ /* Return Codes: returned in Hlbrc */ /* For Command Processors: */ /* 0 = normal exit */ /* -5 = cancelled */ /* (List any other return codes meaningful to this module.) */ /* For User Exits: */ /* 0 = USERASIS (Leave the contents of the message buffer unchanged) */ /* 4 = USERDROP (Drop the message buffer) */ /* 8 = USERSWAP (Change the contents of the message buffer) */ /* */ /* External Module References: */ /* (List modules that are called by this module.) */ /* */ /* Change Activity: */ /* date,author: description of changes */ /* (Keep a log of the changes made to this module for) */ /* (future reference. ) */ /* */ /* #pragma runopts (NOEXECOPS,NOSTAE,NOSPIE,ISASIZE(4K),ISAINC(4K)) */ /* */ /* Standard include files */ /* */ /* #include <stdio.h> /* Standard I/O header */ /* #include <string.h> /* String functions */ /* #include <stdefs.h> /* Standard definitions */ /* #include <stdlib.h> /* Standard library */ /* #include <stdarg.h> /* Standard args */ /* */ /* NetView High-Level Language include files */ /* */ /* #include "dsic.h" /* Include HLL macros */ /* */

Figure 53. C Sample Template (Part 2 of 3)
Varying Length Character Strings

"Chapter 8. Coding Your C Program-Interfaces and Restrictions" on page 119 reviewed the use of varying length characters strings with HLL command procedures. Cnmvlc and Cnmnvlc are provided for working with varying length character strings. Cnmvlc is useful when copying null-terminated text into a varying length character string and building RUs for the CNMI. Cnmnvlc is useful when dealing with data that has nulls in it and data that is not null-terminated.

In several of the samples and examples in this document, varying length character strings are defined as type Dsivorc, a pointer to a structure defining a 256-byte varying length character string. This is only for convenience. It is more efficient to use a varying length character string with a buffer size closer to that of your data.

The following examples show how to use Cnmvlc and Cnmnvlc, and how to define varying length character strings with different buffer sizes.

**Cnmvlc**

Cnmvlc is used to copy a C character string into a varying length character string.

This example copies the text “Hello World” into a varying length character string for use with Cnmsmsg. &msg is defined as a varying length character string.

```
Cnmvlc(&msg, /* character string... */
      NOHEXCNV,    /* do not convert to hex */
      /* put message in varying length */
); /* end of example */
```
"Hello World"); /* message */
Cnmsmsg(&msg, /* message text */
    MSG, /* type is message */
    OPER, /* send message to oper */
    NULLCHAR); /* not used */

Cnmvlc returns a pointer to the varying length character string into which data is being copied. Therefore, it can be embedded directly into calls to HLL service routines. Here is an example of a call to Cnmvlc embedded in a call to the Cnmsmsg service routine:

Cnmsmsg(Cnmvlc(&msg,NOHEXCNV,"Hello World"),MSG,OPER,NULLCHAR);

The next example copies an RU into a varying length character string and converts it to hexadecimal. ru is defined as a varying length character string and puname is a C character string containing the name of the PU where the RU is to be sent.

Cnmvlc(&ru, /* varying length char strng */
    CNVTOHEX, /* convert to hex */
    "8108100000000000E41038D00000111000068090281F10807E4D5C1D4C500";
    /* the RU */
Cnmcnmi(SENRPLY, /* expect a reply */
    &ru, /* RU built above */
    puname.buffer, /* to the PU name specified */
    180); /* timeout after 3 minutes */

Here is an example of using a format string with Cnmvlc. It shows the text “Day 1 of five” copied into a varying length character string. num is defined as an integer with a value of 1 and string is defined as a character string with a value of “five”.

Cnmvlc(&msg, /* varying len char strng */
    NOHEXCNV, /* do not convert to hex */
    "Day %d of %s", /* format string */
    num, /* substitute for %d */
    string); /* substitute for %s */
Cnmsmsg(&msg, /* message text */
    MSG, /* type is message */
    OPER, /* send message to oper */
    NULLCHAR); /* not used */

Cnmnvlc

Cnmnvlc is used to copy C character strings containing null data into varying length character strings.

Here an RU returned by the CNMI is copied into another varying length character string. data is defined as a varying length character string. getblock is defined as a structure of type ORIGBLCK, and msg is defined as a varying length character string.

Note: The RU in the next example can contain null data.

Cnmnvlc copies for the length specified regardless of the presence of nulls.

Cnmgetd(GETLINE, /* a single RU */
    &data, /* inti here */
Defining Varying Length Character Strings

Sometimes, you need to create varying length character strings other than `Dsvarch`. This example copies the text “Hello World” into a user-defined varying length character string:

```c
typedef struct /* create your own varying length*/
{ /* character string... */
 short size; /* ...2 byte size field */
 char buffer??(12??); /* ...12 byte buffer */
} myvlc; /* ...data type */

main()
{
 myvlc msg; /* msg is a varying length */
 /* character string of type myvlc*/
 Cnmvlc(&msg,NOHEXCNV,"Hello World"); /* defined varying length */
 /* character string */
 Cnmsmsg(&msg,MSG,AUTHRCV,NULLCHAR); /* send message to operator */
}
```

Sometimes you must pass a command buffer larger than 256 characters to a C program. In these cases, must change the declaration for `Cmdbuf` provided in the C template. The next example accepts a command buffer larger than 256 characters and sends it to the authorized receiver.

```c
typedef struct /* create your own varying length*/
{ /* character string... */
 short size; /* ...2 byte size field */
 char buffer??(300??); /* ...300 byte buffer */
} myvlc; /* ...data type */

myvlc *Cmdbuf; /* Cmdbuf is a varying length */
 /* character string of type myvlc*/

main(int argc, char *argv??(??)) /* receive parameter list from...*/
{
 /* ...NetView */
 sscanf(argv??(2??),"%x",&Cmdbuf); /* Convert Cmdbuf ptr to hex */
 Cnmsmsg(Cmdbuf,MSG,AUTHRCV,NULLCHAR); /* display message */
}
```

Data Queue Management

NetView uses several data and message queues to work in conjunction with HLL service routines. Information retrieved from these queues through the CNMGETD function can be manipulated to enhance your network manageability. These queues are defined for data and message management:
<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue Number</th>
<th>Function</th>
</tr>
</thead>
</table>
| TRAPQ      | 1            | Message queue. Contains trapped messages. See [TRAP Command \(\text{on page 218}\)](\#)
| OPERQ      | 2            | Operator input queue. See "GO Command" on page 213 and "QUEUE Command" on page 214 |
| DATAQ      | 3            | Data queue. Contains data sent from another HLL command processor or installation exit routine. See "CNMSMSG (CNMSENDMSG): Send Message or Command" on page 282 |
| IDATAQ     | 4            | Initial data queue. Contains the full message or MSU that invokes the HLL command processor through NetView automation. It also contains messages that drive DSIX02A and DSIX17. This is also the queue where an application command processor receives an MDS-MU from the NetView high performance transport, the MS transport, or operations management for an unsolicited request or asynchronous reply. For invocations from pipes, the IDATAQ contains the data flowing into the HLL command processor. |
| CNMIQ      | 5            | CNMI solicited data queue. Contains RUs solicited through the CNMI service routine. Chained RUs are treated like multiline messages. See "CNMCNMI (CNMI): CNMI Access Under a DST" on page 224 |
| MDSMUQ     | 6            | MDS-MU data queue. Contains MUs received as synchronous replies. The MDS-MUs are received from operations management, the MS transport, and the high performance transport. Refer to Tivoli NetView for z/OS Application Programmer’s Guide for more information. |

The examples in the following sections show how the preceding queues are used with HLL command procedures.

**Sending Messages**

[Figure 54 on page 140](\#) shows how to send messages.
Invoking Synchronous Commands

Figure 55 on page 141 shows an HLL command processor invoking another command. The command could be another HLL command, a command list, a VTAM command, or a NetView command.

C High-Level Language Services

Figure 54. Sending Messages (C)
Sending Commands

Figure 56 on page 143 shows sending a command to execute on another task. The command to be run under the other task could be another HLL command, a command list, a VTAM command, or a NetView command.

You can use this process to invoke commands under DSTs, other OSTs, or the PPT.
C High-Level Language Services

/******************************************************************************
/* Internal data definitions */
*******************************************************************************/

char oper1 ??(9??) = "OPER1 "; /* 8 char task name for Cnmsmsg */
Dsivarch logoff, /* used to store logoff command */
msgbuf; /* var len char strng for messages */

/*******************************************************************************/
/* */
/* Execution */
/* */
/*******************************************************************************/

/* copy command into varying */
/* length character string... */
Cnmvlc(&logoff, /* ...varying length string */
0, /* ...do not convert to hex */
"LOGOFF"); /* ...command to be copied */

/* send the command... */
Cnmsmsg(&logoff, /* ...text of the command to run */
COMMAND, /* ...this is a command */
TASK, /* ...run it on a task */
oper1); /* ...task name is oper1 */
if (Hlbptr->Hlbrc == CNM_GOOD) /* inform user of success */
{
/* copy message into varying */
/* length character string... */
Cnmvlc(&msgbuf, /* ...varying length string */
0, /* ...do not convert to hex */
"OPER1 Logoff scheduled successfully"); /* ...message */

/* issue message... */
Cnmsmsg(&msgbuf, /* ...text of message */
MSG, /* ...this is a message */
OPER, /* ...to the operator */
NULLCHAR); /* ...not used */
}
Waiting and Trapping

Figure 57 on page 144 uses the TRAP and WAIT commands to issue a command, trap the output, and respond depending on the output that is encountered. Figure 58 on page 147 provide similar function using the PIPE command. Both examples activate the given LU and issue an appropriate message.

Example of Using the TRAP and WAIT Commands (C)

The syntax for the command shown in Figure 57 on page 144 follows:

```c
CACTLU

Where:

`luname`

Is the name of the LU to be activated.
```
C High-Level Language Services

buahriq origptr; /* work block for Cnmgetd */
char *token, /* used to parse command buffer */
nodename??(9??); /* LU to activate */
Dsivarch command, /* varying len char strng for cmds */
msgbuf; /* varying len char strng for msgs */

/* retrieve node name from command buffer */
/* build vtam command to activate node */

if (token = NULL) /* node name invalid */
    if (token != NULL) /* if nodename specified */
        if (token != NULL)
            if (token != NULL)
                if (token != NULL)
                    if (token != NULL)
                        "TRAP AND SUPPRESS ONLY MESSAGES IST*);
                        /* command */
                        Cnmcmd(&command); /* issue command to trap VTAM messages*/
                        /************************* */
                        /* put command to wait for 10 sec in */
                        /* varying length character string... */
                        Cnmvlc(&command, /* ...varying length string */
                               0, /* ...do not convert to hex */
                               "V NET,ACT,ID=%s",nodename); /* ...command */
                        Cnmcmd(&command); /* issue command to activate node */
                        Cnmcmd(&command); /* issue command to wait for 10 secs */
                        Cnmgetd(GETMSG, /* ...function is get a message */
                                &msgbuf, /* ...result goes here */
                                255, /* ...max input length */
                                &origptr, /* ...must provide a work area */
                                TRAPQ, /* ...message is trapped */
                                1); /* ...get the first one */

Figure 57. Waiting and Trapping Using the TRAP and WAIT Command (C) (Part 1 of 3)
C High-Level Language Services

Figure 57. Waiting and Trapping Using the TRAP and WAIT Command (C) (Part 2 of 3)
Example of Using the PIPE Command (C)

The syntax for the command shown in Figure 58 on page 147 follows:

```
CACTPIP

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Where:

- `luname` is the name of the LU to be activated.
C High-Level Language Services

/* Internal data definitions */
Dsiorig origptr; /* Work block for Cnmgetd */
char *result, /* Used to parse command buffer */
    *token, /* Used to parse command buffer */
    nodename??(9??); /* LU to activate */
Dsivarch command, /* Varying len char strng for cmds */
    varnm, /* Varying len char strng for varnm */
    msgbuf; /* Varying len char strng for msgs */
/* Retrieve node name from command buffer */

token = strtok((char *) &(Cmdbuf->buffer)," "); /* parse command */
token = strtok(NULL," "); /* Buffer for LU name */
strncpy(nodename,token);
if (token != NULL) /* If nodename specified */
{
    /* Build PIPE command to issue user activation */
    Cnmvlc(&varnm, /* length character string */
            0, /* ...do not convert to hex */
            "VNETRESULT"); /* ...name of variable */
    Cnmvars(DCL, /* Declare a variable ... */
            ",", /* ...not needed */
            0, /* ...not needed */
            &varnm, /* ...specifies variable name */
            LOCAL); /* ...local variable pool */
    /* Build the PIPE command. */
    /* The VTAM stage issues the VARY command. */
    /* The CORRWAIT stage waits 10 seconds for each message to */
    /* return to the pipeline. */
    /* The TOSTRING stage selects messages to remain in the */
    /* pipeline up to and including one having 'IST093I' */
    /* in columns 1-7. */
    /* The TAKE stage selects the last message in the pipeline, */
    /* discarding all others. */
    /* The VAR stage writes the message to the variable specified. */
}

Figure 58. Waiting and Trapping Using the PIPE Command (C) (Part 1 of 3)
C High-Level Language Services

/* Copy command into varying */
Cnmvlc(&command, /* length string */
0, /* ...do not convert to hex */
"PIPE VTAM V NET,ACT,ID=%s%s",nodename,
" CORRWAIT 10"
" TOSTRING FIRST 1.7 /IST093I/
" TAKE LAST 1"
" VAR VNETRESULT");
Cnmcmd(&command); /* Issue PIPE command */
Cnmvars(GET, /* Read the variable ... */
&msgbuf, /* ...result goes here */
256, /* ...max input length */
&varnm, /* ...specifies variable name */
LOCAL); /* ...local variable pool */

result = strstr(msgbuf.buffer,"IST093I");
if (result != NULL) /* Did we find IST093I? */
{ /* Inform user activation worked */

/************************************************************/
/* Build message to inform user activation worked */
/************************************************************/

/* Copy message into varying length*/
/* character string... */
Cnmvlc(&msgbuf, /* ...varying length string */
0, /* ...do not convert to hex */
"RESOURCE %s NOW ACTIVE",nodename); /* ...message text */
/* Issue message... */
Cnmsmsg(&msgbuf, /* ...message to issue */
MSG, /* ...type is message */
OPER, /* ...issue to operator */
NULLCHAR); /* ...unused */
}
else /* IST093I not found, must be error*/
{ /* Copy message into varying length*/
/* character string... */
Cnmvlc(&msgbuf, /* ...varying length string */
0, /* ...do not convert to hex */
"ERROR - ACTIVATION UNSUCCESSFUL"); /* ...message text */
/* Issue message... */
Cnmsmsg(&msgbuf, /* ...message to issue */
MSG, /* ...type is message */
OPER, /* ...issue to operator */
NULLCHAR); /* ...unused */
}

Figure 58. Waiting and Trapping Using the PIPE Command (C) (Part 2 of 3)
else /* Nodename not specified */

Cnmvlc(msgbuf, /* Copy message into varying length*/
0, /* ...varying length string */
"ERROR - NODENAME NOT SPECIFIED" /* ...message text */); /* Issue the message... */

Cnmsmsg(&msgbuf, /* ...message to issue */
MSG, /* ...type is message */
OPER, /* ...issue to operator */
NULLCHAR /* ...unused */);

Figure 58. Waiting and Trapping Using the PIPE Command (C) (Part 3 of 3)

For more information about the PIPE command, refer to Tivoli NetView for z/OS
Customization: Using Pipes.

Retrieving Information

Figure 59 on page 150 shows how an HLL command processor or installation exit
routine can retrieve information from the NetView program. HLL command
processors and installation exit routines can access this information.

See “CNMINFC (CNMINFOC): Query NetView Character Information” on page
253 and “CNMINFI (CNMINFOI): Query NetView Integer Information” on page
256 for a list of the values supported.
/* Internal data definitions */
Dsivarch cdata, /* store data returned by Cnminfc */
            msgbuf; /* var len char strng for messages */
int idata; /* store data returned by Cnminfi */
/* Execution */
/* retrieve date and time... */
Cnminfc("DATETIME",
        &cdata, /* ...result goes here */
        18); /* ...at most eight bytes */
cdata.buffer??(cdata.size??) = '\0'; /* terminate data with null */
/* build message to display results */
Cnmvlc(&msgbuf, /* varying length string... */
         0, /* do not convert to hex... */
         "DATE/TIME IS: %s",cdata.buffer); /* ...message */
Cnmsmsg(&msgbuf, /* display results... */
        MSG, /* ...is a message */
        OPER, /* ...to invoking operator */
        NULLCHAR); /* ...not used */
/* retrieve the number of colors */
/* that the terminal supports... */
Cnminfi("COLORS ", /* ...result goes here */
        &idata); /* ...specify the variable */
/* build message to display results */
Cnmvlc(&msgbuf, /* varying length string... */
         0, /* do not convert to hex... */
         "NUMBER OF COLORS SUPPORTED ARE: %d",idata); /* ...message*/
Cnmsmsg(&msgbuf, /* display results... */
        MSG, /* ...message text */
        OPER, /* ...to the invoking operator */
        NULLCHAR); /* ...not used */

Figure 59. Retrieving Information (C)
Command List Variable Access

Figure 60 on page 153 shows the updating of common global variables. This example declares, initializes, and alters a variable, GVARIABLE. Task global variables are updated and read the same way. The only difference is the pool name TBLOBAL is specified instead of CGLOBAL.

Figure 61 on page 153 shows how to retrieve values from a stem variable after issuing a PIPE command with the STEM stage command.
**Figure 60. Command List Variable Access (C) (Part 1 of 2)**
C High-Level Language Services

Figure 60. Command List Variable Access (C) (Part 2 of 2)
/* External data definitions */
struct Vchar4 { /* Varying character 4 string */
    short int size;
    char buffer ??(5??);
};

struct Vchar15 { /* Varying character 15 string */
    short int size;
    char buffer ??(16??);
};

struct Vchar80 { /* Varying character 80 string */
    short int size;
    char buffer ??(81??);
};

Dsivarch pipecmd; /* Buffer to hold PIPE command */
/* Internal data definitions */
struct Vchar4 numvars; /* Number of variables in stem */
struct Vchar15 varname; /* Name of stem variable */
struct Vchar80 stemline; /* Store value of stem variable */

/* Execution */
/* Build a PIPE command. */
/* The < (From Disk) stage reads data into the pipeline from */
/* MYINFILE, a member of a dataset associated with the ddname */
/* DSIPARM. The records are treated as single-line messages. */
/* The NLOCATE stage discards comments lines which begin with */
/* either '*' or '/**'. */
/* The STEM stage writes the remaining messages from the pipeline */
/* to stemmed variables named PIPELINEx. */
Cnmvlc (&pipecmd,0,
    "PIPE < MYINFILE"
    " NLOCATE 1.1 &* 1.2 &*/&"
    " STEM PIPELINE");  /* Build PIPE command */
Cnmcmd(&pipecmd);  /* Issue PIPE command */

Figure 61. Command List Variable Access (C) (Part 1 of 2)
Using Locks

Figure 60 on page 152 shows the updating of common global variables, but it does not show you how to protect the updating of the variable named GVARIABLE by using a lock. The need for protecting the updating needs to be assessed on a case-by-case basis. Figure 62 on page 156 is a modified version of Figure 60 on page 152 to obtain a lock before attempting the update.

The lock name can be the same as the global variable, or it can be different.

If you decide that it is important to synchronize the updating of a variable, you can use Figure 62 on page 156, or you can run all the updates on a given task. Because only one process can occur on a task at a time, the updates are serialized. The task used can be any task, including the PPT.

C High-Level Language Services

Figure 61. Command List Variable Access (C) (Part 2 of 2)

For more information about the PIPE command, refer to Tivoli NetView for z/OS Customization: Using Pipes.
C High-Level Language Services

```c
Dsivarchar datain, /* store data returned by cnmvars */
    variable, /* store variable name for Cnmvars */
    msg; /* var len char str for messages */
int length = 24, /* max len of data from Cnmvars */
    x; /* used to increment value returned */
/* ...by Cnmvars */

/* Execution */

/* Initialize the variable */
Cnmvlc(&datain, /* ...varying length string */
    0, /* ...do not convert to hex */
    "Initial value"); /* ...initial value */
Cnmvlc(&variable, /* ...varying length string */
    0, /* ...do not convert to hex */
    "GVARIABLE"); /* ...variable name */

/* Obtain the lock to secure the accuracy of the update */
Cnmlk(LOCK, /* obtain the lock... */
    &variable, /* ...function is obtain lock */
    "", /* ...not used */
    WAIT); /* ...wait if not available */

if (Hlbptr->Hlbrc == CNM_GOOD)
{
    Cnmsmsg(Cnmvlc(&msg,0,"got lock"),MSG,OPER,NULLCHAR);
    Cnmvars(PUT, /* ...function is write */
        &datain, /* ...data is here */
        length, /* ...length not used */
        &variable, /* ...variable name */
        "CGLOBAL "); /* ...variable pool is cglobal */
}
```

Figure 62. Example of Using Locks (C) (Part 1 of 2)
C High-Level Language Services

Figure 62. Example of Using Locks (C) (Part 2 of 2)
Figure 63 shows how to code an HLL command processor to accept operator input in single-line mode. The interface is similar to the &PAUSE function of the NetView command list language. Input is requested by the application using the WAIT FOR OPINPUT command, and retrieved by the application using the CNMGETD service routine. The operator can respond using the GO command. See "GO Command" on page 213 for more information.

```c
/* Internal data definitions */
Dsivar msgbuf, /* var len char string for messages */
command, /* var len char string for cmds */
inbuf; /* store data returned by Cnmgetd */

/* Execution */
Cnmvlc(&msgbuf, /* length message string... */
0, /* ... do not convert to hex */
"Enter operator input data"); /* ... message */
Cnmsmsg(&msgbuf, /* ... text of message */
MSG, /* ... single line message */
OPER, /* ... to the invoking oper */
NULLCHAR); /* ... not used */
Cnmvlc(&command, /* length character string... */
0, /* ... do not convert to hex */
"WAIT 30 SECONDS FOR OPINPUT"); /*... command */
Cnmcmd(&command); /* 30 seconds... */
if (Hlbptr->Hlbrc == CNM_OPINPUT_ON_WAIT)
{
    Cnmgetd(GETMSG, /* ... function is get message */
    &inbuf, /* ... result goes here */
    255, /* ... max input length */
    Origblck, /* ... must provide work area */
    2, /* ... msg is on OPINPUT queue */
    1); /* ... get the first one */
    inbuf.buffer??(inbuf.size??) = '\0'; /* copy opinput into varying */
    Cnmvlc(&msgbuf, /* length character string... */
    0, /* ... do not convert to hex */
    "OPERATOR INPUT IS: %s", inbuf.buffer); /* ... message */
    Cnmsmsg(&msgbuf, /* ... text of message */
    MSG, /* ... single line message */
    OPER, /* ... to the invoking oper */
    NULLCHAR); /* ... not used */
}
```

Figure 63. Operator Input (C) (Part 1 of 2)
VIEW Command Processor

Figure 64 on page 160 is an example of using the full-screen VIEW command processor. The example creates and initializes the local variable called parm1. The VIEW command processor is invoked and displays a full-screen panel. For more information, refer to the Tivoli NetView for z/OS Customization Guide.
C High-Level Language Services

/* Internal data definitions */

Dsivarch variable, /* store variable name for Cnmvars */
    data, /* store returned data from Cnmvars*/
    command; /* varying len char strng for cmds */

/* Execution */

/* copy variable name into variable */
Cnmvlc(&variable, /* length character string... */
    0, /* ...do not convert to hex */
    "PARM1"); /* ...variable name */

Cnmvars(DCL, /* declare variable to local pool...*/
    &data, /* ...not used */
    48, /* ...not used */
    &variable, /* ...name is parm1 */
    LOCAL); /* ...the pool is local */

Cnmvlc(&data, /* varying length character strng... */
    0, /* ...do not convert to hex */
    "the contents of parm1 go here"); /* ...the data */

Cnmvars(PUT, /* initialize parm1... */
    &data, /* ...result goes here */
    48, /* ...max length is 48 bytes */
    &variable, /* ...name of variable is parm1 */
    LOCAL); /* ...the pool is local */

/* Issue the view command. Give the task name as a unique name */
/* to go on the View stack. */
Cnmvlc(&command, /* character string... */
    0, /* ...do not convert to hex */
    "VIEW %.8s TESTHLL INPUT",Origblck->Orig_task); /* cmd*/

Cnmcmd(&command); /* issue the command */

Figure 64. Using the Full-Screen View (C)

Figure 65 on page 161 shows the panel definition that is invoked by this example:
Message Processing

Figure 66 on page 163 lists the message attributes of a message. The invocation must be a result of an entry in the NetView automation table. This example functions for both single-line and multiline messages. Refer to the Tivoli NetView for z/OS Automation Guide for more information.
C High-Level Language Services

/*******************************************************************************/
/* Internal data definitions */
/*******************************************************************************/
Dsiorig getblock; /* orig block for Cnmgetd */
Dsivarch inbuf, /* returned data from Cnmgetd */
datain, /* returned data from Cnmgeta */
message, /* var len char string for messages */
temp; /* temp var len char string used... */
/* ...to build messages */
/* counter */
int i;
char *attr??(12??) = {
"AREAID ", /* array of message attributes... */
"DESC ", /* ...to be obtained by Cnmgeta */
"JOBNAME ",
"JOBNUM ",
"MCSFLAG ",
"MSGTYP ",
"REPLYID ",
"ROUTCDE", 
"SESSID ",
"SMSSID ",
"SYSCONID", 
"SYSID "};
/*******************************************************************************/
/**/ /* Execution */
/**/ /*
Cnmgetd(GETMSG, /* function is get a message */
&inbuf, /* result goes here */
255, /* max input length */
&getblock, /* must provide a work area */
IDATAQ, /* message on automation queue */
1); /* get the next line */

Figure 66. Message Processing (C) (Part 1 of 2)
Command Authorization Checking

This section describes an example of the command authorization checking capabilities provided by the NetView program. Define the operator, the operator’s authority, and the restrictions on invoking the command, keyword, and value.

The command gives the return code that the command authorization checking service routine returned to the operator.

This command checks for this syntax:
C High-Level Language Services

CSCOPCK

Where:

PARMx:
Specifies the keyword variable used to perform authorization checking.

VALx:
Indicates the value variable used to perform authorization checking.

Before using the example in Figure 67 on page 166, perform the following NetView set up, depending upon the method chosen for command authorization checking:

**Command Authorization Checking Using CMDAUTH=TABLE**

If you are using CMDAUTH=TABLE with TBLNAME=CMDTABLE in NetView domain CNM01 in network NETA, perform the following set up prior to using the sample code in Figure 67 on page 166.

**In DSIPARM(CMDTABLE)**
Restrict access to the command, keywords, and values as well as the operators and classes that can access them.

The command CSCOPCK can be executed by operators in GRP1 and GRP2. Operators in GRP1 can issue any keyword or keyword value, but operators in GRP2 cannot use the value of VAL1 with keyword PARM2 and cannot issue PARM3 at all.

```
PROTECT NETA.CNM01.CSCOPCK
PROTECT NETA.CNM01.CSCOPCK.PARM2
PROTECT NETA.CNM01.CSCOPCK.PARM2.VAL1
PROTECT NETA.CNM01.CSCOPCK.PARM3
PROTECT NETA.CNM01.CSCOPCK.PARM3.VAL1
GROUP GRP1 OPER1,OPER2,FRANK,BILL,JENNY
GROUP GRP2 OPER3,OPER4,FRED,LISA,BETH
GROUP GRP3 JOE
PERMIT GRP1 NETA.CNM01.CSCOPCK
PERMIT GRP1 NETA.CNM01.CSCOPCK.PARM2
PERMIT GRP1 NETA.CNM01.CSCOPCK.PARM2.VAL1
PERMIT GRP1 NETA.CNM01.CSCOPCK.PARM3
PERMIT GRP1 NETA.CNM01.CSCOPCK.PARM3.VAL1
PERMIT GRP2 NETA.CNM01.CSCOPCK
PERMIT GRP2 NETA.CNM01.CSCOPCK.PARM2
```

**Command Authorization Checking Using CMDAUTH=SAF**

If you are using CMDAUTH=SAF and OPERSEC=SAFDEF in NetView domain CNM01 in network NETA, perform the following set up prior to using the sample code in Figure 67 on page 166. RACF is used as the SAF product in this example.

**In RACF**
Restrict access to the command, keywords, and values as well as the operators and classes that can access them.

The command CSCOPCK can be executed by operators in GRP1 and GRP2. Operators in GRP1 can issue any keyword or keyword value, but operators in GRP2 cannot use the value of VAL1 with keyword PARM2 and cannot issue PARM3.
C High-Level Language Services

RDEFINE NETCMDS NETA.CNM01.CSCOPCK UACC(NONE)
RDEFINE NETCMDS NETA.CNM01.CSCOPCK.PARM2 UACC(NONE)
RDEFINE NETCMDS NETA.CNM01.CSCOPCK.PARM2.VAL1 UACC(NONE)
RDEFINE NETCMDS NETA.CNM01.CSCOPCK.PARM3 UACC(NONE)
RDEFINE NETCMDS NETA.CNM01.CSCOPCK.PARM3.VAL1 UACC(NONE)
ADDGROUP GRP1
ADDGROUP GRP2
ADDGROUP GRP3
CONNECT OPER1 GROUP(GRP1)
CONNECT OPER2 GROUP(GRP1)
CONNECT FRANK GROUP(GRP1)
CONNECT BILL GROUP(GRP1)
CONNECT JENNY GROUP(GRP1)
CONNECT OPER3 GROUP(GRP2)
CONNECT OPER4 GROUP(GRP2)
CONNECT FRED GROUP(GRP2)
CONNECT LISA GROUP(GRP2)
CONNECT BETH GROUP(GRP2)
CONNECT JOE GROUP(GRP3)
PERMIT NETA.CNM01.CSCOPCK CLASS(NETCMDS) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.CSCOPCK.PARM2 CLASS(NETCMDS) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.CSCOPCK.PARM2.VAL1 CLASS(NETCMDS) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.CSCOPCK.PARM3 CLASS(NETCMDS) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.CSCOPCK.PARM3.VAL1 CLASS(NETCMDS) ID(GRP1) ACCESS(READ)
PERMIT NETA.CNM01.CSCOPCK CLASS(NETCMDS) ID(GRP2) ACCESS(READ)
PERMIT NETA.CNM01.CSCOPCK.PARM2 CLASS(NETCMDS) ID(GRP2) ACCESS(READ)
/* Internal data definitions */

Dsivarch msgbuf; /* Buffer area for messages */
char *cn; /* Ptr to cmd that invoked us */
char *kw; /* Ptr to keyword of invocation */
char *kv; /* Ptr to keyvalue of invocation */
char *token; /* Ptr to keyvalue of invocation */
char cmdname[9] = " "; /* Command that invoked us */
char keyword[9] = " "; /* Keyword of invocation */
char keyvalue[9] = " "; /* Keyvalue of invocation */
int len; /* Length */

/* Execution */

/* Scan the keyword and the value */

/* Syntax of command is: */
/* CMDNAME KEYWORD(KEYVALUE) */
/* Parse the command buffer for: */

if (token != NULL)
{
    len = strlen(token); /* Get length of command name */
    strcpy(cmdname,token); /* Save command name */
    if (len < 8) /* Pad with blanks? */
        strncat(cmdname," ",8 - len);
}

if (token != NULL) /* enough parms? */
    Cnmscop(cmdname, keyword, keyvalue); /* Perform authorization checking */
else /* not enough parms specified */
    Hlbptr->Hlbrc = CNM_BAD_INVOCATION; /* set bad rc */

Figure 67. Command Authorization Checking (C) (Part 1 of 2)
C High-Level Language Services

Altering Data

The DSIEX02A installation exit routine, shown in Figure 68 on page 168, changes the echoed command message (MSGTYPE=*) to be more informative by giving the time that the command was entered.

Without the installation exit, if you enter the WHO command, the output is:

WHO

With the exit, the output is:

Command entered was: "WHO" at 12:00:00
C High-Level Language Services

/********************************************************************/
/* Descriptive Name: High Level Language C Dslex02a Example */
/ ********************************************************************/
/* Change Activity: */
/* date,author: description of changes */
/********************************************************************/
#pragma runopts (NOEXECOPS,NOSTAE,NOSPIE,ISASIZE(4K),ISAINC(4K))
/********************************************************************/
/* Standard include files*/
/********************************************************************/
#include <stdlib.h> /* Standard library */
#include <stdarg.h> /* Standard args */
/********************************************************************/
/* NetView high level language include files */
/********************************************************************/
#include "dsic.h" /* Include HLL macros */
/********************************************************************/
/* External data definitions */
/**************************************************************************/
Dsihlb *Hlbptr; /* Pointer to the HLB */
Dsiarch *Cmdbuf; /* Pointer to command buffer */
Dsiorig *Origblck; /* Pointer to Origin block */
main(int argc, char *argv??(??))
{
  /**************************************************************************/
  /* Internal data definitions */
  /**************************************************************************/
  Dsiorig getblock; /* Area for the Origin Block */
  Dsiarch datain; /* Old command text */
  Dsiarch time; /* Area for time */
  Dsiarch msgbuf; /* message buffer */
  /*********************************************************************************
  /* Convert operand pointers from character to hex addresses */
  /*********************************************************************************/
  sscanf(argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

Figure 68. Altering Data (C) (Part 1 of 2)
Figure 69 on page 170 shows how to display the character representation of the contents of the storage that the NetView program can access. For example, after locating the address of the main vector table using DISPMOD DSIMNTEX, you can display the first 4-bytes of the DSIMVT control block.

```c
/** Initialization */
/**

/**

/** Execution */
/**

/*! Retrieve the time... */
Cnminfc("TIME ", /* ...variable is time of day */
   &time, /* ...the result goes here */
   255); /* ...max length of 255 */
Cnmmsgd(PEEKLINE, /* ...subfunction is peek */
   &datain, /* ...result goes here */
   255, /* ...max length is 255 */
   &getblock, /* ...use new Origin block */
   IDATAQ, /* ...initial data queue (4) */
   1); /* ...check the first line */
datain.buffer??(datain.size??) = '\0'; /* put null at end of data */

if (getblock.Orig_msg_type == '*') {
   Cnmvite(&msgbuf,0,"Command entered was: %s at %.8s",datain.buffer,
          time.buffer);
   /* Replace the text... */
   Cnmaltd(REPLINE, /* ...subfunction is replace */
      &msgbuf, /* ...text of new message */
      &getblock, /* ...used peeked Origin block */
      IDATAQ, /* ...initial data queue */
      1); /* ...replace the first line */
}
Hlbptr->Hlbrc = CNM_GOOD; /* clear rc */
```

Figure 68. Altering Data (C) (Part 2 of 2)
C High-Level Language Services

/* Internal data definitions */
int numparms; /* Number of parms scanned */
int *xaddr; /* Hex value of srce_ptr */
int numbytes; /* Number of bytes to display */
char *inbufr_p; /* Buffer where data is copied */
char *srceptr; /* Address to copy from */
int i, x; /* Work counter */
Dsivar msgbuf; /* Buffer for Cnmsmsg */

/* Execution */
inbufr_p = inputbfr;
numbytes = 0;
numparms = sscanf((char *) &(Cmdbuf->buffer), /* parse cmd buffer */
                   "%s%s%sx", /* Format string */
                   &xaddr, /* The address to display */
                   &numbytes); /* For this number of bytes */
if (numparms != 2) /* Address and length given ? */
{
    Cnmvlc(&msgbuf,0,"Invalid number of operands");
    Cnmsmsg(&msgbuf, /* No, display error message... */
            MSG, /* ...message */
            OPER, /* ...to the operator */
            NULLCHAR); /* ...not used */
    Hlbptr->Hlbrc = 1; /* set return code */
    return; /* Terminate processing */
}
if (numbytes <= 0) /* A valid length given? */
{
    Cnmvlc(&msgbuf,0,"Invalid length given");
    Cnmsmsg(&msgbuf, /* No, display error message... */
            MSG, /* ...message */
            OPER, /* ...to the operator */
            NULLCHAR); /* ...not used */
    Hlbptr->Hlbrc = 1; /* set return code */
    return; /* Terminate processing */
}
if (numbytes >= 4096) /* A valid length given? */
{
    Cnmvlc(&msgbuf,0,"Invalid length given, must be less than FFF");
    Cnmsmsg(&msgbuf, /* No, display error message... */
            MSG, /* ...message */
            OPER, /* ...to the operator */
            NULLCHAR); /* ...not used */
    Hlbptr->Hlbrc = 1; /* set return code */
    return; /* Terminate processing */
}

Figure 69. Displaying Contents of Storage (C) (Part 1 of 2)
Data Set Access

Figure 70 on page 172 shows the opening (using CNMMEMO), reading (using CNMMEMR), and closing (using CNMMEMC) of NetView partitioned data sets. This example reads the CNMSTYLE member of DSIPARM and displays the contents to the operator.
C High-Level Language Services

```c
int token; /* Token used to match open to */
/* ...read and close */
Dsvarch msgbuf; /* Line that is read */

Dsivarch msgbuf; /* Line that is read */

/**********************************************************************/
/* Open the member */
/**************************************************************************/

Cnmmemo(&token, /* ...token returned by Cnmmemo */
"DSIPARM ", /* ...ddname of PDS */
"CNMSTYLE"); /* ...member name of PDS */
if (Hlbptr->Hlbrc != CNM_GOOD)
{
    Cnmvlc(&msgbuf, /* ...do not convert to hex */
            "OPEN FOR DATASET FAILED RC= %d", Hlbptr->Hlbrc); /* msg */
    /* Send message... */
    Cnmsmsg(&msgbuf, /* ...member in DDNAME not found */
            MSG, /* ...single line message */
            OPER, /* ...to the operator */
            NULLCHAR); /* ...taskname ignored */
}
else
{
    /**********************************************************************/
    /* Read the member */
    /**********************************************************************/

    Cnmmemr(token, /* ...provide token from OPEN */
             &msgbuf, /* ...result goes here */
             80); /* ...read 80 bytes */

    while (Hlbptr->Hlbrc == CNM_GOOD) /* Read until EOF */
    {
        msgbuf.size = 72; /* only write 72 bytes of record */
        Cnmsmsg(&msgbuf, /* ...write first 72 bytes */
                MSG, /* ...single line message */
                OPER, /* ...to the operator */
                NULLCHAR); /* ...taskname ignored */
        Cnmmemr(token, /* ...provide token from OPEN */
                 &msgbuf, /* ...result goes here */
                 80); /* ...read 80 bytes */
    }

    /**********************************************************************/
    /* Close the member */
    /**********************************************************************/

    Cnmmemc(token); /* ...provide token from OPEN */
}
```

Figure 70. Data Set Access (C)
Communicating with Devices

The NetView program provides the Cnmcnmi service routine for use in communicating with devices in the network through the CNMI. Any returned data can be accessed using the Cnmgetd service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

Figure 71 on page 174 uses the Cnmcnmi service routine to send a product set ID data request to a specified PU. Any data returned is sent as a message to the operator.

The command syntax is:

```
CCNMI
```

Where:

- **ALL**
  - Specifies that vital product data is to be retrieved for the PU and its attached ports.

- **OWN**
  - Specifies that vital product data is to be retrieved for the PU only. This keyword is the default.

**puname**

- Is the name of the PU from which the vital product data is retrieved. This parameter is required.
C High-Level Language Services

/***********************************************************/
/* External data definitions */
/***********************************************************/
typedef struct
{
  short size;
  char buffer[1025];
} Bigvlc;

main(int argc, char *argv)
{
/***********************************************************/
/* Internal data definitions */
/***********************************************************/
  int rcode; /* Return code */
  int count; /* Count of scanned args */
  Dsivar charname; /* puname varying length */
  Dsivar msgbuf; /* Message buffer */
  Dsior orig getblock; /* Area for the work orig block */
  Bigvlc datain; /* Buffer for the RU */
  char own or all[4]; /* Own or all placeholder */
  Dsivar fwdru; /* Forward RU */
  Dsivar ru; /* RU data */
  Dsivar own; /* 01 if own specified */
  Dsivar all; /* 83 if all specified */
  Dsivar puhdr; /* puname header */
  Dsivar endofru; /* end of RU */
  char *ptr; /* ptr used to build fwdru */

Figure 71. CNMI (C) (Part 1 of 5)
Vital Product Data RU definitions

From the VTAM Programming Manual, a forward RU is defined below

<table>
<thead>
<tr>
<th>Byte</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81</td>
<td>Network services, logical services</td>
</tr>
<tr>
<td>1</td>
<td>08</td>
<td>Management services</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>Request code</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
<td>Format 0</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>Ignore target names,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Solicit a reply, and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No CNM header contained</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>Reserved</td>
</tr>
<tr>
<td>6-7</td>
<td>000E</td>
<td>Length of NS RU</td>
</tr>
<tr>
<td>8-15</td>
<td></td>
<td>NS RU -- NMVT -- documented in SNA Ref Sum</td>
</tr>
<tr>
<td>8-A</td>
<td>4103BD</td>
<td>NS Header for NMVT</td>
</tr>
<tr>
<td>8-C</td>
<td>0000</td>
<td>Retired</td>
</tr>
<tr>
<td>D-E</td>
<td>0111</td>
<td>PRID</td>
</tr>
<tr>
<td>F</td>
<td>00</td>
<td>unsolicited NMVT,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>only NMVT for this PRID</td>
</tr>
<tr>
<td>10-16</td>
<td></td>
<td>One MS major vector</td>
</tr>
<tr>
<td>10-11</td>
<td>0006</td>
<td>Length field of PSID (Product Set ID) vector</td>
</tr>
<tr>
<td>12-13</td>
<td>8090</td>
<td>Code point for PSID</td>
</tr>
<tr>
<td>14-15</td>
<td></td>
<td>Length of subvector</td>
</tr>
<tr>
<td>14</td>
<td>02</td>
<td>Length of subvector</td>
</tr>
<tr>
<td>15</td>
<td>81</td>
<td>Request information on control unit only</td>
</tr>
<tr>
<td>15</td>
<td>83</td>
<td>Request information on control unit and its attached devices</td>
</tr>
<tr>
<td>16</td>
<td>F1</td>
<td>From VTAM programming, PU</td>
</tr>
<tr>
<td>17</td>
<td>08</td>
<td>Length of PU name</td>
</tr>
<tr>
<td>18</td>
<td>PUNAME</td>
<td>Eight byte PUNAME, left justified</td>
</tr>
<tr>
<td>20</td>
<td>00</td>
<td>End of RU</td>
</tr>
</tbody>
</table>

Initialization

Cnmvlc(&ru,1,"81081000000000004103BD0000000111000006809002");
Cnmvlc(&own,1,"81");
Cnmvlc(&all,1,"83");
Cnmvlc(&puhdr,1,"F108");
Cnmvlc(&endofru,1,"00");
rcode = 0;

Figure 71. CNMI (C) (Part 2 of 5)
ptr = (char*) &fwdr u.buffer;
count = sscanf((char*) &(Cmdbuf->buffer ),"%*s%s%s",       
               puname.buffer,ownorall);

if (puname.size < 8) /* Pad with blanks if needed */
   strncat(puname.buffer," ",8 - puname.size);
if ((count == 1) || (strncmp(ownorall,"OWN",3) == 0))
{
    memmove(ptr,ru.buffer,ru.size);
    ptr=ptr+ru.size;
    memmove(ptr,own.buffer,own.size);
    ptr=ptr+own.size;
    memmove(ptr,puhdr.buffer,puhdr.size);
    ptr=ptr+puhdr.size;
    memmove(ptr,puname.buffer,puname.size);
    ptr=ptr+puhdr.size;
    memmove(ptr,endofru.buffer,endofru.size);
    fwdru.size = ru.size+own.size+puhdr.size+puname.size+endofru.size;
}
else if (strncmp(ownorall,"ALL",3) == 0) /* ALL specified */
{
    memmove(ptr,ru.buffer,ru.size);
    ptr=ptr+ru.size;
    memmove(ptr,all.buffer,all.size);
    ptr=ptr+all.size;
    memmove(ptr,puhdr.buffer,puhdr.size);
    ptr=ptr+puhdr.size;
    memmove(ptr,puname.buffer,puname.size);
    ptr=ptr+puhdr.size;
    memmove(ptr,endofru.buffer,endofru.size);
    fwdru.size = ru.size+all.size+puhdr.size+puname.size+endofru.size;
}
else /* Else invalid parm inform user */
{
    Cnmvlc(&msgbuf,0,"Invalid command syntax");
    Cnmsmsg(&msgbuf,MSG,TASK,Origblk->Orig_task);
    rcode = 8;
}

Figure 71. CNMI (C) (Part 3 of 5)
if (rcode == 0) { /* Good so far? */
    /* Send RU over the CNMI... */
    Cnmcnmi(SENRPLY, /* ...expect a reply */
        &fwdru, /* ...RU built above */
        puname.buffer, /* ...to the PU name specified */
        180); /* ...timeout after 3 minutes */
    if (Hlbptr->Hlbrc == CNM_GOOD) /* Everything ok? */
    { /* Yes, continue */
        /* Read in the first RU returned */
        Cnmgetd(GETLINE, /* ...a single RU */
            &datain, /* ...inti here */
            1024, /* ...truncate after 1024 bytes */
            &getblock, /* ...provide a new origin block */
            CNMIQ, /* ...on the CNMI queue (5) */
            1); /* ...the first RU */
        while (Hlbptr->Hlbrc == 0) /* End of queue reached? */
        { /* Send info to the operator... */
            Cnmsmsg(&datain, /* ...from here */
                MSG, /* ...issue message */
                TASK, /* ...to the task */
                Origblk->Orig_task); /* ...that originated request */
            /* Read in the next RU returned */
            Cnmgetd(GETLINE, /* ...a single RU */
                &datain, /* ...inti here */
                1024, /* ...truncate after 1024 bytes */
                &getblock, /* ...provide a new origin block */
                CNMIQ, /* ...on the CNMI queue (5) */
                1); /* ...the first RU */
        }
    }
}

Figure 71. CNMI (C) (Part 4 of 5)
Performing I/O on a VSAM File (Keyed File Access)

The following code for a NetView HLL command processor allows input or output (I/O) to a VSAM file through the Cnmkio service routine.

The command processor must execute on a DST. Use either the Cnmsmsg service routine (with a type of COMMAND), or the EXCMD command.

Figure 72 on page 179 creates a database with five records and these keys and data:

<table>
<thead>
<tr>
<th>Key</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>A</td>
</tr>
<tr>
<td>02</td>
<td>B</td>
</tr>
<tr>
<td>03</td>
<td>C</td>
</tr>
<tr>
<td>04</td>
<td>D</td>
</tr>
<tr>
<td>05</td>
<td>E</td>
</tr>
</tbody>
</table>
C High-Level Language Services

/* Internal data definitions */
Dsivar rec; /* store output data for Cnmkio */
Dsivar inrec; /* store data returned by Cnmkio */
Dsivar key; /* store key for Cnmkio */
Dsivar msg; /* store messages to be displayed */
char outdata??(6??) = "ABCDE"; /* output data */
char keydata??(11??) = "0102030405"; /* data for building keys */
char *keyptr; /* pointer to key data */
char *outptr; /* pointer to output data */
int i; /* counter */

/* Execution */

/* WRITE OUT 5 RECORDS... */
/* PUT DIRECT must be used for new records, and PUT UPDATE */
/* must be used for old records. Therefore, we will use GET */
/* Equal to determine if the record is new or not. If new, */
/* then a PUT DIRECT will follow...if not, then a put update */
/* follows. */

Figure 72. VSAM Keyed File Access (C) (Part 1 of 3)
Figure 72. VSAM Keyed File Access (C) (Part 2 of 3)
Coding a DST Installation Exit

Figure 73 on page 182 shows coding a NetView HLL installation exit routine that primes an empty VSAM database for a data services task (DST). If a VSAM database is not primed (has at least one record), subsequent input and output (I/O) requests fail.
C High-Level Language Services

C High-Level Language Services

Figure 73. Coding a DST Installation Exit (C) (Part 1 of 2)
C High-Level Language Services

Figure 74 on page 184 shows coding an installation exit routine, DSIEX03, that sets a task global variable equal to the last time a command was entered on the system. If the last command was the CSNDDAT command, the task global variable is not set. The CSNDDAT command (see “Sending Data” on page 188) receives the variable value.

/* Internal data definitions */

/**********************************************************/
/* Convert operand pointers from character to hex addresses */
/**********************************************************/
sscanf(argv??(1??),"%x",&Hlbptr);
sscanf(argv??(2??),"%x",&Cmdbuf);
sscanf(argv??(3??),"%x",&Origblk);
/**********************************************************/
/* Initialization */
/**********************************************************/

/**********************************************************/
/* Execution */
/**********************************************************/
Cnmvlc(Cmdbuf, /* Set key... */
 1, /* ...convert to hex */
"0000"); /* ...hex zeroes */
memmove((&(Cmdbuf->buffer??(0??))+2), /* Set rest of key... */
"low rec",7); /* ...move in 7 bytes */
Cmdbuf->size = 9; /* set new Cmdbuf size */
Hlbptr->Hlbrc = USERSWAP; /* Set USERSWAP rc */
}

Figure 73. Coding a DST Installation Exit (C) (Part 2 of 2)
C High-Level Language Services

#include <stdlib.h> /* Standard library */
#include <stdarg.h> /* Standard args */

#include "dsic.h" /* Include HLL macros */

Dsihlb *Hlbptr; /* Pointer to the HLB */
Dsivarch *Cmdbuf; /* Pointer to command buffer */
Dsiorig *Origblk; /* Pointer to Origin block */

Dsivarch time; /* Time last command entered */
Dsivarch cvname; /* Name of variable for Cnmvars */

main(int argc, char *argv)
{
}

Figure 74. Coding Installation Exit (C) (Part 1 of 2)
Coding the WAIT FOR DATA Function

CWATDAT and CSNDDAT are standard command processors, which can be invoked from an installation exit. The following sections describes sending messages with a type of request, waiting on the response, and parsing the results. The code in this example is the CWATDAT command.

The command syntax:

CWATDAT

```
CWATDAT taskname
```

Where:

**taskname**

Specifies the task global variable set by the installation exit and retrieved by the CSNDDAT command.
The flow of the WAIT for data function is shown in Figure 75.

Requesting Data

- The requesting OST invokes CWATDAT and specifies the target OST to which to send the request.
- CWATDAT uses CNMSMSG to send the request to the target OST.
- The requesting OST issues a WAIT FOR DATA.
- The WAIT FOR DATA is satisfied and a message is issued.

Sending Data

- The CSNDDAT command is invoked on the target OST and finds the task global variable set by DSIEX03.
- CNMSMSG is invoked to send the value retrieved. The type used in the CNMSMSG is DATA.

Figure 75. Flow of the WAIT FOR DATA Function

Requesting Data

Figure 76 on page 187 describes the initiating command processor that sends a message containing a request, waits on the response, and parses the results.

It also shows how to find the last time that a command was entered on the given OST. A task global variable, LAST_COMMAND_TIME, is set by DSIEX03 (see “Coding an Installation Exit” on page 183), and this value is retrieved by the CSNDDAT command that is invoked on the target task (see “Sending Data” on page 188).
Figure 76. Requesting Data (C) (Part 1 of 2)
Figure 77 on page 189 is to find the last time that a command was entered on the
given task. A task global variable, LAST_COMMAND_TIME, is set by DSIEX03 (see "Coding an Installation Exit"
on page 183). This value is retrieved by the CSNDDAT command that is invoked by the CWATDAT command (see "Coding
the WAIT FOR DATA Function" on page 185) on the target task. This command
processor is executed when the CSNDDAT command is invoked.

Sending Data

Figure 76. Requesting Data (C) (Part 2 of 2)
C High-Level Language Services

Figure 77. Sending Data (C) (Part 1 of 2)
Automating Management Services Units (MSUs)

Figure 78 on page 191 is an example of how to send an MSU directly to the automation table for evaluation.
Translating Code Points

Figure 79 on page 192 is an example of how to translate a numeric code point value into its corresponding EBCDIC text.
Registering Applications with MS Transport and Operations Management

Figure 80 is an example of how to register an application with both the MS transport and the operations management application.

```c
/* INTERNAL DATA DEFINITIONS */

int code;
Dsivarch trdata; /* A Character Varying for Cnmc2t */

/* CONVERT PARAMETER POINTERS FROM CHARACTER TO HEX ADDRESSES */
sscanf(argv[1], "%x", &Hlbptr);
sscanf(argv[2], "%x", &Cmdbuf);
sscanf(argv[3], "%x", &Origblck);
Cnmscan(&Cmdbuf, "%*S%D", code);
Cnmc2t(&trdata, 256, "SNAALERT", code);
Hlbptr->Hlbrc = CNM_GOOD;
```

Figure 80. How to Register an Application (C)
Registering Applications with the High Performance Transport

Figure 81 shows how to register an application with the high performance transport.

Sending an Alert Over the MS Transport

Figure 82 on page 194 is an example of how to send a software alert over the MS transport.
C High-Level Language Services

Figure 82. Sending an Alert over the High Performance Transport (C) (Part 1 of 2)
Figure 83 on page 196 is an example of how to send a software alert over the high performance transport.

/* if the conversion fails, send a message to the operator */

} else {
    Cnmvlc(&msgbuf,NOHEXCNV,"Cnmvlc failure, rc = %i",
          Hlbptr->Hlbrc);
    Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
}

Hlbptr->Hlbrc = CNM_GOOD; /* set a good rc and exit */
C High-Level Language Services

/* INTERNAL DATA DEFINITIONS */
Dsivarch msgbuf,
    alert,
supplied_correlator, /* if we want to build MV x'1549'*/
correlator_area; /* if we want NetView to build it*/
/* CONVERT PARAMETER POINTERS FROM CHARACTER TO HEX ADDRESSES */
sscanf(argv[1],"%x",&Hlbptr);
sscanf(argv[2],"%x",&Cmdbuf);
sscanf(argv[3],"%x",&Origblk);
/* First, convert the hex data into a varying length char string. */
/* The alert is in the following format: */
/* MV 0000 Alert MV */
/* SV 92 - Generic Alert */
/* SV 10 - Product ID (12345) */
/* SV 03 - Hierarchy (NAME1,TYP1) */
/* SV 93 - Probable Causes */
/* SV 96 - Recommended Actions */
Cnmvlc(&alert,CNVTOHEX,"00461212004200000B92000001210100000001";
    10100000110E0A0804F1F7F3F4F5A0401
110303010951C45F1A004040E3B7D1F1
0693100110230C9606011022102304B13110");
/* if the conversion is successful, call Cnmhsmu to send the alert*/

if (Hlbptr->Hlbrc==CNM_GOOD) {
    supplied_correlator.size=0; /* null supplied correlator */
    Cnmhsmu(NONMDSMU, /* let NetView build x'1311' */
        &alert, /* MV x'1212' */
        &supplied_correlator, /* this was null-ed out above */
        &correlator_area, /* let NetView build x'1549' */
        0, /* default time-out value */
        "NO ", /* not a synchronous send */
        " ", /* REPLYCMD */
        "USERAPPL", /* Assumes USERAPPL is registered*/
        "NETA ", /* netid=NETA */
        "CNM01 ", /* destination nau=CNM01 */
        "DESTAPPL", /* sending to user-defined appl */
        REQUEST_WITHOUT_REPLY, /*request without reply expect ed*/
        "PRI_LOW "); /* priority for incoming request */
/* if the send fails, send a message to the operator */
if (Hlbptr->Hlbrc != CNM_GOOD) {
    Cnmvlc(&msgbuf,NOHEXCNV,"Cnmhsmu failure, rc = %i",
        Hlbptr->Hlbrc);
    Cnmmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
}

Figure 83. Sending Data Over the High Performance Transport (C) (Part 1 of 2)
Sending an MDB to NetView for Processing

Figure 83 shows how to send an MDB to NetView for processing.

#include <stdio.h> /* Standard I/O header */
#include <string.h> /* String functions */
#include <stdefs.h> /* Standard Definitions */
#include <stdlib.h> /* Standard Library */
#include <stdarg.h> /* Standard Args */

/* Include the High-Level Language include files */
#include "dsic.h" /* Include HLL macros */

/* Parameters */
Dsihlb *Hlbptr; /* Pointer to the HLB */
Dsivarch *Cmdbuf; /* Pointer to command buffer */
Dsiorig *Origblck; /* Area for the Origin Block */

/* Internal data structures */
typedef struct {
    short int size; /* Length of buffer */
    char buffer??(300??); /* Varying length buffer */
} Myvarch;

/* Internal data definitions */
Dsivarch msgbuf,
    so, /*soptr;

Myvarch mdb,
    *mdbptr;
char mycorr??(16??);

C High-Level Language Services

Figure 84 shows how to send an MDB to NetView for processing.
main(int argc, char *argv[]) {
    /*****************************************************************
    /* Convert parameter pointers from character to hex addresses */
    /*****************************************************************
    sscanf(argv[1], "%x", &Hlbptr);
    sscanf(argv[2], "%x", &Cmdbuf);
    sscanf(argv[3], "%x", &Origblk);

    /*****************************************************************
    /* Create the MDB */
    /* */
    /* The CNMVLCL service provides 2 functions in this call: */
    /* */
    /* 1. Converts the MDB data to hexadecimal. */
    /* */
    /* 2. Calculates and appends the MDB length to the */
    /*     beginning of the MDB. */
    /* */
    /*****************************************************************
    Cnmvlc(&mdb, CNVTOHEX, "0001" /* MDB Type 1 */
           "D4C7D640" /* MDB Acronym */
           "00000001" /* Revision code */
           "0038" /* General Object length */
           "0001" /* General Object type */
           "00000000" /* MDBGMID (message id) */
           "C8C84BD4D44BE2E2" /* Time stamp HH.MM.SS */
           "4BE3C8" /* Time stamp .TH */
           "00" /* Reserved */
           "EBE8EBEBAC4C4C4" /* Date stamp YYYYDDD */
           "00" /* Reserved */
           "0000" /* MDBGGMFLG (message flags) */
           "0000" /* Reserved */
           "00000000" /* Foreground presentation attr */
           "00000000" /* Background presentation attr */
           "D6D9C9C7E2D5C1D4" /* Originating system name */
           "D1D6C240D5C1D4C5" /* Job name */
           "0082" /* Control program object length */
           "0002" /* Control program object type */
           "00000001" /* Object version level */
           "D4E5E240" /* Control program name */
           "C6D4C9C4C6D4C9C4" /* FMID of originating system */
}

Figure 84. Sending an MDB to NetView for Processing (C) (Part 2 of 4)
C High-Level Language Services

"00000000000000000000000000000000" /* 128 Routing codes */
"0000" /* Descriptor codes */
"0000" /* Message level */
"0000" /* Message attribute flags */
"0000" /* Message priority */
"0000" /* Reserved */
"0012" /* ASID of issuer */
"00000000" /* JOB Step TCB for issuer */
"00000000" /* Token for DOM */
"00" /* System id for DOM */
"00" /* DOM flags */
"00" /* Misc routing information */
"00" /* Reserved */
"0000000000000000" /* Originating Job ID */
"0000000000000000" /* Retrieval KEY */
"0000000000000000" /* Automation Token */
"0000000000000000" /* Command and Response Token */
"00000000" /* Console ID */
"0000" /* Message type flags */
"0077" /* Reply ID length */
"0000000000000000" /* Reply ID (EBCDIC) */
"0000" /* Reserved */
"0000" /* Offset to beginning of msg */
"00000000" /* Reply ID (binary) */
"00" /* Areaid */
"00" /* Reserved */
"00000001" /* Number of lines in message */
"0000000000000000" /* Originating Job name */
"0049" /* Text object length */
"0004" /* Text object type */
"78E3C8C9E240" /* Text attributes */
"D4C5E2E2C1C7C540" /* Text of message: "THIS " */
"E6C1E240E2C5D5E340" /* "MESSAGE " */
"C6D9D6D440C340" /* "WAS SENT " */
"C3D5D4D7D4C4C240" /* "FROM C " */
"E2C5D9E5C9C3C5" /* "SERVICE" */
"0000"); /* 2 extra bytes so that length */
/* calculated includes length */
/* of the length field */

/**********************************************************************
/* Create the Source Object */
/**********************************************************************/
Cnmvlc(&so,CNVTOHEX,"0005" /* Source object type */
"07D5D4E8E2E8E2" /* Nickname "MYSYS" */
"0000"); /* 2 extra bytes so that length */
/* calculated includes length */
/* of the length field */

/**********************************************************************
/* Initialize the correlator */
***********************************************************************/
*mycorr = *mycorr ?' *mycorr; /* Exclusive OR */

Figure 84. Sending an MDB to NetView for Processing (C) (Part 3 of 4)
/* Set up pointers to MDB and Source Object */
mdbptr = &mdb;
soptr = &so;

/* Call CNMPMDB */
Cnmpmdb(&mdbptr,&soptr,mycorr);

/* Respond to the operator with the return code from CNMPMDB */
Cnmvlc(&msgbuf,0,"RETURN CODE FROM CNMPMDB: %i",Hlbptr->Hlbrc);
Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);

Hlbptr->Hlbrc = CNM_GOOD;       /* Successful completion */
}

Figure 84. Sending an MDB to NetView for Processing (C) (Part 4 of 4)
### Chapter 10. Testing and Debugging

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using RID to Monitor a Task</td>
<td>203</td>
</tr>
<tr>
<td>RID Command</td>
<td>203</td>
</tr>
<tr>
<td>RID Scenario</td>
<td>205</td>
</tr>
</tbody>
</table>

### Chapter 11. Service Reference

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite Return Codes</td>
<td>209</td>
</tr>
<tr>
<td>Command Reference</td>
<td>212</td>
</tr>
<tr>
<td>GLOBALV Command</td>
<td>212</td>
</tr>
<tr>
<td>GO Command</td>
<td>213</td>
</tr>
<tr>
<td>PIPE Command</td>
<td>214</td>
</tr>
<tr>
<td>QUEUE Command</td>
<td>214</td>
</tr>
<tr>
<td>RESET Command</td>
<td>215</td>
</tr>
<tr>
<td>TRAP Command</td>
<td>218</td>
</tr>
<tr>
<td>WAIT Command</td>
<td>218</td>
</tr>
<tr>
<td>HLL Service Routine Reference</td>
<td>219</td>
</tr>
<tr>
<td>CNMALTD (CNMALTDATA): Alter Data on a Queue</td>
<td>219</td>
</tr>
<tr>
<td>CNMAUTO (CNMAUTOTAB): Invoke Automation Table</td>
<td>221</td>
</tr>
<tr>
<td>CNMCELL (CNMSTRCELL): Storage Cell</td>
<td>222</td>
</tr>
<tr>
<td>CNMCMD (CNMCOMMAND): Invoke NetView Commands</td>
<td>223</td>
</tr>
<tr>
<td>CNMCNMI (CNMI): CNMI Access Under a DST</td>
<td>224</td>
</tr>
<tr>
<td>CNMCPSYS (CNMCOPYSTR): Copy Storage</td>
<td>226</td>
</tr>
<tr>
<td>CNMC2T (CNMCODE2TXT): Code Point Translation</td>
<td>228</td>
</tr>
<tr>
<td>CNMGETA (CNMGETATTR): Query Message Attributes</td>
<td>229</td>
</tr>
<tr>
<td>CNMGETD (CNMGETDATA): Data Queue Manipulation</td>
<td>240</td>
</tr>
<tr>
<td>CNMHREG (CNMHREGIST): High Performance Transport Application Registration</td>
<td>244</td>
</tr>
<tr>
<td>CNMHSMU (CNMHSENDMU): Send High Performance Message Unit</td>
<td>247</td>
</tr>
<tr>
<td>CNMINFC (CNMINFOC): Query NetView Character Information</td>
<td>253</td>
</tr>
<tr>
<td>CNMINFOI (CNMINFOI): Query NetView Integer Information</td>
<td>256</td>
</tr>
<tr>
<td>CNMKIO (CNMKKEYIO): Keyed File Access Under a DST</td>
<td>259</td>
</tr>
<tr>
<td>CNMLK (CNMLOCK): Controlling a Lock</td>
<td>261</td>
</tr>
<tr>
<td>CNMEMC (CNMENCOSMEM): Close NetView Partitioned Data Set</td>
<td>263</td>
</tr>
<tr>
<td>CNMEMEMO (CNMEMOPENMEM): Open NetView Partitioned Data Set</td>
<td>263</td>
</tr>
<tr>
<td>CNMEMEMR (CNMEMREADMEM): Read NetView Partitioned Data Set</td>
<td>265</td>
</tr>
<tr>
<td>CNMNAMS (CNMNAMESTR): Named Storage Data Block</td>
<td>266</td>
</tr>
<tr>
<td>CNMPMDB (CNMPRSMDB): Process Message Data Block</td>
<td>268</td>
</tr>
<tr>
<td>CNMPOOL (CNMSTRPOOL): Storage Pool</td>
<td>269</td>
</tr>
<tr>
<td>CNMQAPI (CNMQAPI): Resource Object Data Manager</td>
<td>271</td>
</tr>
<tr>
<td>CNMRGS (CNMRREGIST): Application Registration</td>
<td>273</td>
</tr>
<tr>
<td>CNMSCAN (CNMSSCAN): Parse or Convert Character String-PL/I Only</td>
<td>277</td>
</tr>
<tr>
<td>CNMSCOP (CNMSCOPECK): Command, Keyword, and Value Authorization Checking</td>
<td>280</td>
</tr>
<tr>
<td>CNMSMSG (CNMSSENDMSG): Send Message or Command</td>
<td>282</td>
</tr>
<tr>
<td>CNMSMU (CNMSSENDMU): Send Message Unit</td>
<td>285</td>
</tr>
<tr>
<td>CNMSUBS (CNMSUBSYM): Substitute System Symbolics</td>
<td>292</td>
</tr>
<tr>
<td>CNMVARQ (CNMVARPOOL): Set or Retrieve Variables</td>
<td>293</td>
</tr>
</tbody>
</table>
Chapter 10. Testing and Debugging

Before reading this chapter, you should have successfully compiled and link-edited your HLL module.

This chapter describes the NetView remote interactive debugger (RID) and how you can use it to do the following tasks:
- Debug HLL programs
- Validate entry and exit parameters to HLL service routines
- Display storage at various predetermined debug points in the code.

Using RID to Monitor a Task

The NetView interactive debugging facility enables you to monitor and debug HLL modules during execution. You must determine a monitoring task and a target task to debug HLL modules effectively. The monitoring task must be an OST. From this task, you issue the RID commands that control the execution of the HLL module running under the target task. The target task can be an OST, a PPT, an NNT, or a DST.

RID begins to monitor the target task immediately after the RID command is issued from the monitoring task. If RID is invoked in STEP mode, the monitoring task controls the execution of the HLL module running under the target task. The monitoring task continues to control the execution of HLL modules running under the target task until RID is invoked with the RUN or END option.

The most common use of the debugger is the default option, which displays operands upon entry to (HAPIENTR) and exit from (HAPIEXIT) HLL service routines.

RID Command

This syntax shows the RID command as it is issued from the monitoring or debugging task.

RID

Where:
CONTINUE
The CONTINUE option is used to resume execution of a task that was stopped by the STEP option of RID. You can specify new debug point match criteria in conjunction with the CONTINUE option. The CONTINUE keyword is provided for readability only. You can resume execution by reissuing the RID command with its original operands.
Testing and Debugging

END
The END option causes RID debugging of a task to cease and allows other operators to invoke RID for the target task. If the target task is stopped and RID is invoked with the END option, the HLL program running under the target task is resumed.

MODNAME
Is the name of the module being monitored by RID. If you specify *, RID monitors all HLL programs running under the target task. * is the default.

OPTION
Specifies the type of debug point.
* All debug points are displayed.

HAPIENTR
Entry to an HLL API service routine.

HAPIEXIT
Exit from an HLL API service routine.

RUN
The RUN option is similar to the STEP option, except that the target task continues to execute after issuing the messages at the debug points. The RUN option resumes execution of a task stopped in the STEP mode.

STEP
Specifies that the target task is stopped whenever control is given to a debug point that matches the criteria indicated by the MODNAME or OPTION operand. Messages providing data captured at the debug point are displayed at the operator station that invoked RID to monitor the target task. STEP is the default.

TASK
Is the name of the task being monitored by RID. The target task can be an OST, PPT, NNT, or DST. PPT can be used as a synonym for the PPT task (xxxxxPPT) where xxxx is the domain ID in the local NetView program.

Usage Notes:
1. Only one NetView operator at a time can monitor a NetView task using RID.
2. Do not use RID to monitor or debug HLL command processors running under the PPT. Running RID against the PPT suspends the PPT, because undesirable results can occur if timer-sensitive functions (such as AT or EVERY) are being performed.
3. The default value (*) for MODNAME and OPTION is used until you specify a value. Once you specify a value for MODNAME, that value is used on all successive RID invocations unless explicitly overridden. MODNAME and OPTION are reset to the default values once RID is invoked with the END option.

Return Codes

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything OK</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>Task not found</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_OPTION</td>
<td>128</td>
<td>Incorrect option</td>
</tr>
<tr>
<td>CNM_BAD_TASKNAME</td>
<td>164</td>
<td>Task name too long</td>
</tr>
</tbody>
</table>
**RID Scenario**

This section describes how to use RID to monitor or debug an HLL module. For this example, the HLL program has already been compiled and link-edited. The process for debugging HLL modules is similar for programs written in PL/I and C. However, there are some differences in the RID messages that are displayed. For this example, YOURPGM is the name of the HLL command processor that is monitored while executing under target task OPER1. OPER2 is the monitoring or debugging task.

To invoke RID, issue the following from OPER2:

```
RID TASK=OPER1
```

*Figure 85* shows the system response. RID defaults to STEP mode operation. Execution of YOURPGM is halted at each debug point.

```
- CNM01  CMN086I RID FUNCTION 'STEP' COMPLETED FOR TASK OPER1

???
```

*Figure 85. System Response to RID Invocation*

From OPER1, enter the name of the HLL program to debug:

YOURPGM
Figure 86 shows the panel displayed on OPER2’s console. This is the entry panel (HAPIENTR) for your PL/I program (ID=PLENTRY). The ID for the entry panel for a C program is CENTRY.

### Figure 86. PL/I Entry Screen for YOURPGM

The numbered boxes in Figure 86 are described as:

1. Message CNM987I displays:
   - **ID**: Unique identifier for the debug point being displayed
   - **MOD**: Name of the module being monitored by RID
   - **SEQ**: The sequence number of this RID panel
   - **TASK**: Name of the target task being monitored by RID
   - **TYPE**: Type of debug point currently being displayed

2. Message CNM988I displays the addresses of the main vector table (MVT), task vector block (TVB), task information block (TIB), transaction block (TRB), and the contents of register 13, which points to your save area. MVT, TVB, and TIB are described in *Tivoli NetView for z/OS Customization: Using Assembler*. The transaction block is used only by Tivoli Customer Support.

3. HLBPTR, BUFFER, ORIGBLCK are the initial parameters passed to your command processor from the NetView program. ISASIZ, HEAPSIZ and PLIOPTS are the default PL/I run-time values unless you have overridden these values in your HLL program.

4. Describes how each of the variables in 3 is declared:
   - **A**: Address
   - **C**: Character
   - **D**: Dump
   - **H**: Hex
   - **I**: Integer
   - **S**: String
   - **U**: Unsigned

5. Lengths of the variables in 3 that are expected by the HLL service routines.
Addresses in storage where the values for the variables in 3 are stored.

Values associated with each of the variables in 3.

Continue to the next debugging point by entering the following OPER2:
RID TASK=OPER1, CONTINUE

Because you did not specify OPTION, RID displays panels upon entrance to (HAPIENTR) and exit from (HAPIEXIT) all HLL service routines invoked from the HLL program being debugged. In Figure 87, RID is displaying the parameters on entry to (HAPIENTR) the CNMSMSG service routine. The parameters for each HLL service routines are explained in "Chapter 11. Service Reference" on page 209.

---

Figure 87. Entry Screen for CNMSMSG

Continue to the next debugging point by entering the following from OPER2:
RID TASK=OPER1

You do not need to issue the RID command with the CONTINUE option when you want to resume execution of a task. CONTINUE is used for readability. Figure 88 displays the parameters on exit from the CNMSMSG service routine. RETCODE is added and is the value of HLBRC (Hlbrc for C programs) on exit from an HLL service routine.

---

Figure 88. Exit Screen for CNMSMSG

Continue to the next debugging point by entering the following from OPER2:
RID TASK=OPER1
The final RID panel displayed in Figure 89 is a PL/I exit panel that corresponds to the PLENTRY panel in Figure 87 on page 207. Notice that TYPE= HAPIEXIT and ID=PLIEXIT. If this was a C program, ID would be CEXIT. RETCODE is added but ISASIZ, HEAPSIZ, and PLIOPTS are no longer displayed.

![Figure 89. PL/I Exit Screen for YOURPGM](image-url)
Chapter 11. Service Reference

Before reading this chapter, you should have an understanding of the information discussed in the previous chapters. This chapter describes the HLL commands and service routines, and their associated operands.

Composite Return Codes

Most of the NetView commands, command lists, and HLL service routines generate return codes upon completion. There are two types of return codes: simple and composite. A simple return code is a constant value that requires no computation. A composite return code is a calculated value that consists of a known (constant) value and one or more unknown values.

The return code section at the end of each HLL command and service routine provides a chart of:

- The return code represented in terms of constants and unknown values (if applicable)
- The return code represented in terms of resolved constants and unknown values (if applicable)
- A description of why the return code was issued

Several of the descriptions refer you to a NetView macro. Each of these macros is referenced in the Tivoli NetView for z/OS Customization: Using Assembler book.

To resolve the unknown values of a composite return code:

1. Start with this equation:
   
   \[ HLBRC = \text{Composite return code equation} \]

2. Resolve all known values. The first and most obvious known value is that of HLBRC. All other known values are represented as constants in DSIPCNM (see “Appendix A. PL/I Control Blocks and Include Files” on page 301) and DSICCNM (see “Appendix C. C Language Control Blocks and Include Files” on page 327). In the case where there is more than one unknown value to resolve (X and Y), the remaining calculated value is split into a major and minor return code.

The following examples show how to calculate unknown values of a composite return code.

Example 1

Upon completion of a call to CNMNAMS, HLBRC=4004. A return code value in the 4000 range implies that the composite return code equation is CMN_BAD_PUSH + X. (See “CNMNAMS (CNMNAMESTR): Named Storage” on page 268.)

The known values are resolved as follows:

\[
\begin{align*}
HLBRC &= \text{CMN_BAD_PUSH} + X \\
4004 &= 4000 + X \\
4004 - 4000 &= X \\
4 &= X
\end{align*}
\]
The unknown value X is equal to 4. The CNMNAMS return code indicates that the return code was generated by the DSIPUSH macro. The DSIPUSH macro, detailed in the Tivoli NetView for z/OS Customization: Using Assembler book, indicates that the return code is caused by insufficient storage.

Example 2

Upon completion of a call to CNMCMD, HLBRC=-3108. A return code value in the -3000 range implies that the composite return code equation is \( X - \text{CNM.BAD.EXCMS} \).

(See "CNMCMD (CNMCOMMAND): Invoke NetView Commands” on page 223.)

The known values are resolved as follows:

\[
\begin{align*}
\text{HLBRC} &= X - \text{CNM.BAD.EXCMS} \\
-3108 &= X - 3000 \\
-3108 + 3000 &= X \\
-108 &= X
\end{align*}
\]

The unknown value of X is equal to -108. The CNMCMD return code indicates that the return code was generated by the DSICES macro. Refer to the description of this macro in Tivoli NetView for z/OS Customization: Using Assembler.

The value of -108 in the previous example implies that \( X = \text{SWBEXCNF} - Y \). The known values are resolved as follows:

\[
\begin{align*}
-108 &= \text{SWBEXCNF} - Y \\
-108 &= -100 - Y \\
-108 + 100 &= -Y \\
12 &= -Y \\
8 &= Y
\end{align*}
\]

The unknown value Y is equal to 8. This DSICES return code designates that an immediate command was located and that the address is returned.

Example 3

Upon completion of a call to CNMCNMI, HLBRC=21600. A return code value greater than 20000 for CNMCNMI implies that the composite return code equation is \( \text{CNM.BAD.ZCMS} + (X * 100) + Y \).

The known values are resolved as follows:

\[
\begin{align*}
\text{HLBRC} &= \text{CNM.BAD.ZCMS} + (X * 100) + Y \\
21600 &= 20000 + (X * 100) + Y \\
1600 &= (X * 100) + Y
\end{align*}
\]

\[
\begin{align*}
1600 / 100 &= \text{MAJOR_RC is the quotient} \\
16 &= \text{MAJOR_RC} \\
00 &= \text{MINOR_RC is the remainder}
\end{align*}
\]

The unknown values X and Y are equal to 16 and 0 respectively. The CNMCNMI return code indicates that the return code was generated by the DSIZCSMS macro. The DSIZCSMS macro in Tivoli NetView for z/OS Customization: Using Assembler indicates that the return code is present because the call to CNMCNMI was not sent from a DST.

These return codes can also be seen in the initial return code value, 21600:

\[
\begin{align*}
1600 &= \text{MAJOR_RC} \\
00 &= \text{MINOR_RC}
\end{align*}
\]
Example 4

Upon completion of a call to the CNMCNMI service routine, HLBRC=20408. A return code value in the 20000 range implies that the composite return code equation is \( \text{CNM\_BAD\_ZCMSMS} + (X \times 100) + Y \).

The known values are resolved as follows:

\[
\begin{align*}
\text{HLBRC} &= \text{CNM\_BAD\_ZCMSMS} + (X \times 100) + Y \\
20408 &= 20000 + (X \times 100) + Y \\
20408 - 20000 &= (X \times 100) + Y \\
408 &= (X \times 100) + Y \\
=> 408 / 100 &= \text{MAJOR\_RC is the quotient} => 4 \\
&= \text{MINOR\_RC is the remainder} => 8
\end{align*}
\]

These return codes can also be seen in the initial return code value, 20408:

\[
408 \Rightarrow 4 = \text{MAJOR\_RC} \\
8 = \text{MINOR\_RC}
\]

The unknown values \( X \) and \( Y \) are equal to 4 and 8 respectively. The CNMCNMI return code indicates that the return code was actually generated by the DSIZCMSMS macro. The DSIZCMSMS macro return code (refer to Tivoli NetView for z/OS Customization: Using Assembler) indicates that the function could not be performed because of a date that is not valid.

Example 5

Upon completion of a call to CNMKIO, HLBRC=28692. Only two composite return codes are issued from CNMKIO. Because the return code value is \textit{not} in the 2000 range, the value of 28692 implies that the composite return code equation is \((\text{CNM\_BAD\_ZVSMS} + X) \times 256 + Y\) or \((100 + X) \times 256 + Y\).

This equation is difficult to resolve without knowing either \( X \) or \( Y \). Use these equations to determine the major and minor return codes:

\[
\begin{align*}
\text{MAJOR\_RC} &= (\text{HLBRC} / 256) - 100 \\
&= (28692 / 256) - 100 \\
&= 112 - 100 \\
&= 12
\end{align*}
\]

\[
\begin{align*}
\text{MINOR\_RC} &= \text{HLBRC} - ((\text{CNM\_BAD\_ZVSMS} + \text{MAJOR\_RC}) \times 256) \\
&= 28692 - (((100 + 12) \times 256) \\
&= 28692 - (112 \times 256) \\
&= 28692 - 28672 \\
&= 20
\end{align*}
\]

The return code section of CNMKIO indicates that the return code was actually generated by the DSIZVSMS macro. This code indicates that the VSAM function was not complete because the request was not valid or because an I/O scheduling error occurred. Refer to Tivoli NetView for z/OS Customization: Using Assembler for more information.
Command Reference

The commands described in the next sections are useful when executing HLL command processors. The GO, QUEUE, and RESET commands are operator commands. You can issue them from the operator console or from an HLL command processor through CNMCMD. The GLOBALV, TRAP, and WAIT commands must be issued from within an HLL command processor using CNMCMD.

Note: Refer to the Tivoli NetView for z/OS Command Reference or the NetView online help for more information about these commands.

GLOBALV Command

The GLOBALV command lets you define and manipulate task global and common global variables. In addition, you can save these variables to a VSAM database, restore them from the database, or purge them from the database. Task global variables are accessible only to the NetView task under which they were created. Common global variables are accessible to any NetView task.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>Not called from HLL.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>No storage available.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Action is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>Variable name is too long.</td>
</tr>
<tr>
<td>CNM_BAD_COMMAND</td>
<td>144</td>
<td>No action or variables specified.</td>
</tr>
<tr>
<td>CNM_BAD_POOL</td>
<td>156</td>
<td>A variable that the dictionary specified.</td>
</tr>
<tr>
<td>CNM_NO_DATA</td>
<td>268</td>
<td>No specified variables found in VSAM database for RESTORE request.</td>
</tr>
<tr>
<td>CNM_BAD_TASK</td>
<td>292</td>
<td>GLOBALV not called from OST, NNT, or PPT.</td>
</tr>
<tr>
<td>CNM_USER_ENDED</td>
<td>304</td>
<td>Reset request from user.</td>
</tr>
<tr>
<td>CNM_DST_ENDED</td>
<td>308</td>
<td>The DST ended with an error.</td>
</tr>
<tr>
<td>CNM_BAD_CHAR</td>
<td>540</td>
<td>A character that is not valid in the variable name.</td>
</tr>
<tr>
<td>CNM_BAD_COUNT</td>
<td>544</td>
<td>Odd count in double-byte character set (DBCS) shift-in and shift-out character.</td>
</tr>
<tr>
<td>CNM_BAD_MQS + X</td>
<td>1000 + X</td>
<td>Nonzero return code X from DSIMQS.</td>
</tr>
<tr>
<td>CNM_BAD_PUSH + X</td>
<td>4000 + X</td>
<td>Nonzero return code X from DSIPUSH.</td>
</tr>
<tr>
<td>CNM_BAD_KVS + X</td>
<td>11000 + X</td>
<td>Nonzero return code X from DSIKVS.</td>
</tr>
<tr>
<td>CNM_BAD_LCS + X</td>
<td>13000 + X</td>
<td>Nonzero return code X from DSILCS.</td>
</tr>
<tr>
<td>CNM_BAD_CDS + Z</td>
<td>14000 + Z</td>
<td>Nonzero return code Z. See values for Z below.</td>
</tr>
<tr>
<td>CNM_BAD_PAS + X</td>
<td>16000 + X</td>
<td>Nonzero return code X from DSIPAS.</td>
</tr>
<tr>
<td>CNM_BAD_PRS + X</td>
<td>25000 + X</td>
<td>Nonzero return code X from DSIPRS.</td>
</tr>
<tr>
<td>(CNM_BAD_ZVSMS + X)</td>
<td>(100 + X) * 256 + Y</td>
<td>Nonzero return code from DSIZVSMS. X is the major return code and Y is the minor return code.</td>
</tr>
</tbody>
</table>
GLOBALV Command

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) or the MVS/ESA VSAM library for more information.

**Values for Z**

<table>
<thead>
<tr>
<th>Value for Z</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Variable name that is not valid. Asterisk used in DEFx, GETx, or PUTx.</td>
</tr>
<tr>
<td>12</td>
<td>Insufficient storage.</td>
</tr>
<tr>
<td>20</td>
<td>Value length limit exceeded.</td>
</tr>
<tr>
<td>28</td>
<td>No command procedure related to action.</td>
</tr>
<tr>
<td>32</td>
<td>Data truncated.</td>
</tr>
</tbody>
</table>

Refer to the [Tivoli NetView for z/OS Command Reference](#) or the NetView online help for more information about the GLOBALV command.

**GO Command**

Use the GO command to resume running a command procedure in a pause or wait state. You can also use the GO command to pass values to a command procedure in a pause state.

Enter the GO command from a terminal to satisfy a wait or pause. A return code of CNM_GO_ON_WAIT is generated when GO is entered in these situations:

- The event specified on the WAIT command is MESSAGES (with or without a time specified).
- The event specified on the WAIT command is DATA (with or without a time specified).
- A time is specified on the WAIT command with no specified events.

A return code of CNM_OPINPUT_ON_WAIT is generated if one of the events specified on the WAIT command is OPINPUT. You can enter GO alone or with operator input. See "[WAIT Command](#)" on page 218 for more information about satisfying a wait.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from the DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_MQS + X</td>
<td>1000 + X</td>
<td>Nonzero return code (X) from the DSIMQS macro.</td>
</tr>
<tr>
<td>CNM_BAD_LCS + X</td>
<td>13000 + X</td>
<td>Nonzero return code (X) from the DSILCS macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

Refer to the [Tivoli NetView for z/OS Command Reference](#) or the NetView online help for more information about the GO command.
PIPE Command

The PIPE command is a powerful and flexible command. It lets you issue commands and manipulate messages in a pipeline. Many of the functions available in the HLL environment as commands and service routines are also available through the PIPE stage commands.

The PIPE command creates a message processing environment that lets you issue a command and wait for that command’s response messages (either synchronous or asynchronous). PIPE ensures that only messages correlated to the command are received and processed in the pipeline. You can use the PIPE command to process commands that generate correlated responses instead of using the WAIT and TRAP commands. The command and message correlation provided through the PIPE command is a major advantage over the WAIT command. See Figure 21 on page 74 and Figure 58 on page 147 for examples of using PIPE in an HLL program.

For a description of the advantages of using the PIPE command instead of WAIT command, refer to Tivoli NetView for z/OS Customization: Using Pipes.

The PIPE command also provides support for stem variables in the NetView HLL environment. See Figure 24 on page 78 and Figure 61 on page 154 for an example of how to access local variables that have been stored in stem variables by the PIPE command.

The PIPE stage command, < (From Disk), provides similar function to the HLL service routines CNMMEMO, CNMMEMR and CNMMEMC. The PIPE stage commands, CONSOLE and LOGTO, provide function that is available through the CNMSMSG service routine. The PIPE command with these stages can be used in place of the associated HLL service routine in these cases. However, RID support is available only when the HLL service routine is used.

Notes:

1. The PIPE command cannot be issued from an HLL installation exit routine or from an HLL command procedure running under a DST. Refer to Tivoli NetView for z/OS Customization: Using Pipes for a complete description of the PIPE command.

2. To obtain information about the PIPE command and stage commands, use the online HELP facility:
   - To display information about the PIPE command, enter:
     
     HELP PIPE
   
   - To display information about a specific stage command, enter:
     
     HELP PIPE stage_command

QUEUE Command

The QUEUE command adds a text message to the operator input queue (OPERQ) of an HLL command processor or installation exit routine running with the HLL_QUEUED_INPUT bit of HLLOPTS turned on.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from the DSIGET macro.</td>
</tr>
</tbody>
</table>
Return Code | Name          | Value | Description
------------|---------------|-------|---------------------------------------
CNM_BAD_MQS + X | 1000 + X     |       | Nonzero return code (X) from the DSIMQS macro.

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information. Refer to the [Tivoli NetView for z/OS Command Reference](#) or the NetView online help for more information about the QUEUE command.

**RESET Command**

High-level language support gives you the option of specifying whether or not a command procedure is cancelable. The reset flag is set on only when RESET, LOGOFF, or CLOSE IMMED are being issued. See "HLL Run-Time Options" on page 38 for more information.

If a command procedure can be cancelled, it behaves according to the rules specified under the RESET command in the NetView online help. The description for RESET NORMAL states that a command stops at its next breakpoint whenever RESET NORMAL is issued. A breakpoint occurs in an HLL command procedure whenever an HLL service routine is started.

If a cancelable command procedure is reset with RESET NORMAL, the command procedure terminates with a -5 return code and terminates its caller if the caller is also cancelable.

If the command procedure cannot be cancelled, it is reset only when RESET IMMED and RESET DUMP are issued.

Whenever RESET is issued, NetView turns on a reset flag that remains on until the command procedure returns control to NetView or uses the CNMINFI service routine to check RESETREQ.

If RESET NORMAL is issued while a noncancelable command procedure is running, the reset flag remains on until the noncancelable command procedure either calls or returns to a cancelable command procedure.

A noncancelable command procedure can check to see if an operator has attempted to reset it by using the RESETREQ function of the CNMINFI service routine. To cancel itself, a command procedure must return to NetView with a -5 return code. This cancels the command procedure that had this return code and all cancelable callers to this command procedure. Cancelable callers include command lists that call a cancelled command procedure, and cancelable command procedures that directly invoke a cancelled command procedure.

Checking the reset flag sets the flag off. As a result, the command procedure that checks the flag must reset the flag. You cannot pass on this responsibility to a higher level through the use of the reset flag. (You can set a user-defined flag for this purpose, but doing so is not recommended.)

**Examples**

In this example, cancelable command procedure X calls cancelable command procedure Y, which calls cancelable command procedure Z. RESET NORMAL is entered while command procedure Z is running.
As a result of entering RESET NORMAL, Z is reset. Z returns a -5 return code to Y, which causes Y to be reset. Y returns a -5 return code to X, and X is reset.

Keep in mind that the HLL command procedures must invoke the CNMINFI service routines for the reset bit to be checked. Y and X continue executing if they do not invoke any more HLL service routines.

Here cancelable command procedure X calls noncancelable command procedure Y, which calls cancelable command procedure Z. RESET is entered when command procedure Z is running.

As a result of entering RESET NORMAL, Z is reset. Y receives a -5 return code from the call to Z. Y is not automatically reset by NetView. It can, if it chooses, check the return code and cancel itself, but this is the responsibility of Y. Even though X is cancelable, it is not cancelled because Y is at a lower execution level, and Y is not reset.

In this example, noncancelable command procedure X calls noncancelable command procedure Y. Noncancelable command procedure Y returns control to noncancelable command procedure X. Noncancelable command procedure X then calls cancelable command procedure Z. RESET is entered just as X begins execution.
Even though RESET NORMAL is external, X is not cancelled because it is defined as noncancelable. X calls Y. Y is not cancelled because Y is defined as noncancelable. Y returns control to X, and then X calls Z. Z is reset and returns control to X with a -5 return code.

Notes:
1. In the example referred to by Figure 92, command procedure X could have checked the reset flag using the HLL service routine CNMINFI. In this case, the reset flag would be turned off and Z would not be reset.
2. Create command procedures that can be cancelled when possible. The HLL support option that cannot be cancelled is provided so that you can code command procedures to perform cleanup (such as free storage) before the procedure is cancelled.
3. When an HLL command procedure is cancelled, the cleanup done is equivalent to that done by STOP in PL/I and EXIT in C.

When cleanup is necessary:

In this scenario, if RESET is issued while Y is executing, Y is terminated and X performs the cleanup.
TRAP Command

The TRAP command enables you specify message trapping criteria for HLL and REXX command procedures. When the TRAP command is issued, all subsequent messages that match the conditions defined by the trapping criteria are added to the message queue (TRAPQ).

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>TRAP was not issued from an HLL or REXX command procedure.</td>
</tr>
<tr>
<td>CNM_BAD_SYNTAX</td>
<td>12</td>
<td>Syntax error. When DSI028I is issued, check domainid.token syntax. When DSI604I is issued, check the format of the TRAP command according to the operand specified.</td>
</tr>
<tr>
<td>CNM_BAD_COMMAND</td>
<td>144</td>
<td>TRAP was not issued from a command procedure running under an OST or NNT.</td>
</tr>
<tr>
<td>CNM_BAD_MRBLD + X</td>
<td>18000 + X</td>
<td>Nonzero return code X. See values for X below.</td>
</tr>
</tbody>
</table>

Values for X

<table>
<thead>
<tr>
<th>Return Code Value</th>
<th>Value for X</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18000 + X</td>
<td>8</td>
<td>Request for storage has failed.</td>
</tr>
</tbody>
</table>

Refer to the [Tivoli NetView for z/OS Command Reference](URL) or the NetView online help for more information about the TRAP command.

WAIT Command

When issued from a command procedure, the WAIT command temporarily suspends processing of that command procedure until a specified event occurs. For an HLL command procedure, the event can be one or more messages, operator input, data, a certain period of time, or any combination of these four. The first occurrence of one of these events satisfies the wait and processing is resumed.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>WAIT was not issued from an HLL or REXX command procedure.</td>
</tr>
<tr>
<td>CNM_TOO_MANY</td>
<td>8</td>
<td>Too many operands.</td>
</tr>
<tr>
<td>CNM_BAD_SYNTAX</td>
<td>12</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>CNM_BAD_COMMAND</td>
<td>144</td>
<td>WAIT was not issued from a command procedure running under an OST or NNT.</td>
</tr>
<tr>
<td>CNM_NO_TRAP</td>
<td>152</td>
<td>WAIT was issued but TRAP was not issued. You must issue valid TRAP before issuing WAIT for messages.</td>
</tr>
<tr>
<td>CNM_NO_PREV_WAIT</td>
<td>248</td>
<td>No previous WAIT; WAIT CONTINUE is not valid.</td>
</tr>
</tbody>
</table>
Return Code Name | Value | Description
--- | --- | ---
CNM_STRG_FAILURE | 616 | DSIGET failure in a service routine.

**Note:** The following return codes are issued when a wait is satisfied. They are generated in place of a 0 return code to inform the operator which event satisfied the wait.

- CNM_TIME_OUT_WAIT | 224 | WAIT timed out.
- CNM_GO_ON_WAIT | 228 | GO satisfied wait.
- CNM_MSG_ON_WAIT | 232 | Message received during wait.
- CNM_OPINPUT_ON_WAIT | 236 | OPINPUT received during wait.
- CNM_DATA_ON_WAIT | 240 | DATA received during wait.

Refer to the [Tivoli NetView for z/OS Command Reference](#) or the NetView online help for more information about the WAIT command.

## HLL Service Routine Reference

The following service routines can be started from a command processor or installation exit routine written in PL/I or C. A description of each service routine is given, along with its associated parameters, usage notes, and return codes.

When writing a command processor or installation exit routine in PL/I, you can invoke an HLL service routine using the call or macro format. The PL/I macro format is provided for those users who want to code only the required parameters for a particular HLL service routine invocation. You must code all of the parameters when using the PL/I call format or C invocation.

When you invoke a service routine from an HLL command processor or installation exit routine written in C, the first letter of the service routine name must be capitalized and the remaining letters must be lowercase. This restriction is a result of C being case-sensitive. PL/I does not have this restriction.

### CNMALTD (CNMALTDATA): Alter Data on a Queue

The CNMALTD routine enables you to alter the contents of the top message on the initial data queue. Lines can be inserted, replaced, or deleted.

The CNMALTD routine syntax follows:

**PL/I CALL FORMAT:**
```
CALL CNMALTD(hlbptr,adfunc,adbuf,adorigin,adqueue,adindex)
```

**PL/I MACRO FORMAT:**
```
CNMALTDATA FUNC(adfunc) DATA(adbuf) ORIGIN(adorigin) QUEUE(adqueue) LINE(adindex)
```

**C INVOCATION:**
```
void Cnmaltd(char *adfunc, void *adbuf, void *adorigin, int adqueue, int adindex)
```

**Where:**

- **adbuf**
  
  Is a varying length character field containing the buffer to be inserted. This field is required with INSLINE and REPLINE, but not used with DELLINE.
CNMALTD (CNMALTDATA)

`adfunc`

Is an 8-byte character field that specifies the function to be performed. This field is required for all CNMALTD calls. The possible values follow:

**DELLINE**

Deletes a line of the current message in the specified queue. The line specified by `adindex` (the index value) is physically removed from the queue. You can delete all lines of a message. If the line that was last returned from GETLINE or GETMSG is deleted, the message pointer is moved back to the line preceding the deleted line. (See [CNMGETF](CNMGETDATA): Data Queue Manipulation on page 240 for information.) `adqueue` and `adindex` are required operands for DELLINE. `adbuf` and `adorigin` are not required operands for this function.

**INSLINE**

Inserts a new line in the message in the specified queue. `adindex` specifies the line number that the new line will have after it is inserted. The index value can be one greater than the number of lines currently in the message to add a line on the end. All parameters are required for this function.

**REPLINE**

Replaces a line of the current message in the specified queue (if it exists). All parameters are required for this function.

`adindex`

Is a 4-byte integer field containing the line number of the message at the head of the queue to be manipulated. This field is required for all functions.

`adorigin`

Is a character field of fixed length `n` (where `n ≥ 38`) to contain an origin block. You must define an origin block (`adorigin`) to be passed as a parameter to CNMALTD. This must be a separate structure from the origin block (ORIGBLCK) that was passed to the HLL command processor or installation exit routine as an initial parameter. ORIG_BLOCK_LENGTH cannot be less than 38. See DSIPORIG in Appendix A. PL/I Control Blocks and Include Files on page 301 or DSICORIG in Appendix C. C Language Control Blocks and Include Files on page 327 for the PL/I and C mappings of an origin block. You are responsible for updating the origin block (`adorigin`) to reflect changes made by CNMALTD. This field is required for INSLINE and REPLINE, but not for DELLINE.

`adqueue`

Is a 4-byte integer field containing the number (index) of the queue on which the operation is to be performed. The only queue allowed for CNMALTD is the initial data queue (IDATAQ). The full message that started the HLL command processor through NetView automation or the message that drives DSIEX02A is on the initial data queue. This field is required for all functions.

`hlbptr`

Is a 4-byte pointer field containing the address of the HLB control block.

Usage Notes:

1. CNMALTD is primarily designed for use in DSIEX02A, where it enables you to alter messages before they are automated or displayed.
2. See "CNMSMSG (CNMSENDMSG): Send Message or Command" on page 282 for the definitions of line types.
Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect adfunc value.</td>
</tr>
<tr>
<td>CNM_BAD_QUEUE</td>
<td>72</td>
<td>Incorrect adqueue value.</td>
</tr>
<tr>
<td>CNM_BAD_INDEX</td>
<td>76</td>
<td>Incorrect adindex value.</td>
</tr>
<tr>
<td>CNM_QUEUE_EMPTY</td>
<td>80</td>
<td>The specified queue is empty.</td>
</tr>
<tr>
<td>CNM_BAD_ORIGBLOCK</td>
<td>84</td>
<td>Incorrect value in ORIG_BLOCK_LENGTH.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>The length of (adbuf) is greater than (&gt;32729).</td>
</tr>
<tr>
<td>CNM_NOT_MLWTO</td>
<td>92</td>
<td>Message is not a multiline message.</td>
</tr>
<tr>
<td>CNM_BAD_LINETYPE</td>
<td>96</td>
<td>Incorrect line type. Must be C,D,E,L,F, or '.'.</td>
</tr>
<tr>
<td>CNM_NO_LINES_IN_MSG</td>
<td>264</td>
<td>All lines in the specified message have been deleted.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMAUTO (CNMAUTOTAB): Invoke Automation Table**

This service routine enables you to invoke the NetView automation table from an HLL command processor or installation exit.

The CNMAUTO routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMAUTO(hlbptr,atdata)
```

**PL/I MACRO FORMAT:**

```pli
CNMAUTOTAB DATA(atdata)
```

**C INVOCATION:**

```c
void Cnmauto(void *atdata)
```

**Where:**

- **atdata**
  
  Is a varying length character field containing the data item to be automated. The data item must be in one of these forms:
  
  - A MDS-MU
  - A CP-MSU
  - A NMVT

  The CNMAUTO routine determines which form of data item has been passed in **atdata** and takes appropriate action to calculate the length of the data item.

  Refer to the SNA library for more information.

- **hlbptr**

  Is a 4-byte pointer field containing the address of the HLB control block.
**CNMAUTO (CNMAUTOTAB)**

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>A match was found and the data item was automated.</td>
</tr>
<tr>
<td>CNM_FAIL_TO_AUTOMATE</td>
<td>24000 + X</td>
<td>Nonzero return code, X, from DSIAUTO macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMCELL (CNMSTRCELL): Storage Cell**

You can use the CNMCELL service routine to allocate and free storage cells from a previously allocated storage pool. The token obtained by CNMPOOL must be passed to CNMCELL to identify the storage pool.

The CNMCELL routine syntax follows:

**PL/I CALL FORMAT:**

CALL CNMCELL(hlbptr,pcfunc,pctoken,pcstrptr)

**PL/I MACRO FORMAT:**

CNMSTRCELL FUNC(pcfunc) TOKEN(pctoken) STRPTR(pcstrptr)

**C INVOCATION:**

void Cnmcell(char *pcfunc, int pctoken, void *pcstrptr)

Where:

- *hlbptr* is a 4-byte pointer field containing the address of the HLB control block.
- *pcfunc* is an 8-byte character field that specifies the function to be performed:
  - **ALLOC** Allocate cell
  - **FREE** Free cell
- *pcstrptr* is a 4-byte pointer field to contain the address of the cell. This field is returned to the caller for ALLOC and provided by the caller for FREE.
- *pctoken* is a 4-byte integer field containing the token identifying the storage pool. The caller provides this field for all functions (the token is returned from CNMPOOL).

**Usage Note:** A storage cell within a pool is associated with the NetView subtask under which it was allocated. The storage cell cannot be referenced from a task other than the one with which it is associated.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>Return Code Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>CNM_BAD_TOKEN</td>
<td>32</td>
<td>Incorrect <code>pctoken</code>.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect <code>pcfunc</code></td>
</tr>
<tr>
<td>CNM_BAD_CLASS</td>
<td>112</td>
<td>Possible storage overlay. Report to Tivoli Customer Support.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by <code>pcstrptr</code> is not addressable.</td>
</tr>
<tr>
<td>CNM_NOT_IN_POOL</td>
<td>204</td>
<td>Cell not in storage pool.</td>
</tr>
<tr>
<td>CNM_BAD_CELL_ADDRESS256</td>
<td></td>
<td>Address is not on valid cell boundary.</td>
</tr>
<tr>
<td>CNM_CELL_ALREADY_FREE60</td>
<td></td>
<td>Cell has already been freed.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](Tivoli%20NetView%20for%20z/OS%20Customization%3A%20Using%20Assembler) for more information.

**CNMCMMD (CNMCOMMAND): Invoke NetView Commands**

The CNMCMMD service routine enables you to execute a NetView command from an HLL command processor. If the called command is a long-running command, the caller is suspended until the long-running command is completed. The caller regains control at the instruction following the CNMCMMD invocation.

The CNMCMMD routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMCMMD(hlbptr,cmdstr)
```

**PL/I MACRO FORMAT:**

```pli
CNMCOMMAND DATA(cmdstr)
```

**C INVOCATION:**

```c
void Cnmcmd(void *cmdstr)
```

Where:

- `cmdstr` is a varying length character field containing the NetView command (including its parameters) to be executed.
- `hlbptr` is a 4-byte pointer field containing the address of the HLB control block.

**Usage Notes:**

1. Commands are started with a HDRMTYPE of HDRTYPEC. This type is consistent with the NetView command list language and REXX.

2. The return code from HLL command processors and NetView long-running commands is returned properly to HLL command processors through the CNMCMMD interface. If you wish to call a long-running command and allow it to be separately rollable, you can prefix the command with CMD HIGH. For example, `CNMCOMMAND DATA ('CMD HIGH BROWSE NETLOGA')` enables the BROWSE panel to roll independently from the calling HLL command processor. Refer to the NetView online help for more information about the CMD command.

3. CNMCMMD does not process immediate commands (type=I in DSICMD).
CNMCMD (CNMCOMMAND)

4. A negative return code generated from CNMCMD indicates a failure in the CNMCMD service routine, whereas a positive return code generated from CNMCMD indicates a failure in the NetView command that was to be executed by CNMCMD. A -5 return code generated from CNMCMD indicates that the NetView command executing was cancelled. In this case, the command processor should do any necessary cleanup and exit setting of HLBRC to -5 to pass the RESET information to its caller.

5. A -1 return code generated from CNMCMD indicates an unexpected error in the called command procedure.

6. You cannot invoke CNMCMD from an HLL installation exit routine or from an HLL command processor while holding a lock.

7. The NetView service point command service (SPCS) commands are not supported under the HLL API and must not be started by CNMCMD. Refer to the NetView online help for a list of SPCS commands.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>-CNM_BAD_INVOCATION</td>
<td>-4</td>
<td>Not started from a command processor.</td>
</tr>
<tr>
<td>-CNM_NO_STORAGE</td>
<td>-24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>-CNM_BAD_LENGTH</td>
<td>-88</td>
<td>Command length is not valid.</td>
</tr>
<tr>
<td>-CNM_LOCKED</td>
<td>-208</td>
<td>CNMCMD issued while holding a lock.</td>
</tr>
<tr>
<td>X-CNM_BAD_EXCMS</td>
<td>X-3000</td>
<td>Nonzero return code X. See values for X that follow.</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>Return code from executed command. See CNMCMD usage notes for return codes -5 and -1.</td>
</tr>
</tbody>
</table>

Values for X

<table>
<thead>
<tr>
<th>Return Code Value for X</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>Nonzero return code, 4 (drop), from installation exit.</td>
</tr>
<tr>
<td>-100-Y</td>
<td>Nonzero return code, Y, from DSICES macro.</td>
</tr>
<tr>
<td>-200-Y</td>
<td>Nonzero return code, Y, from DSILCS (CWB) macro.</td>
</tr>
<tr>
<td>-300-Y</td>
<td>Nonzero return code, Y, from DSIGET macro.</td>
</tr>
<tr>
<td>-400-Y</td>
<td>Nonzero return code, Y, from DSIPRS macro.</td>
</tr>
<tr>
<td>-500-Y</td>
<td>Nonzero return code, Y, from either DSIGET or DSILCS indicates storage failure.</td>
</tr>
<tr>
<td>-600-Y</td>
<td>Nonzero return code, Y, from DSILCS (SWB) macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

CNMCNMI (CNMI): CNMI Access Under a DST

The CNMI service of NetView enables HLL command processors running under a DST to send and receive data across the CNMI. You can use this service in conjunction with CNMGETD to manipulate data on the CNMI solicited data queue (CNMIQ).
The CNMI routine syntax follows:

**PL/I CALL FORMAT:**
```
CALL CNMCNMI(hlbptr,cnfunc,cndata,cndest,cntimout)
```

**PL/I MACRO FORMAT:**
```
CNMI FUNC(cnfunc) DATA(cndata) DEST(cndest) TIMEOUT(cntimout)
```

**C INVOCATION:**
```
void Cnmcnmi(char *cnfunc, void *cndata, char *cndest, int cntimout)
```

**Where:**

- **cndata**
  A varying length character field containing the RU to be sent (beginning with an RH header). This field is required for all functions. The RU length must be at least 3 bytes and no longer than 32729 characters.

- **cndest**
  An 8-byte character field that specifies the name of the PU to which the RU should be sent. This field is required for all functions.

- **cnfunc**
  An 8-byte character field that specifies the function to be performed. This field is required for all CNMCNMI calls.

- **SENDRESP**
  Sends RUs and expects only a positive or negative response.

- **SENDRPLY**
  Sends RUs and expects a reply RU or a negative response.

- **cntimout**
  A 4-byte integer field specifying the number of seconds to wait for a reply or response. This is an optional field. If you do not specify `cntimout`, the default is 0. If you specify a timeout, the RH header must indicate that the embedded network services (NS) RU solicits a reply. This causes NetView to generate a procedure-related identifier (PRID). Refer to the VTAM library for more information. For requests that generate multiple RU (chained) replies, `cntimout` applies only to the first RU in the chain.

- **hlbptr**
  A 4-byte pointer field containing the address of the HLB control block.

**Usage Notes:**

1. You cannot invoke CNMCNMI from an HLL command processor while holding a lock. HLL command processors enter a wait state when sending requests over the CNMI. The wait ends when a response or reply is received or when the specified timeout expires.
2. You cannot issue CNMCNMI from an HLL installation exit routine.
3. Responses to CNMI solicited data requests are placed on the CNMI solicited data queue (CNMIQ).
4. The XITCI installation exit routine is started for both solicited and unsolicited data. See [Chapter 2. HLL Installation Exit Routines on page 11](#) for more information about this exit. Also see [Chapter 3. HLL Data Services Command Processors on page 22](#) for a discussion on “Unsolicited HLL Data Services Command Processors (DSCP).” When the unsolicited HLL DSCP receives...
control, the command buffer (CMDBUF) contains the unsolicited data RU. For more information about installing a DST, see "Chapter 3. HLL Data Services Command Processors" on page 27.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>Not started from a command processor or not under a DST.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_RULENG</td>
<td>48</td>
<td>Incorrect cndata length.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect cnfunc.</td>
</tr>
<tr>
<td>CNM_BAD_TIMEOUT</td>
<td>56</td>
<td>cntimout less than (&lt;) 0.</td>
</tr>
<tr>
<td>CNM_NEED_PRID</td>
<td>60</td>
<td>Timeout specified but PRID not generated. The PRID generation bit in the RU must be set if a timeout is specified.</td>
</tr>
<tr>
<td>CNM_NEG_RESPONSE</td>
<td>64</td>
<td>Negative response received. (Sense code in HLBSENSE.)</td>
</tr>
<tr>
<td>CNM_TIME_OUT</td>
<td>68</td>
<td>Timeout occurred.</td>
</tr>
<tr>
<td>CNM_LOCKED</td>
<td>208</td>
<td>CNMI issued while holding a lock.</td>
</tr>
<tr>
<td>CNM_DST_FAILURE</td>
<td>2000 + X</td>
<td>Nonzero return code, X, which is the DSRB minor return code for solicited CNMI data or VSAM data set services.</td>
</tr>
<tr>
<td>CNM_BAD_ZCSMS</td>
<td>20000 + (X * 100) + Y</td>
<td>Nonzero return code (major), X, and nonzero return code (minor), Y, from DSIZCSMS.</td>
</tr>
</tbody>
</table>

Refer to Tivoli NetView for z/OS Customization: Using Assembler or the MVS/ESA VSAM library for more information.

**CNMCPYS (CNMCOPYSTR): Copy Storage**

The CNMCPYS service routine enables you to copy storage from one address to another. If the source or destination is not addressable, this service routine enables the copy operation to process without ABENDING. However, the service routine does not stop you from overwriting storage if it is addressable.

The CNMCPYS routine syntax follows:

**PL/I CALL FORMAT:**
CALL CNMCPYS (hlbptr, csfrom, csto, cslen, cstype)

**PL/I MACRO FORMAT:**
CNMCPYSSTR FROM(csfrom) TO(csto) LENG(cslen) COPYTYPE(cstype)

**C INVOCATION:**
void Cnmcpys(void *csfrom, void *csto, int cslen, char *cstype)

**Where:**

*csfrom*

Is a 4-byte pointer field containing the address of the source data.
**cslen**

Is a 4-byte integer field containing the number of bytes of storage (0 to 16777215) to be copied.

If the value specified by cslen is greater than the actual length of the specified csfrom buffer, a storage overlay could occur. Take special care when deciding the value of cslen.

**csto**

Is a 4-byte pointer field containing the address of the destination.

**cstype**

Is the type of copy to perform. Valid types are:

- **FIXTOFIX**
  Copy cslen bytes of storage from a fixed-length buffer to another fixed-length buffer.

- **FIXTOVAR**
  Copy cslen bytes of storage from a fixed-length buffer to a varying length buffer.

- **VARTOFIX**
  Copy cslen bytes of storage from a varying length buffer to a fixed-length buffer.

- **VARTOVAR**
  Copy cslen bytes of storage from a varying length buffer to another varying length buffer.

**hlbptr**

Is a 4-byte pointer field containing the address of the HLB control block.

**Usage Notes:**

1. The length field of varying length buffers is not set or altered by CNMCPYS.
2. When using CNMCPYS with C and when copying FIXTOFIX, FIXTOVAR, or VARTOFIX, you must pass CNMCPYS a pointer to a pointer to your fixed-length buffer. You can do this by designating a variable as a pointer to a string, and then passing the address of that pointer to CNMCPYS.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>cslen greater than (&gt; 16777215 or less than (&lt;) 0. Copy not performed.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by csto or csfrom is not addressable.</td>
</tr>
<tr>
<td>CNM_BAD_CSTYPE</td>
<td>252</td>
<td>Incorrect cstype.</td>
</tr>
<tr>
<td>CNM_BAD_ESTAE</td>
<td>15000</td>
<td>Nonzero return code from ESTAE macro.</td>
</tr>
</tbody>
</table>

Refer to the MVS/XA™ library for more information.
CNMC2T (CNMCODE2TXT)

CNMC2T (CNMCODE2TXT): Code Point Translation

You can use NetView with a problem-management database to open problem records when NetView alerts are received. The code point translation service routine is provided in PL/I and C to translate the numeric code points, received in the alert, into readable text.

Note: This function is not supported when NetView is in a distributed system configuration.

The CNMC2T service routine provides translation for various types of code points to National Language Support (NLS) text.

The CNMC2T routine syntax follows:

PL/I CALL FORMAT:
CALL CNMC2T(hlbptr, trdata, trdatlen, trtable, trcode)

PL/I MACRO FORMAT:
CNMCODE2TXT DATA(trdata) LENG(trdatlen) TABLE(trtable) CODE(trcode)

C INVOCATION:
void Cnmc2t(void *trdata, int trdatlen, char *trtable, int trcode)

Where:

hlbptr
  Is a 4-byte pointer field containing the address of the HLB control block.

trcode
  Is a 4-byte integer field containing the code point to be translated.

trdata
  Is a varying length character field to which the code point text is to be returned.

trdatlen
  Is a 4-byte integer field containing the length of trdata. This is the maximum length of the area provided to receive the returned text.

If the value specified by trdatlen is less than the length of the text to be returned, the truncated text is returned in trdata and a return code of CNM_DATA_TRUNC is generated. The full length of the text that was truncated is stored in HLBLENG (Hlbleng).

If the value specified by trdatlen is equal to or greater than the length of the text to be returned, and HLBRC (Hlbrc) = CNM_GOOD, the length of the returned text is stored in HLBLENG (Hlbleng).

If the value specified by trdatlen is greater than the length of the receiving data buffer (trdata), a storage overlay can occur. Take special care when deciding the value of trdatlen.

trtable
  Is an 8-byte character field that specifies the table to be used in translating the code point. Valid table names are:
  SNAALERT  SNA alert description code point
  SNACause  SNA probable cause
  SNADDDATA SNA detailed data
  SNADDDAT5 SNA detailed data code point, subfield X'85'
SNADDAT6  SNA detailed data code point, subfield X'86'
SNAFCAUS  SNA failure cause
SNAICAUS  SNA install cause
SNAREACT  SNA recovery actions
SNAUCAUS  SNA user cause

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>Table not found.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>Data truncated; \trdatlen\ is too small.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>Address of \trdata\ is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_TABLE</td>
<td>312</td>
<td>Incorrect table name.</td>
</tr>
<tr>
<td>CNM_BAD_CODE</td>
<td>316</td>
<td>Code point not found; length of \trdata\ is 0.</td>
</tr>
</tbody>
</table>

CNMGETA (CNMGETATTR): Query Message Attributes

You can use the CNMGETA service routine to obtain attributes of messages on the initial data queue (IDATAQ). You can also retrieve action messages, DOM information for messages, and DOMs processed by HLLs. The values of the attributes are returned to you in character string form.

The CNMGETA routine syntax follows:

**PL/I CALL FORMAT:**

```pli
call cnmgeta(hlbptr,ganame,gadata,gadatlen,gaqueue)
```

**PL/I MACRO FORMAT:**

```pli
cnmgetattr item(ganame) data(gadata) leng(gadatlen) queue(gaqueue)
```

**C INVOCATION:**

```c
void Cnmgeta(char *ganame, void *gadata, int gadatlen, int gaqueue)
```

**Where:**

- `gadata`
  A varying length character field containing the resulting value for the specified attribute.

  **Note:** If an attribute does not apply to the particular message or MSU type data, the `gadata` field is set to a null value for character data. Bit fields are returned as all character zeros.

- `gadatlen`
  A 4-byte integer field containing the length of `gadata`. This is the maximum length of the area provided to receive the returned data. You provide the value for `gadatlen`.

  If the value specified by `gadatlen` is less than the length of the data to be returned, the truncated data is returned in `gadata` and a return code of CNM_DATA_TRUNC is generated. The full length of the data that was truncated is stored in HLBLENG (Hlbleng).
If the value specified by *gadatlen* is equal to or greater than the length of the data to be returned, and HLBRC (*Hlbrc*) = CNM_GOOD, the length of the returned data is stored in HLBLENG (*Hlbleng*).

If the value specified by *gadatlen* is greater than the length of the receiving data buffer (**gadata**), a storage overlay could occur. Take special care when deciding the value of *gadatlen*.

**ganame**

Is an 8-byte character field that specifies the attribute. Valid attributes are:

**ACTIONDL**

An EBCDIC string that indicates why the NetView action message was deleted. Values follow:

- (null) The message is not a DOM (delete operator message).
- LOCAL The message was deleted by operator overstrike or by the CONSOLE DELETE stage.
- NETVIEW The message was deleted by the NetView DOM command, using the NVELID option, or internally by NetView.

**SMSGID**

The message was deleted by an MVS DOM-by-SMSGID. A single message was deleted by its specific ID. This is the most frequent type of MVS DOM.

**TOKEN**

The message was deleted by an MVS DOM-by-token.

**TCB**

The message was deleted because the task ended that issued the message. In some cases MVS converts these messages into DOM-by-SMSGID.

**ASID**

The message was deleted because the address space that issued the message ended. In some cases MVS converts these messages into DOM-by=SMSGID.

**INVALID**

The DOM contained an unrecognizable combination of bit settings. This could indicate a user exit problem or a data area overlay problem.

**ACTIONMG**

An action message with of value of 1 or 0. For a value of 1, NetView treats the message as an MVS action message. Action messages are WTORs, and messages are marked as descriptor code 1, 2, or 11.

**ATTNID**

Is a VSE attention identifier. A plus sign (+) indicates that a reply is required for this message immediately. A (-) indicates that a reply is required for this message.

This function has a value if the message is from a VSE system, but null for non-VSE messages.

**AUTOTOKE**

Is a 1- to 8- character name of the MVS message processing facility (MPF) automation token. This can also be set as a result of a DSIVSAM or DSIVSMX request.
**Note:** If you specify AUTO(YES) or AUTO(NO) in the MPF table, the values “YES” and “NO” are not automation tokens.

**CART**
Is a 8-byte MVS command and response token (CART). The CART might contain non-displayable characters.

**Notes:**
1. CART only has a value if the messages were originally a MDB.
2. CARTs are included in MVS system messages on systems running MVS 4.1 or later.

**DESC**
Are the MVS descriptor codes as a series of 16, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. Refer to the MVS/ESA library for more information about code values.

**IFRAUGMT**
Is the Greenwich Mean Time when the automated internal function request (AIFR) was created. This value is returned as an 8-byte hexadecimal value.

**IFRAUIND**
Returns 2-byte of indicator bits as a series of 16, 1 (on) and 0 (off) EBCDIC characters represent the bits in order. This data is mapped in DSIIFR. The bit positions are:
1. MVS system information attached (WQE data).
2. Message from NetView PPT.
3. Message received cross-domain.
4. Message was PRI routed by ASSIGN command.
5. Message was SEC routed by ASSIGN command.
6. Message was COPY routed by ASSIGN command.
7. Message was routed to authorized receiver.
8. Message was from down-level domain (no AIFR received).
9. Message was unsolicited.

**Notes:**
1. Other bits can be tested, but have no recommended use. All the bits are defined in the DSIIFR mapping control blocks; refer to [Tivoli NetView for z/OS Customization: Using Assembler](http://www.ibm.com/support/docview.wss?uid=swg27047768).
2. Messages with the unsolicited flag on are eligible for ASSIGN PRI and SEC routing.
3. This attribute indicates the AIFR indicator fields IFRAUIND and IFRAUIN2.
4. When using extended multiple console support (EMCS) consoles, only MVS system messages that are received by the task with load module name CNMCSSIR are considered unsolicited messages.
5. For more information about solicited and unsolicited messages, refer to the [Tivoli NetView for z/OS Automation Guide](http://www.ibm.com/support/docview.wss?uid=swg27032662).

**IFRAUIN3**
Returns 1 byte of indicator bits as a series of 8 1 (on) and 0 (off) EBCDIC characters representing the bits in order. This data is mapped in DSIIFR. The following are the bit positions and their meaning:

- (Bits 1-2) Indicate cross-domain command priority as follows:
  - 00 = Default priority
  - 01 = Low priority
  - 10 = High priority
  - 11 = Test the receiver for priority


**CNMGETA (CNMGETATTR)**

- (Bit 3) VM PMX

**IFRAUI3X**
Retrieves the 32 bits of binary flags that are located in the DSIIFR map for a message, beginning at label IFRAUI3X. Browse the assembler macro, DSIIFR, for a description of these fields.

**IFRAUSB2**
Is a 2-byte user field from the message which is returned as a 2 characters.

**Note:** IFRAUSRB and IFRAUSB2 refer to the same user field in the message, but return the value in different formats.

**IFRAUSC2**
Is a 16-byte user field from the message which is returned as a series of 128, 1 (on) and 0 (off) characters representing the bits in order. See also IFRAUSRC.

**Note:** IFRAUSRC and IFRAUSC2 refer to the same user field in the message, but return the value in different formats.

**IFRAUSDR**
Is the 1- to 8- character name of the originating NetView task.

**IFRAUSRB**
Is a 2-byte user field from the message which is returned as a series of 16, 1 (on) and 0 (off) characters representing the bits in order. See also IFRAUSB2.

**Note:** IFRAUSRB and IFRAUSB2 refer to the same user field in the message, but return the value in different formats.

**IFRAUSRC**
Is a 16-byte user field from the message which is returned as 16 characters. See also IFRAUSC2.

**Note:** IFRAUSRC and IFRAUSC2 refer to the same user field in the message, but return the value in different formats.

**IFRAUTA1**
Returns 6-byte of indicator bits as a series of 48, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. IFRAUTA1 enables checking of control information. The bit positions are:

- (1,2,25) HOLD action.
- (5,6,26) SYSLOG action.
- (7,8,27) NETLOG action.
- (9,10,28) HCYLOG action.
- (11,12,29) DISPLAY action.
- (13,14,30) BEEP action.
- (20) Message from MVS.
- (23) VSE format message.
- (24) Action message.
- (47) Automation vector extensions exist.
- (48) Presentation vectors exist in data buffers.

**Notes:**
1. Other bits can be tested, but have no recommended use.
2. See the DSIIFR fields IFRAUTA1 thru IFRAUTA6 for more information.

3. For a description of all bits, refer to Tivoli NetView for z/OS Customization: Using Assembler.

**IFRAUWF1**

Is 4-bytes of MVS-specific WTO information returned as a series of 32, 1 (on) and 0 (off) characters representing the bits in order. Bit positions are:

- (6) Message is a WTOR
- (7) Message is suppressed
- (8) Broadcast to all
- (9) Display JOBNAMES
- (10) Display STATUS
- (14) Display SESSION

**Note:** Other bits can be tested, but have no recommended use.

**JOBNAME**

Is a 1- to 8- character MVS job name identifier. Because the JOBNAME is the name of the job that originated the message, it might not be the same as the name of the job to which the message is referring. For example, the job names might be different when MVS issues a message about the NetView job. Also, JOBNAME can contain the name of an initiator (instead of the job name) when a job is started or terminated. If the message is issued during startup or termination, extract the job name from the message text rather than using the JOBNAME function.

**Note:** The same information is available using MSGCOJBN.

**JOBNUM**

Is a 8-character MVS JOB number identifier.

**Note:** The MVS job identifier might contain embedded blanks. Depending on the MVS release, JOBNUM can be a character string such as JOB 4 which including blanks is eight characters.

**KEY**

Is an 8-character retrieval key associated with the message. KEY might contain nondisplayable values.

**Notes:**

1. All eight bytes must be specified.
2. To use the MVS DR, KEY command, specify only EBCDIC characters for KEY.
3. This function has a value only if the message currently being processed was originally an MDB.

**MCSFLAG**

Returns the system message flags as a series of 8, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. The bit positions are:

- (1) - Send message conditionally to console SYSCONID
- (2) - Send message unconditionally to console SYSCONID
- (3) - RESP
- (4) - REPLY
- (5) - BRDCST
- (6) - HDRCPY only
- (7) - NOTIME
- (8) - NOCPY
**CNMGETA (CNMGETATTR)**

*Note:* This function does not return the same mapping of multiple console support flags as the automation table compare item.

**MSGASID**

Is a MVS system address space identifier from which the message was issued. The value of MSGASID is a 1- to 5- digit decimal number.

*Note:* This value is null for messages that do not come from an MVS address space.

**MSGAUTH**

Is a 2-character value indicating whether the messages was issued from an authorized program. Bit positions and their meanings are:

- 00 - WTO message is not from MVS
- 10 - WTO is from an unauthorized program
- 11 - WTO is from an authorized program

**MSGCATTR**

Is 2-bytes of MVS message attribute flags returned as a series of 16, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. Bit positions and their meaning are:

- (1) - Message is suppressed.
- (2) - Message is the command response.
- (3) - Message issued by authorized program.
- (4) - Message is to be retained by Automation Message Retention Facility (AMRF).

*Notes:*

1. This function has a value only if the message currently being processed was originally an MDB.
2. Other bits can be tested, but have no recommended use.

**MSGCMISC**

Is 1 byte of MVS miscellaneous routing flags returned as a series of 8, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. Bit positions and their meaning are:

- (1) - Display UD (undeliverable) messages.
- (2) - Display only UD messages.
- (3) - Queue by ID only.
- (4) - Indicates whether the message has been marked in the message processing facility (MPF) table as eligible for NetView automation.

*Notes:*

1. This function has a value only if the message currently being processed was originally an MDB.
2. Other bits can be tested, but have no recommended use.

**MSGCM_LVL**

Is 2-bytes of MVS message level flags returned as a series of 16 on (1) and off (0) EBCDIC characters representing the bits in order. Bit positions and their meaning are:

- (1) - WTOR
- (2) - Immediate action
- (3) - Critical eventual action
- (4) - Eventual action
- (5) - Informational
- (6) - Broadcast
Notes:
1. This function has a value only if the message currently being processed was originally an MDB.
2. Other bits can be tested, but have no recommended use.

MSGCMSGT
Is 2 bytes of MVS message type flags returned as a series of 16 on (1) and off (1) EBCDIC characters representing the bits in order. Bit positions and their meaning are:
(1) Display job names
(2) Display status
(3) Monitor active
(6) Monitor SESS

Notes:
1. This function has a value only if the message currently being processed was originally an MDB.
2. Other bits can be tested, but have no recommended use.

MSGCOJBN
Is a 1- to 8- character returned originating job name.

Note: Also available in JOBNAME.

MSGCPROD
Is a 16-character MVS product level. The character are defined as follows:
• The first 4 characters represent an MVS control point object version level.
• The next 4 characters represent the control program name (MVS).
• The last 8 characters represent the function modification identifier (FMID) of the originating system.

Note: This function has a value only if the message currently being processed was originally an MDB.

MSGCSPLX
Is the 1- to 8-character name of the MVS SYSPLEX where the received message originated.

Note: This function, available when running under MVS/ESA Version 4 Release 3 or above, has a value only if the message currently being processed was received from an MVS SYSPLEX or from an MVS system that specified a sysplex name in a couple data set and was originally a MDB.

MSGCSYID
Is a 1- to 3- digit decimal number system identification for DOM. The value of MSGCSYID can be to 255.

Note: This function has a value only if the message currently being processed was originally an MDB.

MSGDOMFL
Is 1 byte of MVS DOM flags returned as a series of 8, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. Bit positions and their meaning are:
(1) DOM by message ID
CNMGETA (CNMGETATTR)

(2) DOM by system ID
(3) DOM by ASID
(4) DOM by job step TCB
(5) DOM by token

Notes:
1. This function has a value only if the message currently being processed was originally an MDB.
2. NetView EMCS consoles are set up by default as DOM(NORMAL) receivers. As a result, the DOMs that are received from MVS by these consoles are normally “DOM by MSGID,” and the TOKEN, SYSID, ASID, and TCB flags will usually not be set on when the DOM is received from MVS.

MSGGBGPA
Is 4 bytes of hexadecimal background presentation attributes. Bytes and their descriptions are:
- Byte 1 - Background control field
- Byte 2 - Background color field
- Byte 3 - Background highlighting field
- Byte 4 - Background intensity field

Note: This function has a value only if the message currently being processed was originally an MDB.

MSGGDATE
Is the message date in a seven-character format yyyyddd, where yyyy is the year and ddd indicates a calendar day.

Notes:
1. This is not necessarily the current date. It may be the date with which MVS associates the message as having been issued.
2. This function has a value only if the message currently being processed was originally an MDB.

MSGGFGPA
Is 4 bytes of hexadecimal foreground presentation attributes. Bytes and their meaning are:
- Byte 1 - Foreground control field
- Byte 2 - Foreground color field
- Byte 3 - Foreground highlighting field
- Byte 4 - Foreground intensity field

Note: This function has a value only if the message currently being processed was originally an MDB.

MSGGMFLG
Is 2 bytes of MVS general message flags returned as a series of 16, 1 (on) and 0 (off) EBCDIC characters representing the bits in order. Bit positions and their meaning are:
- Delete operator message (DOM)

Note: This function has a value only if the message currently being processed was originally an MDB.

MSGGMID
Is a 4-character MVS message identifier. The value of MSGGMID is a composite of MSGGSEQ and MSGGSYID. MSGGMID might contain
non-displayable characters. This field contains the same information as SMSGID, except that SMSGID is returned as a decimal number and MSGGMID is returned as hexadecimal value.

**Note:** This function has a value only if the message currently being processed was originally an MDB.

**MSGGSEQ**

Is a sequence number associated with the message. This value is a 1- to 8- digit decimal number, and is generated from the last 3-bytes of MSGGMID.

**Note:** This function has a value only if the message currently being processed was originally an MDB.

**MSGGSYID**

Is a ID of the system from which the message was issued. This value is a 1- to 3- digit decimal number generated from the first byte of MSGGMID.

**Note:** This function has a value only if the message currently being processed was originally an MDB.

**MSGGTIME**

Is a time that MVS associates with the message. An 11 character (including periods) time in the form hh:mm:ss:th, where hh is the hours, mm is the minutes, ss is the seconds, and th is tenths and hundredths of seconds.

**Note:** This function has a value only if the message currently being processed was originally an MDB.

**MSGSRCNM**

Is a 1- to 17- character source name. This source name is an identifier from the source object that was provided by either the DSIMMDBS or CNMPMDB API invocation. For more information about DSIMMDBS, refer to [Tivoli NetView for z/OS Customization: Using Assembler](#). For more information about CNMPMDB see “**CNMPMDB (CNMPRSMDB): Process Message Data Block**” on page 268. The source name is selected from the source object by the following rules:

- The first nickname, if any
- The first network identifier concatenated to a network addressable unit (NAU) name, with a period (.) between, if both exist in sequence
- The first NAU name, if it exists
- The string “N/A” if none of the other names in this list are specified in the source object
- Null, if there is no source object

For more information about how the source object is defined, refer to the DSIIFRO mapping in [Tivoli NetView for z/OS Customization: Using Assembler](#).

**Note:** This function has a value only if the message currently being processed was originally an MDB with an associated source object.
**CNMGETA (CNMGETATTR)**

**MSGSRCOB**
The entire source object which was provided on the DSIMMDBS or CNMPMDB application programming interface (API) invocation.

This value may contain non-displayable characters. The maximum length that will be returned is 1100 bytes. For more information about DSIMMDBS refer to Tivoli NetView for z/OS Customization: Using Assembler. For more information about CNMPMDB, see “CNMPMDB (CNMPRSMDB): Process Message Data Block” on page 268.

**MSGTOKEN**
Is a 1- to 10- digit decimal number that indicates the token associated with the message.

**Notes:**
1. This function has a value only if the message currently being processed was originally an MDB.
2. You can use a TOKEN value to group WTOs by setting MSGTOKEN prior to issuing the WTO command. Subsequently, these messages can be deleted using a single DOM command by specifying the token value in MSGTOKEN. Refer to Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for information about DOM token.

**MSGTSTMP**
Is a message time-stamp. The value of this field is the time when the NetView message buffer was created. The field is a 6-character string in the form hhmmss where:
- hh is hours
- mm is minutes
- ss is seconds

**MSGTYP**
The system message types flags returned as a series of 3, 1 (on) and 0 (off) EBCDIC characters.

The value of the characters are as follows:
- 1 SESS (corresponds to IFRAUWF1(14))
- 2 JOBNAMES (corresponds to IFRAUWF1(9))
- 3 STATUS (corresponds to IFRAUWF1(10))

**MVSRTAIN**
In NetView HLL procedures, a 3 bit field describing MVS Retain characteristics of the message.

**Note:** 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 in Extended Console Mode. The exact meaning and use of the flags is a property of the operating system. Refer to operating system documentation for specific information.

- 1xx AMRF retained message
- x1x Retain in AMRF
- xx1 Do not retain in AMRF

**NVDELID**
A NetView message deletion ID. A 24-character EBCDIC value for a
message that can be saved and used later as input to the DOM command to delete an action message.

**PARTID**
Is the first 2 characters of the 6-character prefix for VSE messages. The two returned characters will be the message partition ID only if the sending system uses those characters to designate a partition ID for a message.

*Note:* PARTID only has a value if the message originated on a VSE system.

**PRTY**
Is the priority message as set by the originator. This field is a 1- to 5-digit decimal number. NetView does not use this field when processing the message.

*Note:* This function has a value only if the message currently being processed was originally an MDB.

**REPLYID**
Is a reply identifier for WTORs. This field has a maximum length of 8 characters.

For messages from VSE systems, the REPLYID is the last three characters of the 6-character message prefix. The three returned characters will be the message reply ID only if the sending system uses those characters to designate a reply ID for a message.

**ROUTCDE**
Are the MVS routing codes assigned to the message. The value returned is a series of 1 (on) and 0 (off) EBCDIC characters representing the bytes in order. The maximum number of ROUTCDEs assigned to a message is 128.

**SESSID**
Is a 1- to 8- character ID of the terminal access facility (TAF) session that sent the message.

*Note:* If TAF session is started with a SESSID equal to the domain ID, SESSID is set unpredictably and may give unpredictable results.

**SMSGID**
Is a 1- to 10- character decimal number that identifies a particular instance of a message. This function can be used by the DOM command to identify action messages to be removed from the display.

This field contains the same information as MSGGMID, except that SMSGID is returned as a decimal number and MSGGMID is returned as a hexadecimal value.

**SYSCONID**
Is a MVS system console name or console ID associated with the message. System console names are 1- to 8- characters in length; system console IDs are 2-digit decimal numbers.

**SYSID**
Is a 1- to 8- character identifier of the MVS system that sent the message.

SYSID can be used in a sysplex to route commands to the appropriate system.
gaqueue
Is a 4-byte integer field containing the number of the queue holding the message. Only attributes for the initial data queue (IDATAQ) can be obtained.

hlbptr
Is a 4-byte pointer field containing the address of the HLB control block.

Usage Notes:
1. Other information, which exists in the ORIG BLOCK, is available when you call the CNMGETD service routine. See "CNMGETD (CNMGETDATA): Data Queue Manipulation" for more information.
2. Refer to Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for more information.
3. Some attributes apply to all types of messages, while others apply only to certain types of messages. For example, JOBNAME is meaningful only for messages received from MVS.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>The value requested is returned in gadata.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>gadatlen was too small. Data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect ganame.</td>
</tr>
<tr>
<td>CNM_BAD_QUEUE</td>
<td>72</td>
<td>Incorrect gaqueue value.</td>
</tr>
<tr>
<td>CNM_QUEUE_EMPTY</td>
<td>80</td>
<td>The specified queue is empty.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>gadatlen less than (&lt;) 0.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by gadata is not addressable.</td>
</tr>
</tbody>
</table>

CNMGETD (CNMGETDATA): Data Queue Manipulation

The CNMGETD service routine enables you to manipulate input queue data. Input queue data consists of logical groups of data buffers based on the type of information about the queue. The types of information that can be placed on the input queues are:

Single-line Message
Contains one data buffer for each logical group. The data buffer contains the text of the message. The logical group represents the actual message.

Multiline Message
Contains one or more data buffers for each logical group. Each data buffer contains the text from one line of the multiline message. The logical group represents the actual multiline message.

Automation MSU
Contains one or more data buffers for each logical group.

Reply MSU
Contains one or more data buffers for each logical group.

The CNMGETD routine syntax follows:

PL/I CALL FORMAT:
CALL CNMGETD(hlbptr, gdfunc, gdbuf, gdbuflen, gdorigin, gqueue, gdindex)
PL/I MACRO FORMAT:
CNMGETDATA FUNC(gdfunc) DATA(gdbuf) LEN(gdbuflen)
ORIGIN(gdorigin) QUEUE(gdqueue) LINE(gdindex)

C INVOCATION
void Cnmgetd(char *gdfunc, void *gdbuf, int gdbuflen, void *gdorigin,
int gdqueue, int gdindex)

Where:

*gdfunc
Is a varying length character field containing the data buffer to be returned.
This field is required for GETFIRST (GETMSG), GETNEXT (GETLINE), and
PEEKBF (PEEKLINE). It is not required for FLUSHBF (FLUSHLIN),
FLUSHGRP (FLUSHMSG), or FLUSHQ.

*gdbuf
Is a 4-byte integer field containing the length of *gdbuf. This is the maximum
length of the area provided to receive the returned data buffer. You provide the
value for *gdbuf.
This field is required for GETFIRST (GETMSG), GETNEXT (GETLINE), and
PEEKBF (PEEKLINE).

If the value specified by *gdbuf is less than the length of the data buffer to be
returned, the truncated data is returned in *gdbuf and a return code of
CNM_DATA_TRUNC is generated. The full length of the data buffer that is
truncated is stored in HLBLENG (Hlbleng).

Note: GETFIRST (GETMSG) and GETNEXT (GETLINE) requests continue to
advance through the queue independent of the truncation. Use a
combination of PEEKBF (PEEKLINE) with FLUSHBF (FLUSHLIN),
FLUSHGRP (FLUSHMSG), or FLUSHQ when retrieving complete data
buffers of unknown length.

If the value specified by *gdbuf is equal to or greater than the length of the
data buffer to be returned, and HLBRC (Hlbrc) = CNM_GOOD, the length of
the returned data buffer is stored in HLBLENG (Hlbleng).

If the value specified by *gdbuf is greater than the length of the receiving
data buffer (*gdbuf), a storage overlay could occur. Take special care when
deciding the value of *gdbuf.

*gdfunc
Is an 8-byte character field that specifies the function to be performed.
Additional field values are provided to support input queue data of any type.
The values support both message and non-message data. The existing
message-related values (GETMSG, GETLINE, and so on) are still supported.

FLUSHGRP (FLUSHMSG)
Skips to the next logical group in the specified queue.

FLUSHNXT (FLUSHLIN)
Skips over the next data buffer of the specified queue. This function
crosses logical group boundaries until the queue is empty.

FLUSHQ
Discards all logical groups in the specified queue.
Note: If this service is used in a command processor which is driven by automation, do not flush the initial data queue unless you have copied the command buffer string to a local variable because the command buffer is also flushed.

**GETFIRST (GETMSG)**
Returns the first data buffer of the next logical group of buffers on the specified queue. If one or more buffers of a group have already been returned by GETFIRST (GETMSG) or GETNEXT (GETLINE), the current GETFIRST (GETMSG) skips to the next logical group in the queue, discarding any skipped data buffers.

**GETNEXT (GETLINE)**
Returns the next buffer in the specified queue. This function crosses logical group boundaries until the queue is empty. You must check ORIG_LINE_TYPE to determine which line is last.

**PEEKBFR (PEEKLINE)**
Returns a data buffer from the logical group at the head of the specified queue whose buffer number is specified by \textit{gdindex}. Flushed and received data buffers can still be obtained unless the logical group is flushed or a logical buffer in a subsequent logical group was obtained.

\textit{gdindex}
Is a 4-byte integer field containing the number (index) of the group at the head of the queue to be manipulated.

This field is required only for PEEKBFR (PEEKLINE).

\textit{gdorigin}
Is a character field of fixed length \( n \) (where \( n \geq 38 \)) to contain an origin block. You must define an origin block (\textit{gdorigin}) to be passed as an operand to CNMGETD. This must be a separate structure from the origin block (ORIGBLCK) that was passed to the HLL command processor or installation exit routine as an initial parameter. ORIG_BLOCK_LENGTH cannot be less than 38. See "DSIPORIG Control Block" on page 306 and "DSICORIG Control Block" on page 333 for the PL/I and C mappings of an origin block.

This field is required for GETFIRST (GETMSG), GETNEXT (GETLINE), and PEEKBFR (PEEKLINE).

\textit{gdqueue}
Is a 4-byte integer field containing the number (index) of the queue on which to perform the operation. This field is required for all functions. Valid values follow:

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Queue Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAPQ</td>
<td>1</td>
<td>Message queue. Contains trapped messages. See &quot;TRAP Command&quot; on page 218</td>
</tr>
<tr>
<td>OPERQ</td>
<td>2</td>
<td>Operator input queue. See &quot;GO Command&quot; on page 213 and &quot;QUEUE Command&quot; on page 214</td>
</tr>
<tr>
<td>DATAQ</td>
<td>3</td>
<td>Data queue. Contains data sent from another HLL command processor or installation exit routine. See &quot;CNMSMSG (CNMSENDMSG): Send Message or Command&quot; on page 282</td>
</tr>
</tbody>
</table>
### Queue Name | Queue Number | Function
--- | --- | ---
IDATAQ | 4 | Initial data queue. Contains the full message or MSU that invokes the HLL command processor through NetView automation. It also contains messages that drive DSIEX02A. This is also the queue where an application command processor receives an MDS-MU from the NetView high performance transport, the MS transport, or operations management for an unsolicited request or asynchronous reply.
 CNMIQ | 5 | CNMI solicited data queue. Contains RUs solicited through the CNMI service routine. Chained RUs are treated like multiline messages. See [“CNMACNMI (CNMI): CNMI Access Under a DST” on page 224](#).
 MDSMUQ | 6 | MDS-MU data queue. Contains message units (MUs) received as synchronous replies. The MDS-MUs are received from operations management, the MS transport, and the high performance transport.

### hlbp

*hlbp*  
Is a 4-byte pointer field containing the address of the HLB control block.

#### Usage Notes:
1. The following items are available after issuing CNMGETD. The NetView command list language equivalents are listed in the descriptions.

   **ORIG_MSG_TYPE**  
   Provides the 1-character NetView buffer type of the received message or MSU, and is equivalent to &HDRMTYPE. This control variable returns a X’10’ when an MSU buffer string is being handled.

   **ORIG_LINE_TYPE**  
   Provides the MLWTO line type and is equivalent to &LINETYPE. This control variable returns the character M for the MSU buffer and an H for the HIER buffer.

   **ORIG_PROCESS**  
   Is the message identifier of the message currently being processed by NetView and is equivalent to &MSGID. This control variable returns a null value when an MSU buffer string is being handled.

   **ORIG_DOMAIN**  
   Is the domain where the message most recently received by NetView originated and is equivalent to &MSGORIGIN.

2. See [“DSIPORIG Control Block” on page 306](#) and [“DSICORIG Control Block” on page 333](#) for mapping of the origin block.

#### Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>gdbuflen was too small. Data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect gdfunc value.</td>
</tr>
<tr>
<td>CNM_BAD_QUEUE</td>
<td>72</td>
<td>Incorrect gdqueue value.</td>
</tr>
<tr>
<td>CNM_BAD_INDEX</td>
<td>76</td>
<td>Incorrect gdlindex value.</td>
</tr>
<tr>
<td>CNM_QUEUE_EMPTY</td>
<td>80</td>
<td>The specified queue is empty.</td>
</tr>
</tbody>
</table>
**CNMHRGS (CNMHREGIST): High Performance Transport Application Registration**

The CNMHRGS service routine registers any application that wants to send data to or receive data from another application through the high performance transport API. You must register the application before attempting to send or receive data.

CNMHRGS 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.

The CNMHRGS routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMHRGS(hlbptr,hrtype,hrappl,hrcmd,hrlogmod,hrrepl,hrnotify,hrpri)
```

**PL/I MACRO FORMAT:**

```pli
CNMHREGIST TYPE(hrtype) APPL(hrappl) COMMAND(hrcmd)
LOGMODE(hrlogmod) REPLACE(hrrepl) NOTIFY(hrnotify)
PRI(hrpri)
```

**C INVOCATION:**

```c
void Cnmhrgs(char *hrtype, char *hrappl, char *hrcmd,
char *hrlogmod, char *hrrepl char *hrnotify,
char *hrpri)
```

**Where:**

- `hlbptr`
  - Is a 4-byte pointer field containing the address of the HLB control block.

- `hrappl`
  - Is an 8-byte character field that specifies the application to be registered or deregistered.

The identifier name can be one of:

- An architecturally defined 4-byte value (padded with blanks to 8 bytes) for high performance application programs.

The following names are reserved by NetView and cannot be specified in a CNMHRGS invocation:

- **ALERT** X'23F0F3F1'
- **EP_OPS** X'23F0F1F6'
- **EP_SPCS** X'23F0F1F4'
- **LINKSERV** X'23F0F3F5'
- **MDS_RECEIVE** X'23F0F0F1'
- **MDS_ROUTER** X'23F0F1F0'
- **MS_CAPS** X'23F0F1F7'
- **OPS_MGMT** X'23F0F1F7'

---

**Return Code Name** | **Value** | **Description**
--- | --- | ---
CNM_BAD_LENGTH | 88 | `gdbuflen` less than (<) 0.
CNM_BAD_ADDR | 160 | The storage pointed to by either `gdbuf` or `gdorigin` is not addressable.
CNM_NO_LINES_IN_MSG | 264 | All buffers in the specified logical group have been deleted.
**R_BRIDGE**  X'30F0F7F9'
**RTMCMRD_O**  X'30F0F7F2'
**RTMCMRD_R**  X'30F0F5F5'
**RTMCMRD_S**  X'30F0F7F0'
**SPCS**  X'23F0F1F5'
**No character equivalent**  X'30F0F7F3'

- A 1–8 character installation-defined name (padded with blanks). You must use the EBCDIC characters 0–9 and A-Z (capitals only).

The name STATUS is reserved for the NetView status focal point and is not allowed on a CNMHRGS invocation.

**hrcmd**

Is an 8-byte character field that specifies the command processor that is started when unsolicited or asynchronous data is routed to the application. The NetView program verifies that the task is authorized to issue the command specified on hrcmd. This field is required for REGAPPL.

**hrlogmod**

Is an 8-byte character field that specifies the logmode that is used for sending the application data. This name must be a logmode that is defined to the local VTAM and the receiving LU or command processor (CP) with which this application communicates.

**Note:** **hrlogmod** is used only on the initial registration request for an application. You cannot change a registered application’s logmode unless you deregister the application.

**hrnotify**

Is an 8-byte character field that specifies whether the MS or operations management served application receives session outage notification for LUs in contact with the LU 6.2 sessions.

- **ALL**  Indicates that the application receives an MDS-MU containing an SNA condition report with sense data of xxxxxxxxxx every time the last SNASVCMMG session has been lost. This notification is received even if the session outage is not related to an error.

- **ERROR**  Indicates that the application receives an MDS-MU containing an SNA condition report with sense data of xxxxxxxxxx every time the last SNASVCMMG session has been lost because of session failure.

- **NONE**  Indicates that the application does not receive session outage notification. NONE is the default for the PL/I macro format.

If you do not specify the NOTIFY keyword when using the PL/I macro format, the default value is used. Otherwise, this field is required.

**hrpri**

Is an 8-byte character field that specifies the MQS priority for incoming requests. The MQS priority is used when the high performance transport uses the MQS for processing any unsolicited MDS-MUs.

- **HIGH**  Processing begins after any NORMAL requests currently in progress completes, but before queued NORMAL or LOW requests.

- **LOW**  Processing is preempted by HIGH and NORMAL priority requests. This is the default.
CNMRGS (CNMHREGIST)

NORMAL    Processing preempts a queue of LOW priority requests.
TEST      CNMRGS queues the request based on the command priority of the receiving task. The command priority can be set using the OVERRIDE or DEFAULT commands. Refer to the Tivoli NetView for z/OS Command Reference for more information.

hrrepl
Is a 4-byte character field that specifies whether this registration is to supersede any previous registration for this application.
NO      Specifies that this registration should not replace the current registration for this application.
YES    Specifies that this registration replaces the current registration for this application. YES is the default for the PL/I macro format.

This is a required field for REGAPPL. However, if you do not specify the REPLACE keyword in the PL/I macro format, the default value is used.

hrtype
Is a 10-byte character field that specifies the type of request:
DEREGAPPL       Deregisters an application from the high performance transport.
REGAPPL          Registers an application to the high performance transport.

Usage Notes:
1. If you specify a logmode that is not defined in the logmode table, VTAM defaults the logmode to the first entry in the logmode table.
2. 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.
   • For an MDS request, the receiving task is the task from which CNMRGS is started 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 was running.
       - For an incoming request, the receiving task is the task under which the receiving application was running.
     - If the AUOWC does not match an active AUOWC, the receiving task is the task from which CNMRGS is started for the receiving application.
3. You can change the task under which CNMRGS was started by registering the application from the desired task and specifying YES for hrrepl.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Successful registration/deregistration.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>Deregistration unsuccessful. hrappl not registered.</td>
</tr>
</tbody>
</table>
CNMHSMU (CNMHSENDMU): Send High Performance Message Unit

The CNMHSMU service routine enables NetView applications to send data to a specified target through the high performance transport API. The high performance transport uses an LU 6.2 conversation, and VTAM selects the appropriate session for the actual transmission. You can invoke CNMHSMU only in applications registered through CNMHRGS or the REGISTER command.

The data is sent in the form of an MDS-MU. You can supply:

- A completely built MDS-MU
- An MDS-MU that is missing one or more of:
  - A unit of work correlator (UOWC)
  - An origin NETID
  - An origin LUNAME
- 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 CNMHSMU service routine builds the necessary NetView MQS buffer with the specified data and queues it to the high performance transport.

The CNMHSMU routine syntax follows:

PL/I CALL FORMAT:

```
CALL CNMHSMU(hlbptr,hsdtype,hsdata,hsupcor,hsccorr, hstimout,hssynch,hsrplcmd, hsoappl,hsdtnet,hsdstlu,hsdstapl,hsmutype,hspri)
```
CNHSMU (CNMHSENDMU)

PL/I MACRO FORMAT:

```
CNMHSENDMU DATATYPE(hsdtype) DATA(hsdata) SUPPCORREL(hssupcor)
CORRELAREA(hscorrar) TIMEOUT(hstimout) SYNC(hssynch)
REPLYCMD(hsrplcmd) ORIGAPPL(hsoappl)
DESTNET(hsdstnet) DESTLU(hsdstlu) DESTAPPL(hsdstapl)
MUTYPE(hsmutype) PRI(hspri)
```

C INVOCATION:

```c
void Cnmhsmu(char *hsdtype, void *hsdata, void *hssupcor,
void *hscorrar, int hstimout, char *hssynch,
char *hsrplcmd, char *hsoappl, char *hsdstnet,
char *hsdstlu, char *hsdstapl, int hsmutype,
char *hspri)
```

where:

*hlbptr*  
Is a 4-byte pointer field containing the address of the HLB control block.

*hscorrar*  
Is a 53-byte character field in which a new unit of work correlator (X'1549') GDS variable is created and returned by the CNHSMU service routine.

If you specify *hscorrar* for an MDSMU, 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. If you omit *hscorrar*, the MDS-MU must be complete and ready to be transmitted as supplied.

For a NONMDSMU you must specify either *hscorrar* or *hssupcor*. If you specify *hscorrar*, CNHSMU creates the unit of work correlator GDS variable in this area and uses it in building the MDS header. If you specify both *hssupcor* and *hscorrar*, *hssupcor* is used.

*hsdata*  
Is a varying length character field containing the data being sent. For either MDSMU or NONMDSMU the first 2 bytes must contain the entire length of the data and the next 2 bytes must contain the key.

*hsdstapl*  
Is an 8-byte character field that specifies the destination high performance application name.

The application name can be one of:

- 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). You must use the EBCDIC characters 0–9 and A-Z (capitals only).
- A 1–8 character NetView-reserved name (padded with blanks) that represents an architecturally defined 4-byte value. The NetView-defined names and their corresponding values follow:

  - **ALERT** X'23F0F3F1'
  - **EP_ALERT** X'23F0F3F0'
  - **EP_OPS** X'23F0F1F6'
  - **MS_CAPS** X'23F0F1F1'
  - **OPS_MGMT** X'23F0F1F7'

  This field is required for NONMDSMU.

*hsdstlu*  
Is an 8-byte character field that specifies the LU name of the destination LU.
You must specify the 1–8 character LU name (padded with blanks to 8 characters) beginning with an EBCDIC character 0–9 and A-Z (capitals only), @, #, or $, and followed by EBCDIC characters 0–9 and A-Z (capitals only).

This field is required for NONMDSMU.

hsdstnet
Is an 8-byte character field that specifies the ID of the network of the destination LU. You must specify the 1-8 character NETID (padded with blanks to 8 characters) beginning with an EBCDIC character, @, #, or $, and followed by EBCDIC characters 0–9 and A-Z (capitals only). The value of this field defaults to the network name that VTAM determines based on the LU name of the remote node (specified with the hsdstlu field) if you:
• Specify blanks for this field for the PL/I call format.
• Do not specify the DESTNET keyword when using the PL/I macro format.
• Specify blanks for this field for the C invocation format.

hsdtype
Is an 8-byte character field indicating whether the data item specified with the hsdata field is an MDS-MU or a non-MDS-MU.

MDSMU
Indicates that the hsdata is an MDS-MU. MDSMU is the default for the PL/I macro format.

NONMDSMU
Indicates that the hsdata is not a complete MDS-MU because it does not contain an MDS-MU header. The CNMHSMU service routine envelopes this data in an MDS-MU header before sending it.

If you do not specify the DATATYPE keyword when using the PL/I macro format, the default value is used.

hsmutype
Is a 4-byte integer field that specifies the index number that identifies the type of MDS-MU to build. The type identifies whether the MDS-MU is a request, a reply, or an error message, and whether additional messages are expected. The following types are defined as constants:
1 REQUEST_WITH_REPLY
2 REQUEST WITHOUT_REPLY
3 REPLY ONLY
4 REPLY NOTLAST
5 REPLY LAST
6 ERROR MESSAGE

This is a required keyword for NONMDSMU.

hsoappl
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 high performance application programs.
• A 1-8 character installation-defined name (padded with blanks). You must use the EBCDIC characters 0–9 and A-Z (capitals only).

This field is required for NONMDSMU.
CNMHSMU (CNMHSEN DMU)

hspri
Is an 8-byte character field that specifies the MQS priority for incoming solicited requests or any MDS error messages resulting from any outgoing MDS-MUs. The MQS priority is used when the high performance transport uses the MQS for processing any solicited MDS-MUs or any MDS error messages.

**HIGH**  Processing begins after any NORMAL requests currently in progress completes, but before queued NORMAL or LOW requests.

**LOW**  Processing is preempted by HIGH and NORMAL priority requests. This is the default for all requests other than synchronous requests.

**NORMAL**  Processing preempts a queue of LOW priority requests. This is the default for synchronous requests.

**TEST**  CNMHSMU queues the request based on command priority of the receiving task. The command priority can be set using the OVERRIDE or DEFAULT commands. Refer to the `Tivoli NetView for z/OS Command Reference` for more information.

hsrplcmd
Is an 8-byte character field containing the name of the command to be driven with the reply. The `hsrplcmd` field is used only in an application that is sending REQUEST_WITH_REPLY with the reply being received asynchronously. Otherwise, it is ignored.

This is an optional field. The default is the registered command for the invoking application.

hssupcor
Is a varying length character field containing a complete unit of work correlator (X’1549’) GDS variable. The `hssupcor` 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.

`hssupcor` is not valid for an MDS-MU. For a NONMDSMU, you must specify either `hssupcor` or `hscorrar`. If you specify `hssupcor`, the supplied value is used to build the MDS header. No validity checking is done for a correlator supplied by the invoker.

hssynch
Is an 8-byte character field that specifies whether the high performance application is to receive the reply synchronously.

**NO**  Indicates that the reply is received asynchronously. NO is the default for the PL/I macro format.

**NO_BUF**  Do not suspend the application but buffer replies until the last is received. This value is equivalent to the NO value.

**NO_UNBUF**  Do not suspend the application and forward replies immediately.

**YES**  Indicates that the reply is received synchronously.

**YES_BUF**  Suspend the application and buffer its replies. This value is equivalent to the YES value.
If you do not specify the SYNCH keyword when using the PL/I macro format for REQUEST_WITH_REPLY, the default value is used. Otherwise, the field is required for REQUEST_WITH_REPLY.

**hstimout**

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. You can change these values with the DEFAULTS command.

The valid values for **hstimout** are:

- 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 PL/I macro format for REQUEST_WITH_REPLY, the default timeout value is used. Otherwise, this field is required for REQUEST_WITH_REPLY.

**Usage Notes:**

1. For a synchronous REQUEST_WITH_REPLY, control is returned to the invoking program after the last reply, or an error message is received and placed on the MDSMUQ data queue. Otherwise, control is returned after CNMHSMU successfully queues the request to the high performance transport.
2. When the invoking program is suspended because of a synchronous REQUEST_WITH_REPLY, the NetView task where the program is running is not suspended. The task still receives and processes messages and commands.
3. For a synchronous REQUEST_WITH_REPLY from a DST, a DSRB is marked in-use and the DSRB is not available for other use until the suspended program is started again.
4. For MDSMU, all fields within the MDS-MU header must be correct except for origin NETID and LUNAME. The service routine can determine and set these fields. If the correlator is not contained in the data, you must specify **hscorrar**.
5. For **REPLY_ONLY**, **REPLY_NOTLAST**, **REPLY_LAST**, and **ERROR_MESSAGE**, you must specify **hssupcor** to return the correlator sent with the request.
6. The high performance transport implements a timeout value for the application receiving the data. If the invocation of CNMHSMU 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.
7. When VTAM is active, you can use CNMHSMU to send data to another application in the same domain.
8. If **hsdstnet** is not the NETID determined by VTAM for the LU specified in **hsdstlu**, the send fails.
9. A high performance application cannot send data to itself within the same NetView.
10. A return code 24 or 28 from DSIPUSH indicates that DSIOLGFP is not defined correctly in DSICMD.
11. If CNM_BAD_CES is returned:
   • Verify that DSI6SNDP is defined correctly in DSICMD.
   • If you specify hrrepcmd, verify that it is defined correctly in DSICMD.
   • If you specify a synchronous REQUEST_WITH_REPLY, verify that
     DSIOSRCP is defined correctly in DSICMD.

12. Refer to the Tivoli NetView for z/OS Administration Reference for the correct
    definitions of the NetView supplied command processors.

13. For MDSMU, if you omit the NETID subfield of the destination subvector
    from the MDS-MU header, VTAM determines the network name used, based
    on the LU name in the NAU name subfield of the destination subvector.

14. If you do not specify the destination NETID, and the destination LU name
    exists in more than one network, VTAM determines the destination NETID
    based on the active configuration.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>Task is terminating, TVBABEND/TVBLOGOF is on.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>No storage is available.</td>
</tr>
<tr>
<td>CNM_NOT_INASYNCH</td>
<td>44</td>
<td>Send MU service is started from an installation exit.</td>
</tr>
<tr>
<td>CNM_BAD_TIMEOUT</td>
<td>56</td>
<td>Timeout value is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>MDS-MU length is not valid.</td>
</tr>
<tr>
<td>CNM_TASK_INACTIVE</td>
<td>220</td>
<td>DSIOHPDST not active.</td>
</tr>
<tr>
<td>CNM_BAD_DATA_TYPE</td>
<td>400</td>
<td>Data type is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DATA</td>
<td>404</td>
<td>Data missing or is not valid.</td>
</tr>
<tr>
<td>CNMSAME_APPL</td>
<td>408</td>
<td>Application cannot send data to the same application within the same NetView program.</td>
</tr>
<tr>
<td>CNM_SYNH_NOT_COMP</td>
<td>412</td>
<td>SYNH(YES) is not allowed under a NetView installation exit or the PPT.</td>
</tr>
<tr>
<td>CNM_OAPPL_NOT_REG</td>
<td>416</td>
<td>Application is not registered.</td>
</tr>
<tr>
<td>CNM_BAD_UOW</td>
<td>424</td>
<td>UOW is missing or not valid.</td>
</tr>
<tr>
<td>CNM_BAD_OAPPL</td>
<td>440</td>
<td>Origin application name is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DNETID</td>
<td>444</td>
<td>Destination network ID missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DLU</td>
<td>448</td>
<td>Destination LU name is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DAPPL</td>
<td>452</td>
<td>Destination application name is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_REPLY</td>
<td>460</td>
<td>Reply is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_MUTYPE</td>
<td>464</td>
<td>Bad MUTYPE given.</td>
</tr>
<tr>
<td>CNM_BAD_SYNCH</td>
<td>468</td>
<td>Bad SYNCH option.</td>
</tr>
<tr>
<td>CNM_BUSY</td>
<td>472</td>
<td>User list is full.</td>
</tr>
<tr>
<td>CNM_SYNH_CMD_MISSING</td>
<td>508</td>
<td>SYNH(YES) is specified but DSIOSRCP or DSIOLOGFP is not defined or is not defined correctly in DSICMD.</td>
</tr>
</tbody>
</table>
### CNMINFC (CNMINFOC): Query NetView Character Information

The CNMINFC service routine allows you to obtain information about the current NetView environment. CNMINFC returns character data.

The CNMINFC routine syntax follows:

**PL/I CALL FORMAT:**

```plaintext
CALL CNMINFC (hlbptr,icname,icdata,icdatlen)
```

**PL/I MACRO FORMAT:**

```plaintext
CNMINFOC ITEM(icname) DATA(icdata) LENG(icdatlen)
```

**C INVOCATION:**

```c
void Cnminfc(char *icname, void *icdata, int icdatlen)
```

**Where:**

- **hlbptr**
  Is a 4-byte pointer field containing the address of the HLB control block.

- **icdata**
  Is a varying length character field containing the character data to be returned.

- **icdatlen**
  Is a 4-byte integer field containing the length of *icdata*. This is the maximum length of the area provided to receive the returned data. You provide the value of *icdatlen*. This field must be greater than 0 and less than 32729.

If the value specified by *icdatlen* is less than the length of the data to be returned, the truncated data is returned in *icdata* and a return code of CNM_DATA_TRUNC is generated. The full length of the data that was truncated is stored in HLBLENG (Hlbleng).

If the value specified by *icdatlen* is equal to or greater than the length of the data to be returned, and HLBRC (Hlbrc) = CNM_GOOD, the length of the returned data is stored in HLBLENG (Hlbleng).
If the value specified by `icdatlen` is greater than the length of the receiving data buffer (`icdata`), a storage overlay could occur.

**iname**

Is an 8-byte character field that specifies the name of a variable. Valid names are:

**APPLID**

Is the NetView domain ID appended with a 3-character alphanumeric value assigned by the NetView program.

**AUTCONID**

Is the MVS console identifier associated with this autotask. This association was made using the `AUTOTASK` command with the console keyword. This value of `AUTCONID` is the console name or console ID of the MVS console where NetView commands can be entered to run under this autotask.

*Note:* MVS console names are available beginning with MVS 4.1.

**CLOCK**

Is the current value returned by `STCK` instruction (not displayable).

**CURCONID**

Is the MVS console obtained by a NetView task. This console was obtained with the `GETCONID` command or by issuing an MVS command. The value of `CURCONID` is the console name or console ID of the MVS console that this task uses to enter MVS commands.

*Note:* MVS console names are available on systems running MVS 4.1 or later.

**CURSYS**

Is a 1- to 8-character name of the current operating system.

**DATE**

Is the current date in the form of `mm/dd/yy`, where `mm` is the month, `dd` is the day, and `yy` is the year.

**DATETIME**

Equivalent to `&DATE` followed by `&TIME`.

**DATETIM2**

Is the date and time in the `YYYYMMDD HH:MM:SS` format.

**DOMAIN**

Is the 1- to 5-character name of the current NetView domain.

**HCOPY**

Is the name of the device defined as the hard-copy log printer started by the operator. If there is no device defined as the hard-copy printer for this operator, `HCOPY` is null.

**LU**

Is the logical unit name for the operator terminal.

**MVSLEVEL**

Is the currently running version of MVS. For example, if you are running MVS/ESA 4.2.2, `MVSLEVEL` returns the value `SP4.2.2`.

*Note:* The value of `MVSLEVEL` is null if the currently running system is not MVS.

**NETID**

Is the VTAM network identifier. This field has a maximum length of 8 characters.

*Note:* If VTAM has never been active while NetView is active, the value of `NETID` is null.
NVVER  Is the version and release of the currently running NetView program. The value of NVVER is a 4-character string in the form of NVvr, where:
  NV  Indicates NetView.
  v  Indicates the version number of NetView.
  r  Indicates the release number of NetView.

OPID  Is the operator or task ID that issued the call to CNMINFC. OPID is a 1- to 8-character identifier.

OPSYSTEM  Is the operating system for which NetView was compiled, and the HLL command processor or installation exit routine is running. OPSYSTEM must be MVS/ESA.

PID  The process ID for this HLL command processor or installation exit routine. Used for CNMSMSG with smmsgtyp=DATA (not displayable).

STARTIME  The NetView start time (not displayable).

SUPPCHAR  Is the 1-character NetView suppression character.

SYSPLEX  Is the 1- to 8-character name of the MVS SYSPLEX where the command list is executing.

Note: This function, available when running under MVS/ESA Version 4 Release 2.2 or above, has a value only if the command list is executing on an MVS SYSPLEX or on an MVS system that specified a sysplex name in a couple data set.

TASK  Is a character string indicating the type of task under which the command processor or installation exit routine is running. Possible values follow:
  DST  Data Services Task
  HCT  Hardcopy Task
  MNT  Main Task
  NNT  NetView-to-NetView Task
  OPT  Optional Task
  OST  Operator Station Task
  PPT  Primary POI Task
  UNKNOWN  None of the above.

TASKNAME  The name of the task under which the HLL command processor or installation exit routine is running.

TIME  Is the CPU time in the form of hh:mm, where hh is the hour and mm is the minutes. The time is based on a 24-hour clock, so 3:00pm is shown as 15:00.

VTAM  Is the version and release of VTAM is a 4-character string in the form of either VTvr or Vvmr, where:
  VT  Indicates that the access method is the VTAM program.
  v  Indicates the version number
  r  Indicates the release number
  m  Indicates the modification number

Note: The value of the VTAM is null if the VTAM program is not active.
VTCOMPID  Is a 14-character VTAM component identifier. The VTAM component identification can be:

MVS/ESA
5685-08501-xxx

Where: xxx is the release number.

Additional VTAM component identifiers may be added in future updates to VTAM.

Note: The value of VTCOMPID is null if VTAM is not active.

YEAR  Is the 4-digit year.

Usage Notes:
1. Refer to Tivoli NetView for z/OS Customization: Using REXX and the NetView Command List Language for a complete description of the NetView command list language variables.
2. CLOCK, PID, and STARTIME are 8-character representations of the TOD-clock (time-of-day) value returned by the STCK instruction.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>The value requested is returned in icdata.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>icdatlen was too small. Data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect icname. Value unchanged.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>icdatlen less than (&lt;) 0.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by icdata is not addressable.</td>
</tr>
</tbody>
</table>

CNMINFI (CNMINFOI): Query NetView Integer Information
The CNMINFI routine enables you to obtain information about the current NetView environment. CNMINFI returns integer data.

The CNMINFI routine syntax follows:

PL/I CALL FORMAT:
CALL CNMINFI(hlbptr,iiname,iinumb)

PL/I MACRO FORMAT:
CNMINFI ITEM(iiname) DATA(iinumb)

C INVOCATION:
void Cnminfi(char *iiname, int *iinumb)

Where:

hlbptr
Is a 4-byte pointer field containing the address of the HLB control block.
**iname**

Is an 8-byte character field that specifies the name of the variable. Valid names are:

- **ABENDRTN**
  - If processing as a long-running command ABEND routine, then return true.

- **ASID**
  - Is the NetView’s current address space identifier. The value of ASID is a 1-to 5-digit decimal number.

- **ATTENDED**
  - Is a single character variable with a value of 1 or 0.
  - The values for ATTENDED are:
    1. The task is one of the following:
      - An OST with a display
      - An NNT with a corresponding OST
      - An autotask with an associated MVS console assigned using the AUTOTASK command
      - A distributed autotask
    0. The task is one of the following:
      - An autotask without an associated MVS console assigned using the AUTOTASK command
      - Another type of task, such as a DST or an OPT task

**Notes:**

1. If the associated operator is an autotask, the presentation data will not be eligible for display unless the autotask is associated with an active MVS console.

2. ATTENDED can be used in conjunction with DISTAUTO and AUTOTASK variables to further define the characteristics of the task. For example, if ATTENDED is 1, DISTAUTO is 0 (zero), and AUTOTASK is 1 then the task is an autotask with an associated MVS console.

- **AUTOTASK**
  - The single character value of either 1 or 0 indicating whether or not a task is an autotask. The values follow:
    1. Indicates that the task is an autotask
    0. Indicates that the task is not an asterisk

  These *iname* contain Boolean values (0 = false and 1 = true).

- **AWAITINP**
  - If waiting for operator input, then return true. From DSITVB NetView control block.

- **CLOSING**
  - If NetView is terminating, then return true.

- **COLORS**
  - Is the number of colors that can be displayed.

---

1. These *iname* contain Boolean values (0 = false and 1 = true).
DISTAUTO
Denotes if task is a distributed autotask started with the RMTCMD command. The values are as follows:
1 Indicates the task is a distributed autotask
0 Indicates the task is not a distributed autotask

LOGOFRTN³
If processing as a long-running command routine, then return true.

MVTUFLD
Is the user field from the DSIMVT NetView control block (MVTUFLD).

OPER3270³
If an OST with a 327x display terminal is attached, then return true.

RESETREQ³
If RESET or CANCEL was requested, then return true. NetView’s internal RESET flag is turned off as a result of this query. See “RESET Command” on page 215 for further details.

SCRNSER
Is the return serial number of the screen update.

TIBUFLD
Is the user field from the DSITIB NetView control block (TIBUFLD).

TVBUFLD
Is the user field from the DSITVB NetView control block (TVBUFLD).

USEREXIT
If the integer value is 0, the environment is that of a command processor.
If the integer value is 2–17, the environment is that of an installation exit.
If the integer value is one of the following, the environment is that of an installation exit running under a DST:
233 USERDINT
DSM initialization exit
234 USERVINT
VSAM initialization exit
235 USERVINP
VSAM input exit
236 USERVOUT
VSAM output exit
237 USERCINP
CNMI input exit
238 USERCOUT
CNMI output exit
240 USERXLOG
External log exit
241 USERBINT
Sequential log initialization exit
242 USERBOUT
Sequential log output exit

See DSIPCONS in "Appendix A. PL/I Control Blocks and Include Files" on page 301 or DSICCONS in "Appendix C. C Language Control Blocks and Include Files" on page 327 for a list of constants useful when coding installation exit routines.
WEEKDAYN
Is a numeric value from 1 to 7 indicating the day of the week (from Monday through Sunday). These are the values that will be returned:
1 = Monday
2 = Tuesday
3 = Wednesday
4 = Thursday
5 = Friday
6 = Saturday
7 = Sunday

\( iinumb \)
Is a 4-byte integer field containing the integer value returned.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>The value requested is returned in ( iinumb ).</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect ( iiname ). Value unchanged.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by ( iinumb ) is not addressable.</td>
</tr>
</tbody>
</table>

CNMKIO (CNMKEYIO): Keyed File Access Under a DST

The CNMKIO service routine provides access to DST managed key-sequenced VSAM files from an HLL command processor. This service routine performs its function only when started from an HLL command processor running under a DST. Calls from other environments are rejected.

The CNMKIO routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMKIO(hlbptr,vsfunc,vsdata,vsdatlen,vskey,vsoption)
```

**PL/I MACRO FORMAT:**

```pli
CNMKEYIO FUNC(vsfunc) DATA(vsdata) LENG(vsdatlen)
KEY(vskey) OPTIONS(vsoption)
```

**C INVOCATION:**

```c
void Cnmkio(char *vsfunc, void *vsdata, int vsdatlen, void *vskey, char *vsoption)
```

**Where:**

- **hlbptr**
  Is a 4-byte pointer field containing the address of the HLB control block.

- **vsdata**
  Is a varying length character field containing the buffer to be returned or written. You provide this field for PUT. This field is returned for GET and is not required for ERASE or ENDREQ.

- **vsdatlen**
  Is a 4-byte integer field containing the length of \( vsdata \). This is the maximum length of the area provided to receive the returned data. You provide the value of \( vsdatlen \). This field is required only for GET.
If the value specified by `vsdatlen` is less than the length of the data to be returned, the truncated data is returned in `vsdata` and a return code of CNM_DATA_TRUNC is generated. The full length of the data that was truncated is stored in HLBLENG (Hlbleng).

If the value specified by `vsdatlen` is equal to or greater than the length of the data to be returned, and HLBRC (Hlbrc) = CNM_GOOD, the length of the returned data is stored in HLBLENG (Hlbleng).

If the value specified by `vsdatlen` is greater than the length of the receiving data buffer (`vsdata`), a storage overlay could occur. Take special care when deciding the value of `vsdatlen`.

`vsfunc`  
Is an 8-byte character field that specifies the function to performed.  
ENDREQ  Cancels a request for update.  
ERASE    Erases the record.  
GET_EH   Gets a record equal to or higher than the key.  
GET_EQ   Gets a record equal to the key.  

Note: The key field must match exactly, including blanks.  
GET_NEXT Gets the next record in ascending sequence.  
GET_PREV Gets the next record in descending sequence.  
PUT      Writes or rewrites the record.

`vskey`  
A varying length character field containing the VSAM key used for access to the requested data. This field is required for GET_EQ, GET_EH, or ERASE/DIRECT.

`vsoption`  
Is an 8-byte character field that specifies the type of access to the file. You provide this field for all functions except ENREQ.  

DIRECT  
Put a new record directly to the file or erase a record directly from the file (without invoking GET first). You can use PUT/DIRECT only for a new record. If the record already exists, CNM_DUPL_KEY is returned. For an existing record, ERASE/DIRECT gives the same result as GET/UPDATE followed by ERASE/UPDATE. If the record does not exist, CNM_NOT_FOUND is returned. You can use DIRECT only with PUT and ERASE.

NOUPDATE  
The record is not updated. You can use NOUPDATE only with GET.

UPDATE  
Get a record for update or replace. Erase the record that was gotten for update. PUT/UPDATE and ERASE/UPDATE must be preceded by a successful GET/UPDATE. You can use UPDATE only with GET, PUT, and ERASE.

Usage Notes:  
1. You cannot issue CNMKIO from an HLL installation exit routine or from an HLL command processor while holding a lock.  
2. For more information about installing a DST, see "Chapter 3. HLL Data Services Command Processors" on page 27.
Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>Not started from a command processor.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>Record with requested vskey not found.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_END_FILE</td>
<td>36</td>
<td>End of file encountered.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>vsdatlen was too small. Data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect vsfunc.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>vsdatlen less than (&lt;) 0.</td>
</tr>
<tr>
<td>CNM_BAD_OPTION</td>
<td>128</td>
<td>Incorrect vsoption.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by vsdata is not addressable.</td>
</tr>
<tr>
<td>CNM_BAD_COMBO</td>
<td>176</td>
<td>Incorrect combination vsfunc and vsoption.</td>
</tr>
<tr>
<td>CNM_DUPL_KEY</td>
<td>200</td>
<td>Record with requested key already exists. Existing record is not changed.</td>
</tr>
<tr>
<td>CNM_LOCKED</td>
<td>208</td>
<td>CNMKIO issued while holding a lock.</td>
</tr>
<tr>
<td>(CNM_BAD_ZVSMS + X) * 256 + Y</td>
<td>256 + Y</td>
<td>Nonzero return code from DSIZVSMS. X is the major return code from DSIZVSMS. Y is the minor return code from DSIZVSMS.</td>
</tr>
<tr>
<td>CNM_DST_FAILURE + X</td>
<td>2000 + X</td>
<td>Nonzero return code, X, which is the DSRB minor return code for solicited CNMI data or VSAM data set services.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) or the VSAM library for more information.

**CNMLK (CNMLOCK): Controlling a Lock**

You can use the CNMLK service routine to obtain, release, and test the control of a named lock. You can use this service to serialize access to resources shared by multiple tasks. CNMLK does not allow for serialization within a task. HLL command processors holding a lock cannot use any services that can suspend execution of an HLL command processor.

The CNMLK routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMLK(hlbptr, lkfunc, lkname, lkscope, lkoption)
```

**PL/I MACRO FORMAT:**

```pli
CNMLOCK FUNC(lkfunc) NAME(lkname) SCOPE(lkscope) OPTION(lkoption)
```

**C INVOCATION:**

```c
void Cnmlk(char *lkfunc, void *lkname, char *lkscope, char *lkoption)
```

**Where:**

*hlbptr* is a 4-byte pointer field containing the address of the HLB control block.
**CNMLK (CNMLOCK)**

*lkfunc*

Is an 8-byte character field that specifies the function to be performed:

- **LOCK** Obtains control of the lock name.
- **TEST** Tests if the lock name is available.
- **UNLOCK** Releases control of the lock name.

*lkname*

Is a varying length character field to hold the user-defined name of the lock. (Length is 1–12 characters.) This field is required for all functions.

*lkoption*

Is an 8-byte character field that specifies whether the HLL command processor or installation exit routine should wait for the lock to become available. This field is required only for **LOCK**. Possible values follow:

- **NOWAIT**
  Do not wait if the lock is not available. An appropriate return code is returned (CNM_LOCKED or CNM_LOCK_INUSE).

- **WAIT**
  Wait until the lock is available. The task is suspended.

*lkscope*

Is an 8-byte character field reserved for future use. Provide a null or blank value in this field for all functions.

**Usage Notes:**

1. Do not invoke CNMSMSG with `smdestyp` = OPER while holding a lock. The operator task can be running with autowrap off, and the HLL command processor or installation exit routine might hang waiting for the operator to clear the screen, thus holding the lock for an indefinite period.

2. A hierarchical order on lock requests is used to prevent deadlock. The alphabetical order of the lock names defines the hierarchy. Therefore, lock requests for lock names alphabetically less than or equal to a presently held lock name fail, and return code CNM_LOCKED is generated. For example, assume the last lock request was for `lkname` = GVARIABLE. A new lock request for `lkname` = TVARIABLE is successful because TVARIABLE is alphabetically greater than GVARIABLE. However, a new lock request for `lkname` = CVARIABLE is unsuccessful because CVARIABLE is alphabetically less than or equal to GVARIABLE. Return code CNM_LOCKED is generated and the lock request fails.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect <code>lkfunc</code>.</td>
</tr>
<tr>
<td>CNM_BAD_NAME</td>
<td>108</td>
<td>Length of <code>lkname</code> greater than (&gt;) 12 characters or the specified <code>lkname</code> was not locked.</td>
</tr>
<tr>
<td>CNM_BAD_OPTION</td>
<td>128</td>
<td>Incorrect <code>lkoption</code>.</td>
</tr>
<tr>
<td>CNM_LOCKED</td>
<td>208</td>
<td>The <code>lkname</code> is unavailable because it is alphabetically less than or equal to the lock name of the last lock request.</td>
</tr>
</tbody>
</table>
Return Code Name | Value | Description
---|---|---
CNM_LOCK_INUSE | 212 | lkname is not available; currently held by another task.
CNM_BAD_ENQ + X | 21000 + X | Nonzero return code X from DSIENQ macro.

Refer to [Tivoli NetView for z/OS: Customization: Using Assembler](#) or the MVS/XA library for more information.

### CNMMEMC (CNMCLOSMEM): Close NetView Partitioned Data Set

The CNMMEMC service routine enables you to close a member of a NetView partitioned data set that was previously opened by CNMMEMO. The token returned by CNMMEMO must be passed to CNMMEMC to allow the file to be closed. All members opened by CNMMEMO are automatically closed at program termination.

The CNMMEMC routine syntax follows:

**PL/I CALL FORMAT:**

CALL CNMMEMC(hlbptr,mctoken)

**PL/I MACRO FORMAT:**

CNMCLOSMEM TOKEN(mctoken)

**C INVOCATION:**

void Cnmmemc(int mctoken)

Where:

- **hlbptr**
  - Is a 4-byte pointer field containing the address of the HLB control block.
- **mctoken**
  - Is a 4-byte integer field containing the token returned by CNMMEMO.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_TOKEN</td>
<td>32</td>
<td>Token not found.</td>
</tr>
<tr>
<td>CNM_BAD_DKS + X</td>
<td>10000 + X</td>
<td>Nonzero return code, X, from DSIDKS macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS: Customization: Using Assembler](#) for more information.

### CNMMEMO (CNMOPENMEM): Open NetView Partitioned Data Set

The CNMMEMO service routine enables you to open members of NetView partitioned data sets. A token identifying the open member is returned to you. This token is passed to CNMMEMR to read records from the member and to CNMMEMC to close the member.
The CNMMEMO routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMMEMO(hlbptr, motoken, moddname, momemnam)
```

**PL/I MACRO FORMAT:**

```pli
CNMOPENMEM TOKEN(motoken) DATASET(moddname) MEMBER(momemnam)
```

**C INVOCATION:**

```c
void Cnmmemo(int *motoken, char *moddname, char *momemnam)
```

**Where:**

- **hlbptr**
  - Is a 4-byte pointer field containing the address of the HLB control block.

- **moddname**
  - Is an 8-byte character field that specifies the DD name of the partitioned data set. The NetView predefined DD names are:
    - BNJPNL1
    - BNJPNL2
    - CNMPNL1
    - DSICLD
    - DSILIST
    - DSIMSG
    - DSIOOPEN
    - DSIPARM
    - DSIPRF
    - DSIVTAM

- **momemnam**
  - Is an 8-byte character field that specifies the name of the member.

- **motoken**
  - Is a 4-byte integer field containing the token to be used by subsequent CNMMEMR and CNMMEMC requests.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_DDNAME</td>
<td>16</td>
<td>Incorrect <code>moddname</code> (not found).</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td><code>momemnam</code> not found.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by <code>motoken</code> is not addressable.</td>
</tr>
<tr>
<td>CNM_BAD_DKS + X</td>
<td>10000 + X</td>
<td>Nonzero return code, X, from DSIDKS macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.
The CNMMEMR service routine enables you to read a record from a member of a NetView partitioned data set that was previously opened by CNMMEMO. The token returned by CNMMEMO must be passed to CNMMEMR to allow reading.

The CNMMEMR routine syntax follows:

**PL/I CALL FORMAT:**

CALL CNMMEMR(hlbptr,mrtoken,mrdata,mrdatlen,mrinclude)

**PL/I MACRO FORMAT:**

CNMREADMEM TOKEN(mrtoken) DATA(mrdata) LENG(mrdatlen) INCL(mrinclude)

**C INVOCATION:**

void Cnmmemr(int mrtoken, void *mrdata, int mrdatlen, char *mrinclude)

Where:

- **hlbptr**
  - Is a 4-byte pointer field containing the address of the HLB control block.

- **mrdata**
  - Is a varying length character field to contain the received record.

- **mrdatlen**
  - Is a 4-byte integer field containing the length of **mrdata**. This is the maximum length of the area provided to receive the returned record. You provide the value of **mrdatlen**.

  - If the value specified by **mrdatlen** is less than the length of the record to be returned, the truncated record is returned in **mrdata** and a return code of CNM_DATA_TRUNC is generated. The full length of the truncated record is stored in HLBLENG (Hlbleng).

  - If the value specified by **mrdatlen** is equal to or greater than the length of the record to be returned, and HLBRC (hlbrc) = CNM_GOOD, the length of the returned record is stored in HLBLENG (Hlbleng).

  - If the value specified by **mrdatlen** is greater than the length of the receiving data buffer (mrdata), a storage overlay could occur.

- **mrinclude**
  - Is a 6-byte character field that specifies whether INCLUDE cards are to be processed:
    - **INCL** Specifies to process INCLUDE cards.
    - **NOINCL** Specifies that INCLUDE cards are not processed.

- **mrtoken**
  - Is a 4-byte integer field containing the token returned by CNMMEMO.

**Usage Note:** If the return code indicates that there was a problem with an INCLUDE card or the member specified by the include card (CNM_SYNTAX_ERROR, CNMINVAL_MEMBER, CNMINVAL_NEST, CNM_SYSTEM_ERROR), the record returned in **mrdata** is the INCLUDE card that is not valid.
CNMMEMR (CNMREADMEM)

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_IOERROR</td>
<td>28</td>
<td>I/O error occurred.</td>
</tr>
<tr>
<td>CNM_BAD_TOKEN</td>
<td>32</td>
<td>Token not found.</td>
</tr>
<tr>
<td>CNM_END_FILE</td>
<td>36</td>
<td>End of file encountered.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>mrdatlen was too small. Data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>mrdatlen less than (&lt;) 0.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by mrdata is not addressable.</td>
</tr>
<tr>
<td>CNM_SYNTAX_ERROR</td>
<td>520</td>
<td>Syntax error in INCLUDE card.</td>
</tr>
<tr>
<td>CNM_INVAL_MEMBER</td>
<td>524</td>
<td>Incorrect member specified in INCLUDE card.</td>
</tr>
<tr>
<td>CNM_INVAL_NEST</td>
<td>528</td>
<td>Incorrect nesting of INCLUDE cards.</td>
</tr>
<tr>
<td>CNM_SYSTEM_ERROR</td>
<td>532</td>
<td>Internal NetView system error while processing INCLUDE card.</td>
</tr>
<tr>
<td>CNM_BAD_INCLUDE</td>
<td>536</td>
<td>Incorrect value for mrinclude.</td>
</tr>
<tr>
<td>CNM_BAD_DKS + X</td>
<td>10000 + X</td>
<td>Nonzero return code, X, from DSIDKS macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

CNMNAMS (CNMNAMESTR): Named Storage

The CNMNAMS routine allows you to allocate, free, locate, and reallocate named areas of virtual storage.

The CNMNAMS routine syntax follows:

**PL/I CALL FORMAT:**

```pli
call cnmnams(hlbptr, nsfunc, nsptr, nsname, nsleng, nsclass)
```

**PL/I MACRO FORMAT:**

```
cmnnamestr func(nsfunc) strptr(nsptr) name(nsname) leng(nsleng) class(nsclass)
```

**C INVOCATION:**

```c
void cnmnams(char *nsfunc, void *nsptr, void *nsname, int *nsleng, int nsclass)
```

Where:

- `hlbptr` is a 4-byte pointer field containing the address of the HLB control block.
- `nsclass` is a 4-byte integer field containing the class of the named storage area, as follows:
  - 0 = Residency of caller
  - 1 = 31-bit storage
  - 2 = 24-bit storage
This field is required by the caller for ALLOC and REALLOC. This field is not required for FREE or LOCATE.

**nsfunc**

Is an 8-byte character field that specifies the function to be performed:

- **ALLOC** Allocated the named storage area.
- **FREE** Frees or deallocates the named storage area.
- **LOCATE** Locates the existing named storage area.
- **REALLOC** Reallocates the named storage area. Old data is preserved.

**nslnge**

Is a 4-byte integer field containing the size of the named storage area. This field is required by the caller for ALLOC and REALLOC and is returned to the caller for LOCATE. This field is not required for FREE.

**nsname**

Is a varying length character field containing the name of the storage area. This field is required for all functions and is provided by the caller.

**nsptr**

Is a 4-byte pointer field containing the address of the named storage. This field is required by the caller for ALLOC, REALLOC, and LOCATE. This field is not required for FREE.

**Usage Notes:**

1. The named storage areas provide a way of sharing data among different HLL command processors and installation exit routines or among multiple invocations of an HLL command processor or installation exit routine. Once allocated, a named storage area remains allocated until it is either explicitly freed or the task under which it was allocated terminates.

2. A named storage area is associated with the NetView subtask under which it was allocated. Named storage areas can be manipulated (using LOCATE, FREE, or REALLOCATE) only by HLL command processors and installation exit routines running under the mainline of that task. You cannot reference a named storage area from a task other than the one with which it is associated.

3. If ALLOC is requested for a name that has already been allocated, the address of the existing area is returned along with a nonzero return code.

4. If a previously allocated named storage area is reallocated to be larger than the original area, the content of the original area is preserved. If a previously allocated named storage area is reallocated to be smaller than the original area, the content of the original area is truncated at the length specified by the **nslnge** operand.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>REALLOC, FREE, or LOCATE requested but no previous ALLOC was done.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect function.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td><strong>nslnge</strong> less than (&lt;) 0.</td>
</tr>
<tr>
<td>CNM_DUPL_NAME</td>
<td>104</td>
<td><strong>nsname</strong> already allocated. Allocation not done.</td>
</tr>
<tr>
<td>CNM_BAD_NAME</td>
<td>108</td>
<td>Length of <strong>nsname</strong> greater than (&gt;) 12 characters.</td>
</tr>
</tbody>
</table>
CNMAMS (CNMNAMESTR)

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_BAD_CLASS</td>
<td>112</td>
<td>Incorrect nsclass.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by nsname or nsleng is not addressable.</td>
</tr>
<tr>
<td>CNM_BAD_PUSH + X</td>
<td>4000 + X</td>
<td>Nonzero return code, X, from DSIPUSH macro.</td>
</tr>
<tr>
<td>CNM_BAD_POP+X</td>
<td>5000 + X</td>
<td>Nonzero return code, X, from DSIPOP macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMPMDB (CNMPRSMDB): Process Message Data Block**

The CNMPMDB service routine requests the NetView message processor to process a message data block (MDB) and its companion source object.

CNMPMDB accepts an MDB and a source object, transforms them into an automation internal function request (AIFR), and sends the AIFR through normal NetView message processing including invoking the automation table. The transformation of the MDB and source object into an AIFR includes establishing the values for the AIFR variables, such as MSGSRCNM and MSGSRCOB.

The CNMPMDB routine syntax follows:

**PL/I CALL FORMAT:**

```pli
call cnmpmdb(hlbptr,mdmdb,mdsource,mdcorr)
```

**PL/I MACRO FORMAT:**

```pli
cnmprsmdb mdb(mmdb) source(mdsoure) correl(mdcorr)
```

**C INVOCATION:**

```c
void Cnmpmdb(void *mdmdb, void *mdsource, void *mdcorr)
```

**Where:**

- **hlbptr**
  - Is a 4-byte pointer field containing the address of the HLB control block.

- **mdcorr**
  - Is a 16-character correlator value used for chaining together multiple related MDBs.

You must pass this parameter as all binary zeros for CNMPMDB invocations for single (non-chained) MDBs.

For multiple related MDBs, pass all binary zeros for the **mdcorr** parameter on the first CNMPMDB invocation.

CNMPMDB will return a correlator value in **mdcorr**. On subsequent calls to CNMPMDB for the multiple MDBs, pass the correlator value which was returned in **mdcorr** on the first invocation of CNMPMDB.

When an MDB is sent with an end of text indicator, the **mdcorr** will be returned with a zero value.
mdmdb
Is a 4-byte pointer field containing the address of an MDB. The maximum length of an MDB is 4 KB bytes. The maximum length of multiple related MDBs is 63 KB.

mdsource
Is a 4-byte pointer field containing the address of the source object. This is an optional field. If you do not wish to specify a source object, pass zero in mdsource.

Usage Notes:
1. The CNMPMDB service routine returns control to the invoking program after it has scanned the automation table and done the synchronous portion of message processing.
2. The service does not free the MDB or source object.
3. If you create DOM MDBs, keep in mind that not all forms of DOM can be transported over OST-NNT sessions. If you are using any OST-NNT sessions, it is recommended that you only create DOM MDBs which indicate DOM by MSGID.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>The MDB is accepted for processing.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Storage is not available.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by mdcorr is not addressable.</td>
</tr>
<tr>
<td>CNM_BAD_SEQUENCE</td>
<td>564</td>
<td>Third operand is not a valid correlator value.</td>
</tr>
<tr>
<td>CNM_BAD_MDB</td>
<td>568</td>
<td>First operand is not valid for message data block (MDB).</td>
</tr>
<tr>
<td>CNM_BAD_SOURCE_Obj</td>
<td>572</td>
<td>Second operand is not a valid source object.</td>
</tr>
</tbody>
</table>

CNMPOOL (CNMSTRPOOL): Storage Pool
You can use the CNMPOOL service routine to allocate, free, and locate storage pools. A storage pool is composed of one primary and zero or more secondary blocks of storage. Each storage block has a specified number of cells (of equal size) that can be allocated or freed using CNMCELL. Storage pool services provide a way to effectively manage large numbers of fixed size storage elements.

The CNMPOOL routine syntax follows:

**PL/I CALL FORMAT:**
CALL CNMPOOL(hlbptr,spfunc,sptoken,spname,spleng,sppricnt,spseccnt,spclass)

**PL/I MACRO FORMAT:**
CNMSTRPOOL FUNC(spfunc) TOKEN(sptoken) NAME(spname) LENGS SPLENG PRICELLS(sppricnt) SECCELLS(spseccnt) CLASS(spclass)

**C INVOCATION:**
void Cnmpool(char *spfunc, int *sptoken, void *spname, int spleng, int sppricnt, int spseccnt, int spclass)
CNMPOOL (CNMSTRPOOL)

Where:

hlbptr
Is a 4-byte pointer field containing the address of the HLB control block.

spclass
Is a 4-byte integer field containing the storage class of the pool:
- 0 = Residency of caller
- 1 = 31-bit addressable
- 2 = 24-bit addressable

This field is required for ALLOC, but not for FREE or LOCATE.

spfunc
Is an 8-byte character field that specifies the function to be performed:
- ALLOC Allocates the pool
- FREE Frees the pool
- LOCATE Locates the pool

spleng
Is a 4-byte integer field containing the size of each cell in the pool. This field is required for ALLOC, but not for FREE or LOCATE.

spname
Is a varying length character field containing the name of the storage pool. This field is required for all functions and provided by the caller.

sppricnt
Is a 4-byte integer field containing the number of cells in the primary block. This field is required only for ALLOC.

spsecnt
Is a 4-byte integer field containing the number of cells in the secondary block. This field is required only for ALLOC.

sptoken
Is a 4-byte integer field to contain the token identifying the storage pool. This field is returned for ALLOC and LOCATE for use with CNMCELL service. This field is not required for FREE.

Usage Notes:
1. A storage cell within a pool is associated with the NetView subtask under which it was allocated. It cannot be referenced from a task other than the one with which it is associated.
2. All storage pool names must be unique within a given task.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>spname not found.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect spfunc.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>spleng less than (&lt;) 4.</td>
</tr>
<tr>
<td>CNM_DUPL_NAME</td>
<td>104</td>
<td>spname already allocated. Allocation not done.</td>
</tr>
<tr>
<td>CNM_BAD_NAME</td>
<td>108</td>
<td>Length of spname greater than (&gt; 12.</td>
</tr>
<tr>
<td>CNM_BAD_CLASS</td>
<td>112</td>
<td>Incorrect spclass.</td>
</tr>
</tbody>
</table>
Return Code | Name            | Value | Description                                      |
-------------|-----------------|-------|-------------------------------------------------|
CNM_BAD_ADDR | 160             |       | The storage pointed to by sptoken is not addressable. |
CNM_BAD_PRI_COUNT  | 192          |       | Incorrect spprint.                               |
CNM_BAD_SEC_COUNT  | 196          |       | Incorrect spsecnt.                              |

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMQAPI (CNMOPREP): Resource Object Data Manager**

The CNMQAPI service routine enables interaction with a specified resource object data manager (RODM). All RODM API functions are supported through this interface, including querying for data, changing data, and triggering methods.

**Note:** CNMQAPI 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.

The CNMQAPI routine syntax follows:

**PL/I CALL FORMAT:**

```pli
CALL CNMQAPI(hlbptr,qaacb,qatif,qaresp,qafunc,qawaitf,qawaitt)
```

**PL/I MACRO FORMAT:**

```pli
CNMOPREP ACB(qaacb) TIF(qatif) RESPONSE(qaresp)
FUNCTION(qafunc) WAITF(qawaitf) WAITT(qawaitt)
```

**C INVOCATION:**

```c
void Cnmqapi (char *qaacb, char *qatif, char *qaresp,
              char *qafunc, char *qawaitf, int *qawaitt)
```

Where:

- **hlbptr**
  - Is a 4-byte field containing the address of the HLB control block.

- **qaacb**
  - Is a RODM access block following the format of the RODM API. For more information about RODM, refer to the [Tivoli NetView for z/OS Resource Object Data Manager and GMEHS Programmer’s Guide](#).

The access block contains these 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. Refer to the [Tivoli NetView for z/OS Administration Reference](#) for a description of the DSIQTSKI keywords. The name is left-justified and must be padded with blanks (X’40’) to 8 characters.

- **signon_token**
  - Specifies the RODM signon token to be used within the call.
CNMQAPI (CNMOPREP)

CNMQAPI ignores this field and fills it with the sign-on token received by DSIQTSK when it initially connects to the RODM being accessed. Refer to Tivoli NetView for z/OS Administration Reference for more information.

user_appl_id

This field specifies the application name of the caller.

CNMQAPI 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. Refer to the Tivoli NetView for z/OS Administration Reference for a description of the DSIQTSKI keywords.

qafunc

Is 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. Refer to the Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide for a description of the function block format.

qaresp

Is a response block following the format of the RODM API response block control structure. Refer to Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide for information about the RODM response block.

qatif

Is a transaction information block following the format of the RODM API transaction information block. Refer to Tivoli NetView for z/OS Resource Object Data Manager and GMFHS Programmer’s Guide for a description of the RODM API transaction information block.

qawaitf

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. Valid values follow:

N  Do not wait for checkpoint completion. N is the default.
Y  Wait for checkpoint completion.

qawait

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 started. The valid value range is 10—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.

Refer to Tivoli NetView for z/OS Administration Reference for more information.

Usage Notes:

1. An application can connect to RODM with an option specifying that RODM can truncate its responses if the application response block is smaller than the RODM 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 uses this option, saving CNMQAPI from having to deal with overflow cleanup.

2. CNMQAPI returns two sets of return codes.
The caller can use either of them. 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 and GMFHS Programmer’s Guide.

3. The second set is in the HLB. The HLBRC field of the HLB contains a return code upon return from CNMQAPI.

4. APF-authorized command processors can access RODM without a password. Provide a keyword for command processors that can be authority-checked to control access to RODM.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Successful.</td>
</tr>
<tr>
<td>CNM_NO_PASSWORD</td>
<td>8</td>
<td>A command processor tried to access RODM without specifying a password but was either not APF-authorized or the APF-authorization code was not 1.</td>
</tr>
<tr>
<td>CNM_NO_REPOS</td>
<td>604</td>
<td>RODM not under control of RODM access and control component.</td>
</tr>
<tr>
<td>CNM_API_FAILURE</td>
<td>608</td>
<td>CNMQAPI failure. Internal macro call failure—could be storage.</td>
</tr>
<tr>
<td>CNM_INVAL_PARMS</td>
<td>612</td>
<td>Incorrect parameters received.</td>
</tr>
<tr>
<td>CNM_NO_STOR</td>
<td>616</td>
<td>No storage.</td>
</tr>
<tr>
<td>CNM_TIMEOUT</td>
<td>620</td>
<td>Checkpoint in progress.</td>
</tr>
<tr>
<td>CNM_BAD_OPR</td>
<td>25000 + X</td>
<td>Nonzero return code, X, from DSIOPR macro.</td>
</tr>
</tbody>
</table>

Refer to Tivoli NetView for z/OS Customization: Using Assembler for more information.

**CNMRGS (CNMREGIST): Application Registration**

The CNMRGS service routine enables registration and deregistration of any MS application with the NetView MS transport, or any operations management served application with the NetView operations management application. In addition, MS applications can register as focal points while both MS applications and operations management served applications can register as capable of receiving focal point information. Refer to the Tivoli NetView for z/OS Application Programmer’s Guide for more information about management services and remote operations.

The CNMRGS routine syntax follows:

**PL/I CALL FORMAT:**

```
CALL CNMRGS(hlbptr,rgtype,rgappl,rgcmd,rgfpcat,rgfocpt,rgrepl,rgnotify,rgpri)
```

**PL/I MACRO FORMAT:**

```
CNMREGIST TYPE(rgtype) APPL(rgappl) COMMAND(rgcmd)
FPSCATEGORY(rgfpcat) FOCALPOINT(rgfocpt)
REPLACE(rgrepl) NOTIFY(rgnotify) PRI(rgpri)
```

**C INVOCATION:**
### CNMRGS (CNMREGIST)

```c
void Cnmrgs(char *rgtype, char *rgappl, char *rgcmd, char *rgfpcat,
char *rgfocpt, char *rgrepl, char *rgnotify, char *rgpri)
```

**Where:**

- **hlbptr**
  - Is a 4-byte pointer field containing the address of the HLB control block.

- **rgappl**
  - Is an 8-byte character field that specifies the MS application or operations management served application being registered or deregistered.

  The application name can be one of:

  - An architecturally defined 4-byte value (padded with blanks to 8 bytes) for MS application programs.

  The following names are reserved by NetView and cannot be specified in a CNMRGS invocation:

<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>MS_CAPS</td>
<td>X'23F0F1F1'</td>
</tr>
<tr>
<td>MDS_ROUT</td>
<td>X'23F0F1F0'</td>
</tr>
<tr>
<td>OPS_MGMT</td>
<td>X'23F0F1F7'</td>
</tr>
<tr>
<td>RTMTCMD_R</td>
<td>X'30F0F5F5'</td>
</tr>
<tr>
<td>RTMTCMD_S</td>
<td>X'30F0F7F0'</td>
</tr>
<tr>
<td>RTMTCMD_O</td>
<td>X'30F0F7F2'</td>
</tr>
<tr>
<td>R_BRIDGE</td>
<td>X'30F0F5F9'</td>
</tr>
<tr>
<td>SPCS</td>
<td>X'23F0F1F5'</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>

  - 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 name STATUS is reserved for the NetView status focal point and is not allowed on a CNMRGS invocation.

- **rgcmd**
  - Is an 8-byte character field that specifies the command procedure to be started when unsolicited or asynchronous solicited data is routed to the application.

  The NetView program verifies that the task has the authority to issue the command specified. This field is required for REGSMAPPL and REGOMSERVD.

- **rgfocpt**
  - Is an 8-byte character field that specifies whether the MS application is a focal point application:

    **NO**  The MS application is not a focal point application. NO is the default for the PL/I macro format.

    **YES** The MS application is a focal point application.

  If you do not specify the FOCALPOINT keyword when using the PL/I macro format for REGMSAPPL, the default value is used. Otherwise, this field is required for REGMSAPPL.
**rgfpcat**

Is an 8-byte character field that specifies the name of an application registered as a focal point application. This is the focal point category from which you want to receive information.

**Note:** NetView supplies three focal point applications, ALERT (X'23F0F3F1'), OPS_MGMT (X'23F0F1F7') and SPCS (X'23F0F1F5'). For REGOMSERVD, you can specify only OPS_MGMT for **rgfpcat**. For more information about the focal points, refer to the *Tivoli NetView for z/OS Automation Guide*.

**rgnotify**

Is an 8-byte character field that specifies whether the MS or operations management served application receives session outage notification for LUs in contact with the MS transport.

**ALL** Indicates that the application receives an MDS-MU containing an SNA condition report with sense data of xxxxxxxxxxxx every time the last SNASVCMG session has been lost. This notification is received even if the session outage is not related to an error.

**ERROR** Indicates that the application receives an MDS-MU containing an SNA condition report with sense data of xxxxxxxxxxxx every time the last SNASVCMG session has been lost because of a session failure.

**NONE** Indicates that the application does not receive session outage notification. NONE is the default for the PL/I macro format.

If you do not specify the NOTIFY keyword when using the PL/I macro format, the default value is used. Otherwise, this field is required.

**rgpri**

Is an 8-byte character field that specifies the MQS priority for incoming requests. The MQS priority is used when the MS transport uses the MQS for processing any unsolicited MDS-MUs.

**HIGH** Processing begins after any NORMAL requests currently in progress completes, but before queued NORMAL or LOW requests.

**LOW** Processing is preempted by HIGH and NORMAL priority requests. This is the default.

**NORMAL**

Processing preempts a queue of LOW priority requests.

**TEST** CNMRGS queues the request based on the command priority of the receiving task. The command priority can be set using the OVERRIDE or DEFAULT commands. Refer to the NetView online help.

**rgrepl**

Is an 8-byte character field that specifies whether this registration is to supersede the previous registration for this application.

**NO** Specifies that this registration does not replace the current registration for this application.

**YES** Specifies that this registration replaces the current registration for this application. YES is the default for the PL/I macro format.
**CNMRGS (CNMREGIST)**

If you do not specify the REPLACE keyword when using the PL/I macro format for REGMSAPPL or REGOMSERVD, the default value is used. Otherwise, this field is required for REGMSAPPL and REGOMSERVD.

**rgtype**
- Is a 12-byte character field that specifies the type of request, as follows:
  - **DEREGMSAPPL**
    - Deregisters an MS application from the NetView MS transport.
  - **DEREGOMSERVD**
    - Deregisters an operations management served application from
  - **REGMSAPPL**
    - Registers an MS application to the NetView MS transport.
  - **REGOMSERVD**
    - Registers a second-level application to operations management.

**Usage Notes:**
1. You can register an application as both an MS application and an operations management served application.
2. When you specify that an MS application is a focal point, the focal point category name is the application name specified in **rgappl**.
3. Any registered application that was once a focal point application continues to receive focal point data from any application that did not attempt to send data while it was deregistered. To recognize that an application is no longer a focal point, the sending application must attempt to send data to a focal point application that has been deregistered.
4. NetView determines the task where an application receives an MDS-MU as follows:
   - For an MDS reply, the receiving task is the task under which the requesting application is running.
   - For an MDS request, the receiving task is the task from which CNMRGS is started for the receiving application.
   - For an MDS error message:
     - If the 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 CNMRGS is started for the receiving application.
5. You can change the task under which CNMRGS was started by reregistering the application from the desired task and specifying YES for **rgrepl**.

**Return Codes:**

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Successful registration/deregistration.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td>Deregistration unsuccessful. <strong>rgappl</strong> not registered.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
</tbody>
</table>
CNMRGS (CNMREGIST)

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_NOT_IN_ASYNC</td>
<td>44</td>
<td>Deregistration unsuccessful. Issued from an exit.</td>
</tr>
<tr>
<td>CNM_DUPL_NAME</td>
<td>104</td>
<td>Registration unsuccessful. <em>rgappl</em> already registered.</td>
</tr>
<tr>
<td>CNM_BUSY</td>
<td>472</td>
<td>Registration/deregistration unsuccessful. Queues are in use.</td>
</tr>
<tr>
<td>CNM_BAD_APPL_NAME</td>
<td>476</td>
<td>Registration/deregistration unsuccessful. <em>rgappl</em> syntax incorrect.</td>
</tr>
<tr>
<td>CNM_APPL_NAME_RSTD</td>
<td>480</td>
<td>Registration/deregistration unsuccessful. <em>rgappl</em> restricted.</td>
</tr>
<tr>
<td>CNM_BAD_FPCAT_NAME</td>
<td>484</td>
<td>Registration/deregistration unsuccessful. <em>rgfpcat</em> syntax incorrect.</td>
</tr>
<tr>
<td>CNM_BAD_FPCAT_CHOICE</td>
<td>488</td>
<td>Registration unsuccessful. <em>rgfpcat</em> has an incompatible value.</td>
</tr>
<tr>
<td>CNM_BAD_REG_TYPE</td>
<td>492</td>
<td>Incorrect <em>rgtype</em>.</td>
</tr>
<tr>
<td>CNM_BAD_FOCALPT_VALUE</td>
<td>496</td>
<td>Incorrect <em>rgfocpt</em>.</td>
</tr>
<tr>
<td>CNM_CANT_BE_FOCALPT</td>
<td>500</td>
<td>Registration unsuccessful. <em>rgfocpt</em> has an incompatible value.</td>
</tr>
<tr>
<td>CNM_BAD_REPLACE_VALUE</td>
<td>504</td>
<td>Incorrect <em>rgrepl</em>.</td>
</tr>
<tr>
<td>CNM_BAD_PRI_VALUE</td>
<td>512</td>
<td>Incorrect <em>rgpri</em> value specified. Value must be blank, LOW, NORMAL, HIGH, or TEST.</td>
</tr>
<tr>
<td>CNM_BAD_NOTIFY</td>
<td>592</td>
<td>Incorrect <em>rgnotify</em> value specified.</td>
</tr>
<tr>
<td>CNM_BAD_CES + X</td>
<td>9000 + X</td>
<td>Nonzero return code, X, from DSICES macro using <em>rgcmd</em>.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMSCAN (CNMSSSCAN): Parse or Convert Character String–PL/I Only**

You can use the CNMSCAN service routine to extract data from an input string and assign the extracted data to one or more receiving variables. The input string is scanned from left to right and is interpreted according to the specifications defined by the format string. Each receiving variable must have the same data type as its corresponding type specifier in the format string. You can specify up to 10 receiving variables in the argument list. The number of fields successfully parsed and converted is returned to you in *panumfld*.

When the first format specification is found, the value of the first input field is converted according to the first format specification and stored in the first receiving variable in the argument list. When the second format specification is found, the value of the second input field is converted according to the second format specification and stored in the second receiving variable in the argument list. This process continues until all of the format specifications in the format string are processed.

The CNMSCAN routine syntax follows:

**PL/I CALL FORMAT:**
CALL CNMSCAN(hlbptr, pastring, pattern, panumfld, pafld1, ..., pafld10)

PL/I MACRO FORMAT:
CNMSSCAN DATA(pastring) FORMAT(pattern) COUNT(panumfld)
P1(pafld1) ... P10(pafld10)

Where:
hlbptr
Is a 4-byte pointer field containing the address of the HLB control block.

pafld1, ......, pafld10
Is a list of receiving variables. The last variable named in this list receives the value of the last input field parsed and converted according to the last specification in the format string. You must declare each of the variables named in this list to have the same data type as its corresponding type specifier in the format string. You can specify up to 10 variables to receive parsed and converted data.

panumfld
Is a 4-byte integer field containing the number of fields successfully parsed and converted. This field is returned to you.

pastring
Is a varying length character field containing the input string to be parsed and converted.

pattern
Is a varying length character field containing the format specifications. The format string (pattern) determines how the data elements in the input string (pastring) are parsed and converted.

Usage Notes:
1. The format string consists of a series of format specifications that are defined as follows:
   • The character %, which designates the beginning of each format specification. (Required for each format specification.)
   • The character *, which indicates that the data in the corresponding input field is skipped. No assignment is made to a receiving variable. (Optional)
   • A numerical value that defines a maximum field width to scan in the input string. (Optional)
   • The character h (halfword) or l (long or fullword), which indicates the size of the argument that the value of the parsed or converted input data is assigned. (Optional)
   • Any number of white-space characters, which can be interspersed within or between format specifications for readability. However, do not insert blanks between braces ({}). (Optional)
   • The type specifier for the parsed or converted input data to be stored in the receiving variable. (Required)
2. The type specifier directs the conversion of the input field. CNMSCAN places the result in the receiving variable, unless you specify assignment suppression with an *. An input field is a string of characters other than spaces, unless the type specifier is a c or { }. The input field extends to the next character that does not meet the criteria of the type specifier, or until the width of the field, if specified, is reached.
3. The type specifier determines the interpretation of the next input field. If the input field does not meet this expectation, CNMSCAN returns to its caller. Valid specifiers are:

- **c** Expect any character. Space characters that are ordinarily skipped are read. Specify a field width to parse and convert more than one character. For example, %3c retrieves the next three characters of the input string. To skip over spaces before obtaining a character, use %1s. See the description that follows on the type specifier s.

**Note:** The receiving variable to contain the character string result must be declared as a fixed-length character string.

- **d** Expect a decimal value. Input is an optionally signed sequence of decimal digits. Any spaces in the input string preceding the decimal digits are skipped. The decimal digits are delimited by the next nondecimal character in the input string.

- **n** A data element is not parsed and converted from the input string. The value stored is the number of characters successfully read (including blanks) from the input string up to that point in the call to CNMSCAN. 

**Note:** If the end of the input string occurs before the %n is reached, the value stored is zero (0).

- **s** Expect a character string. Any spaces in the input string preceding the character string are skipped. The character string is delimited by a space. If you do not specify a field width, the field width defaults to the length of the string.

**Note:** The receiving variable to contain the character string result must be declared as a varying length character string.

- **u** Expect an unsigned decimal value. Any spaces in the input string preceding the decimal digits are skipped. The decimal digits are delimited by the next nondecimal character in the input string.

- **x** Expect a hexadecimal value. Input is an optionally signed sequence of hexadecimal digits. Any spaces in the input string preceding the hexadecimal digits are skipped. The hexadecimal digits are delimited by the next nonhexadecimal character in the input string.

- **{}** Expect a string that is not delimited by spaces. The character string is set within the braces. The corresponding input field is read to the first character that does not appear between braces ({}). If the first character is ¬ (or ‘5F’x), the effect is reversed. The input field is read to the first character that appears between braces ({}). Do not insert blanks between braces ({})) unless this is the desired effect.

  - `{}` Parses until the end of the string.
  - `{¬a}` Parses until the character a is found.
  - `{a}` Parses until any character other than an a is found.
  - `{} Pares until any character is found.

**Note:** The receiving variable to contain the character string result must be declared as a varying length character string.

4. If your format string specifies fewer data elements to be parsed and converted than your input string contains, the remaining data elements in the input string are ignored.
5. CNMSCAN returns when it encounters a format specification it does not expect or when it reaches the end of the input string.

6. If you invoke CNMSCAN using the PL/I call format and all 10 paflds are not specified, a warning message is issued at compile time.

7. You can use CNMSCAN only in an HLL command processor or installation exit routine written in PL/I.

8. The PARSEL2R command provides a function similar to CNMSCAN. However, because of its conversion capabilities, CNMSCAN is more suitable to HLL command processors.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by panumfld or pafld1,...,pafld10 is not addressable.</td>
</tr>
</tbody>
</table>

CNMSCOP (CNMSCOPECK): Command, Keyword, and Value Authorization Checking

The CNMSCOP service routine determines whether the user is authorized to issue a specific command, keyword, and value combination from a particular operator ID. The security check is based on the authority of the operator ID of the task invoking CNMSCOP. The type of authorization checking performed is determined by the setting of the CMDAUTH keyword on the REFRESH command or in CNMSTYLE. Only NetView authorization checking is performed. No attempt is made to determine if a resource is in the task’s (operator’s) span of control.

The CNMSCOP routine syntax follows:

**PL/I CALL FORMAT:**

```pli
call cnmscop(hlbptr,sccmd,sckwd,scvalue)
```

**PL/I MACRO FORMAT:**

```pli
cnmscopeck verb(sccmd) keyword(sckwd) value(scvalue)
```

**C INVOCATION:**

```c
void cnmscop(char *sccmd, char *sckwd, char *scvalue)
```

**Where:**

- `hlbptr` is a 4-byte pointer field containing the address of the HLB control block.
- `sccmd` is an 8-byte character field that specifies the verb of the command to be checked for authorization. Blanks or (*) imply that the command verb that started the HLL command processor should be used. This field is required.
- `sckwd` is an 8-byte character field that specifies a keyword of the command to be checked for authorization. Blanks or (*) imply that no specific keyword is checked.
scvalue

Is an 8-byte character field that specifies a value of sckwd to be checked for authorization. Blanks or (*) imply that no specific keyword value is to be checked.

Usage Notes:

1. Authorization to access commands, keywords, and keyword values is defined according to the value of the CMDAUTH keyword on the REFRESH command or in CNMSTYLE. For more information, refer to Tivoli NetView for z/OS Security Reference.

2. CNMSCOP does not check the validity of a command. Use it only to verify whether an operator has authorization to issue a particular command.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_COMMAND_NA</td>
<td>132</td>
<td>sccmd not authorized.</td>
</tr>
<tr>
<td>CNM_KEYWORD_NA</td>
<td>136</td>
<td>sckwd not authorized.</td>
</tr>
<tr>
<td>CNM_VALUE_NA</td>
<td>140</td>
<td>scvalue not authorized.</td>
</tr>
<tr>
<td>CNM_BAD_COMMAND</td>
<td>144</td>
<td>Incorrect syntax of scmd or sccmd was not found. Check for length greater than (&gt; 8 or incorrect characters in scmd. sccmd might be incorrectly defined. See usage notes if CNMSCOP was started from an HLL installation exit routine.</td>
</tr>
<tr>
<td>CNM_BAD_KEYWORD</td>
<td>148</td>
<td>VALUE (scvalue) was specified without a keyword (sckwd). sckwd must be specified when scvalue is specified.</td>
</tr>
<tr>
<td>CNM_SAF_FAILURE</td>
<td>624</td>
<td>An unexpected return code was received from SAF.</td>
</tr>
<tr>
<td>CNM_KWRD_VAL_NA</td>
<td>628</td>
<td>The keyword and value combination specified is not authorized. This return code can only be issued when a command authorization table or an SAF product is being used for command authorization.</td>
</tr>
<tr>
<td>CNM_BAD_SEC_ENVIR</td>
<td>632</td>
<td>Authorization to issue this command, 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.</td>
</tr>
<tr>
<td>CNM_TBL_FAILURE</td>
<td>636</td>
<td>Authorization to issue this command, 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.</td>
</tr>
</tbody>
</table>
**CNMSCP (CNMSCOPECK)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_NO_SEC_INFO</td>
<td>640</td>
<td>Authorization to issue this command, 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, keyword or value being checked.</td>
</tr>
<tr>
<td>CNM_NO_OP_INFO</td>
<td>644</td>
<td>Authorization to issue this command, 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, keyword or value being checked.</td>
</tr>
<tr>
<td>CNM_BAD_CES + X</td>
<td>9000 + X</td>
<td>Nonzero return code, X, from DSICES macro.</td>
</tr>
<tr>
<td>CNM_BAD_KVS + X</td>
<td>11000 + X</td>
<td>Nonzero return code, X, from DSIKVS macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMSMSG (CNMSENDMSG): Send Message or Command**

The CNMSMSG service routine enables you to send a message or command to specific destinations in your network.

The CNMSMSG routine syntax follows:

**PL/I CALL FORMAT:**

CALL CNMSMSG(hlbptr, smtext, smmsgtyp, smdestyp, smdestid)

**PL/I MACRO FORMAT:**

CNMSENDMSG DATA(smtext) MSGTYPE(smmsgtyp) DESTTYPE(smdestyp) DEST(smdestid)

**C INVOCATION:**

void Cnmsmsg(void *smtext, char *smmsgtyp, char *smdestyp, char *smdestid)

**Where:**

**hlbptr**

Is a 4-byte pointer field containing the address of the HLB control block.

**smdestid**

Is an 8-byte character field that specifies the destination ID. This is a required operand when *smdestyp* is EXTLOG, SEQLOG, TASK, or OPCLASS.

When *smdestyp* is EXTLOG, SEQLOG, or TASK, *smdestid* is the name of the destination task. You can use an asterisk (*) to imply “self” when *smdestyp* = TASK. Specifying *smdestid* = * is the same as issuing CNMSMSG with *smdestyp* = OPER and *smdestid* = null.

When *smdestyp* is OPCLASS, *smdestid* is the group ID of a particular group of operators defined by the ASSIGN command. Refer to the ASSIGN command in the NetView online help for more information.
**Note:** PPT is accepted as a synonym for the primary POI task (xxxxxPPT) where xxxx is the domain ID in the local NetView program.

**smdestyp**

Is an 8-byte character field that specifies the destination type. This is a required operand. Possible values follow:

- **AUTHRCV**  
  Authorized receiver
- **EXTLOG**  
  External system management facility (SMF) log
- **NETVLOG**  
  NetView log
- **OPCLASS**  
  All operators in group
- **OPER**  
  Operator task invoking this service routine
- **SEQLOG**  
  Sequential log
- **SYSOP**  
  System console
- **TASK**  
  Another task

**smmsgtyp**

Is an 8-byte character field that specifies the message type. This is a required operand. The possible values follow:

- **COMMAND**  
  Is the command to be executed. (The command is asynchronously scheduled for execution.)
- **DATA**  
  Is the nonprintable data in response to REQUEST.

**Note:** NetView places a process ID in the origin block (ORIGBLCK). This ID must be included at the beginning of the returned data. This process ID is used to route data to the correct instance of an HLL command processor or installation exit routine if there are multiple activations of the same HLL command processor or installation exit routine. The data returned from CNMSMSG with **smmsgtyp**=DATA can be read by CNMGETD from the data queue (DATAQ).

- **MSG**  
  Single line message. ORIG LINE TYPE=''
- **MSG_C**  
  Control line message. ORIG LINE TYPE='C'
- **MSG_D**  
  Data line message. ORIG LINE TYPE='D'
- **MSG_E**  
  End of multiline message. ORIG LINE TYPE='E'
- **MSG_F**  
  MSG_D and MSG_E combined. ORIG LINE TYPE='F'
- **MSG_L**  
  Label line message. ORIG LINE TYPE='L'

**REQUEST**

Is the request for data. REQUEST is similar to COMMAND except the command to execute is the name of the HLL command processor that is to return data through CNMSMSG with **smmsgtyp**=DATA.

**smtext**

Is a varying length character field containing the message text. This is a required operand. The values are explained as follows:

- If **smmsgtyp** is **MSG**, **MSG_C**, **MSG_L**, **MSG_D**, **MSG_E**, or **MSG_F**, **smtext** is the text of the message.
If `smmsgtyp` is COMMAND, `smtext` is either a command procedure name or a NetView command.

If `smmsgtyp` is REQUEST, the first token must be the name of the HLL command processor that sends a reply to the request. NetView concatenates the process ID to the end of `smtext` and the request for data is sent. As a result, if `smtext` contains input for the target task, the target task must take into account that the last 8 bytes of CMDBUF are always the process ID.

If `smmsgtyp` is DATA, the process ID must be concatenated with the data. You must specify `smtext` as follows:

```plaintext
smtext = ORIGIN->ORIG_PROCESS||'text'
```

Table 20 shows message and destination type combinations that are not valid.

<table>
<thead>
<tr>
<th>OPER</th>
<th>TASK</th>
<th>SYSOP</th>
<th>NETVLOG</th>
<th>EXTLOG</th>
<th>SEQLOG</th>
<th>AUTHRCV</th>
<th>OPCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSG_C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSG_L</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSG_D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSG_E</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSG_F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>REQUEST</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>DATA</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
5. You can display as many control (MSG_C) and label (MSG_L) lines on a
console as desired. However, a maximum of six control or label lines are held
on the screen if the data lines for that multiline message cause the screen to
wrap.

6. Do not invoke CNMSMSG with *smdestyp = OPER* while holding a lock. The
operator task could be running with autowrap off, and the HLL command
processor or installation exit routine might hang waiting for the operator to
clear the screen, thus holding the lock for an indefinite period.

7. When AUTHCHK SOURCEID is specified either in CNMSTYLE or on the
REFRESH command, command authorization checking is performed against the
original source of the command rather than against the environment in which
the command actually runs. When CNMSMSG is used to queue a command, a
SOURCEID is also queued. If the invoker is running in an installation exit, the
SOURCEID is the name of the task (TVBOPID) under which the installation
exit that started CNMSMSG is running. If CNMSMSG is started in a command
processor, the SOURCEID is the ID of the task under which the command is
issued or the existing SOURCEID at the time the command was queued. For
more information about SOURCEID, refer to the *Tivoli NetView for z/OS
Administration Reference*.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>Not started from an allowed installation exit.</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>Nonzero return code from DSIGET macro.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td><em>sntext</em> is too long.</td>
</tr>
<tr>
<td>CNM_BAD_MSGTYP</td>
<td>116</td>
<td>Incorrect message type.</td>
</tr>
<tr>
<td>CNM_BAD Destype</td>
<td>120</td>
<td>Incorrect destination type.</td>
</tr>
<tr>
<td>CNM_TYP_CONFLICT</td>
<td>124</td>
<td>Conflict between message and destination type.</td>
</tr>
<tr>
<td>CNM_LOG_INACTIVE</td>
<td>216</td>
<td>DSIWLS failure. Log was inactive.</td>
</tr>
<tr>
<td>CNM_TASK_INACTIVE</td>
<td>220</td>
<td>DSIMQS failure. Task was inactive.</td>
</tr>
<tr>
<td>CNM_BAD_MQS + X</td>
<td>1000 + X</td>
<td>Bad return code, X, from DSIMQS.</td>
</tr>
<tr>
<td>CNM_BAD_WLS + X</td>
<td>6000 + X</td>
<td>Bad return code, X, from DSIWLS.</td>
</tr>
<tr>
<td>CNM_BAD_PSS + X</td>
<td>7000 + X</td>
<td>Bad return code, X, from DSIPSS.</td>
</tr>
<tr>
<td>CNM_BAD_WTO + X</td>
<td>8000 + X</td>
<td>Bad return code, X, from DSIWCS.</td>
</tr>
</tbody>
</table>

Refer to *Tivoli NetView for z/OS Customization: Using Assembler* for more
information.

**CNMSMU (CNMSENDMU): Send Message Unit**

The CNMSMU service routine allows MS applications and operations management
served applications in NetView to send data to a specified target. You can invoke
CNMSMU only in applications registered through CNMRGS or the REGISTER
command.

The data is sent in the form of an MDS-MU. The invocation can supply:

- A completely built MDS-MU
- An MDS-MU that is missing one or more of:
CNMSMU (CNMSENDMU)

- A unit of work correlator
- An origin NETID
- An origin LUNAME

* Data and sufficient other fields for the service routine to build an MDS-MU header.

**Note:** Refer to the SNA library for more information about MDS-MUs.

The CNMSMU service routine builds the necessary NetView MQS buffer with the specified data and queues it for the NetView MS transport.

The CNMSMU routine syntax follows:

**PL/I CALL FORMAT:**

```pli
call cnmsmu(hlbptr,sudtype,sudata,susupcor,sucorrar,sutimout,susynch,
surplcmd,suoappl,sudstnet,sudstlu,sudstapl,sumutype,supri)
```

**PL/I MACRO FORMAT:**

```pli
CNMSENDMU DATATYPE(sudtype) DATA(sudata)
SUPPCORREL(susupcor) CORRELAREA(sucorrar)
TIMEOUT(sutimout) SYNCH(susynch)
REPLYCMD(surplcmd) ORIGAPPL(suoappl)
DESTNET(sudstnet) DESTLU(sudstlu)
DESTAPPL(sudstapl) MUTYPE(sumutype)
PRI(supri)
```

**C INVOCATION:**

```c
void Cnmsmu(char *sudtype, void *sudata, void *susupcor,
void *sucorrar, int sutimout, char *susynch,
char *surplcmd, char *suoappl, char *sudstnet,
char *sudstlu, char *sudstapl, int sumutype,
char *supri)
```

**Where:**

* `hlbptr`  
  Is a 4-byte character field containing the address of the HLB control block.

* `sucorrar`  
  Is a 53-byte area in which a new unit of work correlator (X'1549') GDS variable is created and returned by the CNMSMU service routine.

If you specify `sucorrar` for MDSMU, 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 MS transport. If you omit `sucorrar`, the MDS-MU must be complete and ready to be transmitted as supplied.

For a NONMDSMU, you must specify either `sucorrar` or `susupcor`. If you specify `sucorrar`, CNMSMU creates the UOWC GDS variable in this area and uses it in building the MDS header.

* `sudata`  
  Is a varying length character field containing the data being sent. For either MDSMU or NONMDSMU, the first 2 bytes must contain the entire length of the data and the next 2 bytes must contain the key.

* `sudstapl`  
  Is an 8-byte character field that specifies the destination application name.

The application name can be one of:
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). You must use the EBCDIC characters 0–9 and A–Z (capitals only).

A 1–8 character NetView-reserved name (padded with blanks) that represents an architecturally defined 4-byte value. NetView-reserved names and the corresponding values follow:

- ALERT: X'23F0F3F1'
- EP_ALERT: X'23F0F3F0'
- EP_OPS: X'23F0F1F6'
- MS_CAPS: X'23F0F1F1'
- OPS_MGMT: X'23F0F1F7'

This field is required for NONMDSMU.

**sudstlu**

Is an 8-byte character field that specifies the LU name or VTAM CP name of the destination LU or VTAM CP. If the destination LU is not specified, then the default is the CP. You must specify the 1–8 character LU or VTAM CP name (padded with blanks to 8 characters) beginning with an EBCDIC character 0–9 and A–Z (capitals only), @, #, or $, and followed by EBCDIC characters 0–9 and A–Z (capitals only).

This is a required field for NONMDSMUs.

**Note:** For sends within the same NetView, the send services always fills in the NetView LU name as the origin LU.

**sudstnet**

Is an 8-byte character field that specifies the ID of the network of the destination LU or VTAM CP. You must specify the 1–8 character NETID (padded with blanks to 8 characters) beginning with an EBCDIC character 0–9 and A–Z (capitals only), @, #, or $, and followed by EBCDIC characters 0–9 and A–Z (capitals only). The value of this field defaults to the network name that VTAM determines based on the LU name or VTAM CP name of the remote node (specified with the hsdstlu field) if:

- You specify blanks for this field for the PL/I call format.
- You do not specify the DESTNET keyword when using the PL/I macro format.
- You specify blanks for this field for the C invocation format.

**sudtype**

Is an 8-byte character field indicating whether the data item specified in the sudata keyword is an MDS-MU or a non-MDS-MU.

**MDSMU**

Indicates that the sudata is an MDS-MU. MDSMU is the default for the PL/I macro format.

**NONMDSMU**

Indicates that the sudata is not a complete MDS-MU because it does not contain an MDS-MU header. The CNMSMU service routine envelopes this data in an MDS-MU before sending it.

If you do not specify the DATATYPE keyword in the PL/I macro format, the default value is used.
**sumutype**

Is a 4-byte integer field that specifies the index number that identifies the type of MDS-MU to build. The type identifies whether the MDS-MU is a request, a reply, or an error message, and whether additional messages are expected. Types defined as constants are as follows:

1. REQUEST_WITH_REPLY
2. REQUEST_WITHOUT_REPLY
3. REPLY_ONLY
4. REPLY_NOTLAST
5. REPLY_LAST
6. ERROR_MESSAGE

This field is required for NONMDSMU.

**suoappl**

Is an 8-byte character field that specifies the origin 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). You must use the EBCDIC characters 0—9 and A—Z (capitals only).
- A 1–8 character NetView-reserved name (padded with blanks) that represents an architecturally defined 4-byte value. NetView-reserved names and the corresponding values follow:
  - **EP_OPS** X'23F0F1F6'
  - **OPS_MGMT** X'23F0F1F7'

OPS_MGMT should only be used as `suoappl` if operations management on the origin NetView program is defined as a focal point.

This field is required for NONMDSMU.

**supri**

Is an 8-byte character field that specifies the MQS priority for incoming solicited requests or any MDS error messages resulting from any outgoing MDS-MUs. The MQS priority is used when the MS transport uses the MQS for processing any solicited MDS-MUs or any MDS error messages.

- **HIGH** Processing begins after any NORMAL requests currently in progress completes, but before queued NORMAL or LOW requests.
- **LOW** Processing is preempted by HIGH and NORMAL priority requests. This is the default for all requests other than synchronous requests.
- **NORMAL** Processing preempts a queue of LOW priority requests. This is the default for synchronous requests.
- **TEST** CNMSMU queues the request based on the command priority of the receiving task. The command priority can be set using the OVERRIDE or DEFAULT commands. Refer to the NetView online help.

**surplcmd**

Is an 8-byte character field containing the name of the NetView command to be driven with the reply. The `surplcmd` field is used only in an MS application that is sending a REQUEST_WITH_REPLY with the reply being received asynchronously. Otherwise, it is ignored.
This is an optional field. The default is the registered command for the invoking application.

**susupcor**

Is a varying length character field containing a complete unit of the unit of work correlator (X'1549') GDS variable. Refer to the SNA library for more information about defining the correlator.

The **susupcor** field is not valid for MDSMU. For NONMDSMU, you must specify either **suspcor** or **sucorrar**. If you specify **susupcor**, the supplied value is used to build the MDS header. No validity checking is done for a correlator supplied by the invoker.

**susynch**

Is an 8-byte character field that specifies whether the MS application is to receive the reply synchronously.

- **NO**
  - Indicates that the reply is received asynchronously. NO is the default for the PL/I macro format.

- **NO_BUF**
  - Do not suspend the application but buffer replies until the one last is received. This value is equivalent to the NO value.

- **NO_UNBUF**
  - Do not suspend the application and forward replies immediately.

- **YES**
  - Indicates that the reply is received synchronously.

- **YES_BUF**
  - Suspend the application and buffer its replies. This value is equivalent to the YES value.

If you do not specify the SYNCH keyword when using the PL/I macro format for REQUEST_WITH_REPLY, the default value is used. Otherwise, this field is required for REQUEST_WITH_REPLY.

**sutimout**

Is a 4-byte integer field that specifies the number of seconds to wait for a reply to an outstanding REQUEST_WITH_REPLY.

For a REQUEST_WITH_REPLY that generates multiple replies, the timeout value applies only to the last reply.

For requests that generate multiple replies, the timeout value applies only to the last reply.

NetView initializes the default and maximum timeout values for the LU 6.2 transport send services. The initial default and maximum timeout values are 120 seconds and 86400 seconds, respectively. You can change these values with the DEFAULTS command.

The valid values for **sutimout** are:

- **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 TIMEOUT keyword when using the PL/I macro format for REQUEST_WITH_REPLY, the default timeout value is used. Otherwise, this field is required for REQUEST_WITH_REPLY.

**Usage Notes:**
CNMSMU (CNMSENDMU)

1. For a synchronous REQUEST_WITH_REPLY, control is returned to the invoking program after the last reply or an error message is received and placed on the MDSMUQ data queue. Otherwise, control is returned after CNMSMU successfully queues the request to the MS transport.

2. When the invoking program is suspended because of a synchronous REQUEST_WITH_REPLY, the NetView task where the program is running is not suspended. The task still receives and processes messages and commands.

3. For a synchronous REQUEST_WITH_REPLY from a data services task (DST), a DSRB is marked in-use and the DSRB is not available for other use until the suspended program is resumed.

4. For MDSMU, all fields within the MDS-MU header must be correct except for origin NETID and LUNAME. NETID and LUNAME must be left out, not blank or null, for the service routine to determine and set these fields. If the correlator is not contained in the data, you must specify sucorrar.

5. For REPLY_ONLY, REPLY_NOTLAST, REPLY_LAST, and ERROR_MESSAGE, you must specify susupcor to return the correlator sent with the request.

6. The MS transport implements a timeout value for the application receiving the data. If the invocation of CNMSMU 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.

7. When VTAM is active, you can use CNMSMU to send data to another application in the same domain.

8. If sudstnet is not the NETID determined by VTAM for the LU specified in sudstlu, the send fails.

9. An MS application or operations management served application cannot send data to itself within the same NetView program.

10. If you specify both susupcor and sucorrar, susupcor is used.

11. Return code 24 or 28 from DSIPUSH indicates that DSIOLGFP is not defined correctly in DSICMD.

12. If CNM_BAD_CES is returned:
   • Make sure DSI6SNDP is defined correctly in DSICMD.
   • If you specify hrrepcmd, make sure it is defined correctly in DSICMD.
   • If you specify a synchronous REQUEST_WITH_REPLY, make sure DSIOSRCP is defined correctly in DSICMD.
   • If the sending application is an operations management served application, make sure DSIOARCP is defined properly in DSICMD.

13. Refer to the Tivoli NetView for z/OS Administration Reference for the correct definitions of the NetView supplied command processors.

14. For MDSMU, if you omit the NETID subfield of the destination subvector from the MDS-MU header, VTAM determines the network name used, based on the LU name in the NAU name subfield of the destination subvector.

15. If you do not specify the destination NETID, and the destination LU name exists in more than one network, VTAM determines the destination NETID based on the active configuration.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Requested function was performed.</td>
</tr>
<tr>
<td>CNM_BAD_INVOCATION</td>
<td>4</td>
<td>Task is terminating, and a request-with-reply is being sent.</td>
</tr>
<tr>
<td>Return Code Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CNM_NO_STORAGE</td>
<td>24</td>
<td>No storage is available.</td>
</tr>
<tr>
<td>CNM_NOT_IN_ASYNC</td>
<td>44</td>
<td>Send MU service is started in an asynchronous exit.</td>
</tr>
<tr>
<td>CNM_BAD_TIMEOUT</td>
<td>56</td>
<td>Time-out value is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>MDS-MU length is not valid.</td>
</tr>
<tr>
<td>CNM_TASK_INACTIVE</td>
<td>220</td>
<td>DS6DST task is inactive.</td>
</tr>
<tr>
<td>CNM_BAD_DATA_TYPE</td>
<td>400</td>
<td>Data type is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DATA</td>
<td>404</td>
<td>DATA missing or is not valid.</td>
</tr>
<tr>
<td>CNMSAME_APPL</td>
<td>408</td>
<td>Application cannot send data to the same application within the same NetView program.</td>
</tr>
<tr>
<td>CNM_SYNCH_NOT_COMP</td>
<td>412</td>
<td>SYNCH(YES) is not allowed under a NetView installation exit or a PPT.</td>
</tr>
<tr>
<td>CNM_OAPPL_NOT_REG</td>
<td>416</td>
<td>Application is not registered.</td>
</tr>
<tr>
<td>CNM_BAD_SAPPL</td>
<td>420</td>
<td>Operations management served application is not registered.</td>
</tr>
<tr>
<td>CNM_BAD_UOW</td>
<td>424</td>
<td>UOW missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_RTI</td>
<td>428</td>
<td>R&amp;TI missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_OAN</td>
<td>432</td>
<td>OAN missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DAN</td>
<td>436</td>
<td>DAN is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_OAPPL</td>
<td>440</td>
<td>Origin application name is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DNETID</td>
<td>444</td>
<td>Destination network ID missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DLU</td>
<td>448</td>
<td>Destination LU name missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_DAPPL</td>
<td>452</td>
<td>Destination application name missing or is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_OII</td>
<td>456</td>
<td>OII in R&amp;TI does not match TVBOPID.</td>
</tr>
<tr>
<td>CNM_BAD_REPLY</td>
<td>460</td>
<td>Reply is not valid.</td>
</tr>
<tr>
<td>CNM_BAD_MUTYPE</td>
<td>464</td>
<td>Bad MUTYPE given.</td>
</tr>
<tr>
<td>CNM_BAD_SYNCH</td>
<td>468</td>
<td>Bad SYNCH option.</td>
</tr>
<tr>
<td>CNM_BUSY</td>
<td>472</td>
<td>Too many concurrent requests.</td>
</tr>
<tr>
<td>CNM_SYNCH_CMD_MISSING</td>
<td>508</td>
<td>SYNCH(YES) is specified but DSIOSRCP is not defined or is not defined correctly in DSICMD.</td>
</tr>
<tr>
<td>CNMSAME_OMAPPL</td>
<td>548</td>
<td>Operations management served application cannot send data to the same operations management served application within the same NetView.</td>
</tr>
<tr>
<td>CNM_REQ_CANCELED</td>
<td>552</td>
<td>Synchronous request canceled by user.</td>
</tr>
<tr>
<td>CNM_TASK_NO_AUTH</td>
<td>556</td>
<td>Task does not have authorization to execute the registered command associated with the origin application or OAN.</td>
</tr>
<tr>
<td>Return Code Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CNM_BAD_OM_MDSEM</td>
<td>560</td>
<td>Operations management served application is not allowed to send MDS error message. Routing report should be sent instead.</td>
</tr>
<tr>
<td>CNM_SNACR_MISSING</td>
<td>588</td>
<td>SNACR (X'1532') is missing from MDS error message.</td>
</tr>
<tr>
<td>CNM_NETID_UNINIT</td>
<td>596</td>
<td>NETID is not initialized.</td>
</tr>
<tr>
<td>CNM_BAD_MQS + X</td>
<td>1000 + X</td>
<td>X is the return code from DSIMQS.</td>
</tr>
<tr>
<td>CNM_BAD_PUSH + X</td>
<td>4000 + X</td>
<td>X is the return code from DSIPUSH.</td>
</tr>
<tr>
<td>CNM_BAD_CES + Y</td>
<td>9000 + Y</td>
<td>Reply command is not valid. Y is the return code from DSICES.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMSUBS (CNMSUBSYM): Substitute System Symbolics**

You can use the CNMSUBS service routine to substitute any MVS or user-defined system symbolics found in the source string for their system values and place the resulting string in the target string. The NetView supplied &DOMAIN symbolic is also included in the substitution process.

The CNMSUBS routine syntax follows:

**PL/I CALL FORMAT:**

```
CALL CNMSUBS(hlbptr,sssource,sstarget,sslen,sstype)
```

**PL/I MACRO FORMAT:**

```
CNMSUBSYM SOURCE(sssource) TARGET(sstarget) LENG(sslen) COPYTYPE(sstype)
```

**C INVOCATION:**

```
void Cnmsubs(void *sssource, void *sstarget, int *sslen, char *sstype)
```

*Where:*

*hlbptr*

Is a 4-byte pointer field containing the address of the HLB control block.

*sslen*

Is a 4-byte integer field containing the length of both *sssource* and *sstarget* which must be the same length.

If the value specified by *sslen* is less than the length of the data to be returned, the truncated data is returned and a return code of CNM_DATA_TRUNC is generated. The length of the returned data is stored in HLBLENG(*hlbleng)*.

If the value specified by *sslen* is equal to or greater than the length of the data to be returned and HLBRC(*hlbrc*) = CNM_GOOD, the length of the returned data is stored in HLBLENG(*hlbleng)*.

If the value specified is greater than the length of the receiving data area (*sstarget*), a storage overlay can occur.

*sssource*

Is an 4-byte pointer field containing the address of the source character string.
which has MVS system symbolics embedded in it or a single MVS system symbolic. Substitution is always performed on the &DOMAIN symbolic, unless substitution was disabled when NetView was started. For MVS and user-defined system symbolics, substitution is not performed if you are not running on an MVS system, you are running on an MVS system prior to MVS Version 5 Release 2, substitution was disabled when NetView was started, or you have not defined an MVS system symbolic on your MVS system.

sstarget
Is a 4-byte pointer field containing the address of the target character string which will contain the source string with the MVS system symbolic values substituted upon completion of this service.

dssertype
Is an 8-byte character field specifying whether the source and target fields are fixed or varying fields. Valid types are:

- **FIXTOFIX**
  Both sssource and ssstarget are fixed length fields.

- **FIXTOVAR**
  sssource is a fixed length field and ssstarget is a varying length field.

- **VARTOFIX**
  sssource is a varying length field and ssstarget is a fixed length field.

- **VARTOVAR**
  Both sssource and ssstarget are varying length fields.

Usage Notes:
1. The source and target fields may not overlap.
2. The length field of varying length fields is not set or altered by CNMSUBS.
3. When using CNMSUBS with &C and ssstype’s of FIXTOFIX, FIXTOVAR, or VARTOFIX, you must pass CNMSUBS the address of a pointer to your fixed length field. You can do this by coding a variable as a pointer to a string, and then passing CNMSUBS the address of that pointer.

Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td>sslen was too small. ssstarget data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td>sslen less than (&lt;) zero.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by sssource or ssstarget is not addressable, or the storage pointed to by ssstarget is protected (not writable).</td>
</tr>
<tr>
<td>CNM_BAD_SSTYPE</td>
<td>320</td>
<td>Incorrect ssstype.</td>
</tr>
<tr>
<td>CNM_BAD_ESTAE</td>
<td>15000</td>
<td>Nonzero return code from ESTAE macro.</td>
</tr>
</tbody>
</table>

Refer to [Tivoli NetView for z/OS Customization: Using Assembler](#) for more information.

**CNMVARS (CNMVARPOOL): Set or Retrieve Variables**
A variable pool is a collection of named variables whose values can be set or retrieved by NetView command procedures, assembler command processors, and
HLL installation exit routines. You can access the following types of variable pools from HLL command processors or HLL installation exit routines:

- The HLL command processor or installation exit routine's own pool
- The variable pool of the calling command procedure if the current HLL command processor or installation exit routine was called from a command procedure
- The task global pool shared by all command procedures and HLL installation exit routines running under a NetView task
- The common global pool shared by all command procedures and HLL installation exit routines running in a NetView address space.

The CNMVARS routine syntax follows:

**PL/I CALL FORMAT:**

```plaintext
CALL CNMVARS(hlbptr,cvfunc,cvdata,cvdatlen,cvname,cvpool)
```

**PL/I MACRO FORMAT:**

```plaintext
CNMVARPOOL FUNC(cvfunc) NAME(cvname) POOL(cvpool)
DATA(cvdata) LENG(cvdatlen)
```

**C INVOCATION:**

```c
void Cnmvars(char *cvfunc, void *cvdata, int cvdatlen,
void *cvname, char *cvpool)
```

**Where:**

- `cvdata` 
  Is a varying length character field containing the value of the named variable. This field is required for PUT and GET, but not used for DCL.

- `cvdatlen` 
  Is a 4-byte integer field containing the length of `cvdata`. This is the maximum length of the area provided to receive the returned data. You provide `cvdatlen`. This field is required only for GET.

  If the value specified by `cvdatlen` is less than the length of the data to be returned, the truncated data is returned in `cvdata` and a return code of CNM_DATA_TRUNC is generated. The full length of the data that was truncated is stored in HLBLENG (`Hlbeng`).

  If the value specified by `cvdatlen` is equal to or greater than the length of the data to be returned, and HLBRC (`Hlbrc`) = CNM_GOOD, the length of the returned data is stored in HLBLENG (`Hlbeng`).

  If the value specified by `cvdatlen` is greater than the length of the receiving data buffer (`cvdata`), a storage overlay can occur.

- `cvfunc` 
  Is an 8-byte character field that specifies the function to be performed. This field is required for all CNMVARS calls.

  - **DCL** 
    Declares the local variable to belong to one of the global pools, or resets it to the local pool. (You cannot declare a variable that belongs to the caller’s pool.)

  - **GET** 
    Gets the variable value

  - **PUT** 
    Sets the variable value
**cvname**

Is a varying length character field that specifies the name of the variable. This field is required for all functions.

Valid characters are A–Z, 0–9, @, #, $, +, ,, !, ?, and underscore. The first character of `cvname` cannot be a number or a period.

**cvpool**

Is an 8-byte character field that specifies the variable pool. This field is required for all functions.

**CALLER**

The local pool of the calling command procedure or HLL installation exit routine (if one exists).

**CGLOBAL**

Common global

**LOCAL**

The local pool of the current HLL command processor or installation exit routine

**TGLOBAL**

Task global

**hlbptr**

Is a 4-byte pointer field containing the address of the HLB control block.

**Usage Notes:**

1. You can access all existing NetView command list language and REXX global variables (both task and common) using CNMVARs. In the NetView command list language, all variable names (local and global) are restricted to a length of 1–11 characters. In REXX, local variable names can be 1–250 characters, while global variables must be 1–31 characters. In HLL, all variable names (local and global) are restricted to a length of 1–31 characters.

2. If you are accessing REXX or HLL global variables from the NetView command list language, the REXX and HLL variable names must adhere to NetView command list language rules. The character set allowed for variable names in NetView command list language is also smaller than in REXX and HLL. The valid characters for REXX variable names are the same as HLL; see the operand `cvname` previously mentioned.

3. You must have a calling NetView command list language, REXX, or HLL command procedure before you can PUT or GET to a CALLER pool. Otherwise, a return code of CNM_BAD_POOL is issued.

4. You do not have to initially put a value into the calling HLL command processor or installation exit routine’s LOCAL pool before issuing a PUT in the called HLL command processor or installation exit routine CALLER pool. When control is returned to the calling HLL command processor or installation exit routine, you can issue a GET for the same variable name in the LOCAL pool to retrieve the value set in the called HLL command processor or installation exit routine’s CALLER pool.

5. Any PUTs in an HLL command processor or installation exit routine’s CALLER pool change the value of the same variable name in the calling command procedure or HLL installation exit routine’s LOCAL pool.

6. You can get three different values (`cvdata`) in the same variable name (`cvname`) if you specify different pools (`cvpool`). For example:
7. The calling HLL command processor or installation exit routine’s LOCAL pool is the same as the called HLL command processor or installation exit routine’s CALLER pool.

8. The DCL operand can be useful when an HLL command processor invokes VIEW. You must ensure that the variables are properly declared to the corresponding common or task global pools. Otherwise, the variables used can be from the local pool and the full-screen panel is not automatically updated.

9. You must declare task and common global variables to their respective pools before invoking VIEW from an HLL command processor. Otherwise, VIEW does not pick up the values. DCL is not necessary for local variables.

10. **Example 1: Declaring Variables to the Task or Common Global Pool**
REXX or NetView command list language has declared variables to the task or common global pool and values have been assigned to these variables. These values need to be displayed on a full-screen panel from an HLL command processor or installation exit routine. Before invoking VIEW, you must code in the HLL command processor or installation exit routine one of the following:

    CNMVARS FUNC('DCL') NAME(cvname) POOL('CGLOBAL')
    CNMVARS FUNC('DCL') NAME(cvname) POOL('TGLOBAL')

11. **Example 2: Changing the Value of a Variable**
REXX or NetView command list language has declared variables to the task or common global pool and values have been assigned to these variables. These values must be changed within an HLL command processor or installation exit routine and then displayed on a full-screen panel. Before invoking VIEW, you must code in the HLL command processor or installation exit routine one of the following:

    CNMVARS FUNC('PUT') NAME(cvname) POOL('CGLOBAL') DATA(cvdata)
    CNMVARS FUNC('PUT') NAME(cvname) POOL('TGLOBAL') DATA(cvdata)

    Note: The DCL can precede the PUT.

12. **Example 3: Setting Values for Task and Common Variables**
An HLL command processor or installation exit routine has set values for either task or common variables. Before invoking VIEW, you must code in the HLL command processor or installation exit routine:

    CNMVARS FUNC('DCL') NAME(cvname) POOL('CGLOBAL')
    or
    CNMVARS FUNC('DCL') NAME(cvname) POOL('TGLOBAL')

    Note: If only a DCL is done (with no PUT), the VIEW panel is blank for that variable.

13. **Example 4: Declaring a Variable to the Local Pool**
You might need to declare (DCL) a variable back to the local pool. Assume you have the same variable name in both the local and common global pool. If you have just started VIEW, the variable is declared to the common global pool. If you want to change the value of the variable in the local pool, issue:

    CNMVARS FUNC('DCL') NAME(cvname) POOL('LOCAL')
Return Codes:

<table>
<thead>
<tr>
<th>Return Code Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNM_GOOD</td>
<td>0</td>
<td>Everything is OK.</td>
</tr>
<tr>
<td>CNM_NOT_FOUND</td>
<td>20</td>
<td><code>cvname</code> not found or value of <code>cvname</code> is null.</td>
</tr>
<tr>
<td>CNM_DATA_TRUNC</td>
<td>40</td>
<td><code>cvdatlen</code> was too small. Data truncated.</td>
</tr>
<tr>
<td>CNM_BAD_FUNC</td>
<td>52</td>
<td>Incorrect <code>cfunc</code>.</td>
</tr>
<tr>
<td>CNM_BAD_LENGTH</td>
<td>88</td>
<td><code>cvdatlen</code> less than (&lt;) 0 or <code>cvdata</code> greater than (&gt;) 255.</td>
</tr>
<tr>
<td>CNM_BAD_NAME</td>
<td>108</td>
<td>Incorrect <code>cvname</code>.</td>
</tr>
<tr>
<td>CNM_BAD_POOL</td>
<td>156</td>
<td>Incorrect <code>cvpool</code>.</td>
</tr>
<tr>
<td>CNM_BAD_ADDR</td>
<td>160</td>
<td>The storage pointed to by <code>cvdata</code> is not addressable.</td>
</tr>
<tr>
<td>CNM_BAD_CDS + X</td>
<td>14000 + X</td>
<td>Nonzero return code, X. See values for X below.</td>
</tr>
</tbody>
</table>

Values for X:

<table>
<thead>
<tr>
<th>Value for X</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Incorrect variable name.</td>
</tr>
<tr>
<td>8</td>
<td>Variable name already defined in dictionary.</td>
</tr>
<tr>
<td>12</td>
<td>Insufficient storage.</td>
</tr>
<tr>
<td>20</td>
<td>Value length limit was exceeded.</td>
</tr>
<tr>
<td>28</td>
<td>No command procedure related to current action.</td>
</tr>
<tr>
<td>32</td>
<td>Data was truncated.</td>
</tr>
</tbody>
</table>
Part 6. Appendixes
Appendix A. PL/I Control Blocks and Include Files

This appendix describes the PL/I control blocks and include files needed to write command processors and installation exits in PL/I.

DSIPLI Include File

This file includes all of the external HLL control blocks and include files needed to run PL/I programs in the NetView environment.

```plaintext
/**************************************************************************/
/* */
/* NAME = DSIPLI */
/* */
/* DESCRIPTIVE NAME = Main HLL PL/I Include File */
/* */
**************************************************************************/
/* */
/* Licensed Materials - Property of IBM */
/* 5697-B82 */
/* (C) Copyright IBM Corp. 1989, 1997. All rights reserved. */
/* */
/* US Government Users Restricted Rights - Use, duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
/* */
**************************************************************************/
/* */
/* STATUS = NetView Version 2 Release 4 */
/* */
/* FUNCTION = DSIPLI is required and must be included by all HLL */
/* programs written in PL/I. DSIPLI includes all of the */
/* external HLL control blocks and include files needed to run */
/* PL/I programs in the NetView environment. */
/* */
/* NOTES = see below */
/* */
/* DEPENDENCIES = none */
/* */
/* RESTRICTIONS = none */
/* */
/* REGISTER CONVENTIONS = not applicable */
/* */
/* PATCH LABEL = not applicable */
/* */
/* MODULE TYPE = include file */
/* */
/* PROCESSOR = PL/I */
/* */
/* EXTERNAL REFERENCES = none */
/* */
/* CHANGE ACTIVITY */
/* */
/* SPO=PCN13854,BASE,02/27/93,DKG: Add zero flag for base code */
/* */
**************************************************************************/
/* INCLUDE DSIPCONS; */
/* INCLUDE DSIPHLB; */
/* INCLUDE DSIPORIG; */
/* INCLUDE DSIPHLLS; */
/* INCLUDE DSIPCNM; */
/* INCLUDE DSPRMP; */
```
This file contains the definitions for constants that are helpful when coding HLL modules in PL/I.

DCL TRAPQ FIXED BIN(31,0) INIT(1),
   OPERQ FIXED BIN(31,0) INIT(2),
   DATAQ FIXED BIN(31,0) INIT(3),
   IDATAQ FIXED BIN(31,0) INIT(4),
   CNMIQ FIXED BIN(31,0) INIT(5),
   MDSMUQ FIXED BIN(31,0) INIT(6);

DCL INSLINE CHAR(8) INIT('INSLINE '),
   REPLINE CHAR(8) INIT('REPLINE '),
   DELLINE CHAR(8) INIT('DELLINE ');

DCL FIXTOFIX CHAR(8) INIT('FIXTOFIX'),
FIXTOVAR CHAR(8) INIT('FIXTOVAR'),
VARTOFIX CHAR(8) INIT('VARTOFIX'),
VARTOVAR CHAR(8) INIT('VARTOVAR');

/***************************************************************************/
/* Constants for calls to CNMGETD */
/***************************************************************************/
DCL GETFIRST CHAR(8) INIT('GETFIRST'),
GETMSG CHAR(8) INIT('GETMSG '),
GETNEXT CHAR(8) INIT('GETNEXT '),
GETLINE CHAR(8) INIT('GETLINE '),
PEEKBFR CHAR(8) INIT('PEEKBFR '),
PEEKLINE CHAR(8) INIT('PEEKLINE'),
FLUSHNXT CHAR(8) INIT('FLUSHNXT'),
FLUSHLIN CHAR(8) INIT('FLUSHLIN'),
FLUSHGRP CHAR(8) INIT('FLUSHGRP'),
FLUSHMSG CHAR(8) INIT('FLUSHMSG'),
FLUSHQ  CHAR(8) INIT('FLUSHQ ');

/***************************************************************************/
/* Constants for CNMCNMI */
/***************************************************************************/
DCL SENDRESP CHAR(8) INIT('SENDRESP'),
SENDPLY CHAR(8) INIT('SENDPLY');

/***************************************************************************/
/* Constants for CNMLK */
/***************************************************************************/
DCL UNLOCK CHAR(8) INIT('UNLOCK '),
LOCK  CHAR(8) INIT('LOCK '),
TEST  CHAR(8) INIT('TEST '),
WAIT  CHAR(8) INIT('WAIT '),
NOWAIT CHAR(8) INIT('NOWAIT ');

/***************************************************************************/
/* Constants for CNMNAMS, CNMPOOL and CNMCELL */
/***************************************************************************/
DCL ALLOC CHAR(8) INIT('ALLOC '),
FREE  CHAR(8) INIT('FREE '),
LOCATE CHAR(8) INIT('LOCATE '),
REALLOC CHAR(8) INIT('REALLOC ');

DCL RESIDENT FIXED BIN(31,0) INIT(0),
STORAG31 FIXED BIN(31,0) INIT(1),
STORAG24 FIXED BIN(31,0) INIT(2);

/***************************************************************************/
/* Constants for CNMMSG */
/***************************************************************************/
DCL MSG CHAR(8) INIT('MSG '),
MSG_C CHAR(8) INIT('MSG_C '),
MSG_L CHAR(8) INIT('MSG_L '),
MSG_D CHAR(8) INIT('MSG_D '),
MSG_E CHAR(8) INIT('MSG_E '),
MSG_F CHAR(8) INIT('MSG_F '),
COMMAND CHAR(8) INIT('COMMAND '),
REQUEST CHAR(8) INIT('REQUEST '),
DATA  CHAR(8) INIT('DATA '),
OPER  CHAR(8) INIT('OPER '),
TASK  CHAR(8) INIT('TASK '),
SYSOP CHAR(8) INIT('SYSOP '),
NETVLOG CHAR(8) INIT('NETVLOG '),
EXTLG  CHAR(8) INIT('EXTLOG '),
SEQLOG CHAR(8) INIT('SEQLOG '),
AUTHRCV CHAR(8) INIT('AUTHRCV '),
OPCLASS CHAR(8) INIT('OPCLASS ');
DSIPCONS

/********************************************************************
/* Constants for CNMVARS */
********************************************************************/
DCL PUT CHAR(8) INIT('PUT '),
DCL CHAR(8) INIT('DCL'),
GET CHAR(8) INIT('GET'),
LOCAL CHAR(8) INIT('LOCAL '),
TGLOBAL CHAR(8) INIT('TGLOBAL '),
CGLOBAL CHAR(8) INIT('CGLOBAL '),
CALLER CHAR(8) INIT('CALLER ');

/********************************************************************
/* Constants for CNMKIO */
********************************************************************/
DCL GET_EQ CHAR(8) INIT('GET_EQ '),
GET_EH CHAR(8) INIT('GET_EH '),
GET_NEXT CHAR(8) INIT('GET_NEXT'),
GET_PREV CHAR(8) INIT('GET_PREV'),
ERASE CHAR(8) INIT('ERASE '),
ENDEQ CHAR(8) INIT('ENDEQ '),
UPDATE CHAR(8) INIT('UPDATE '),
NOUPDATE CHAR(8) INIT('NOUPDATE'),
DIRECT CHAR(8) INIT('DIRECT ');

/********************************************************************
/* Constants for calls to CNMGETP */
********************************************************************/
DCL BYHEADER CHAR(1) INIT('H'),
BYNAME CHAR(1) INIT('P'),
BYNEXT CHAR(1) INIT('N');

/********************************************************************
/* Constants for calls to CNMC2T */
********************************************************************/
DCL SNAALERT CHAR(8) INIT('SNAALERT'),
SNACAUSE CHAR(8) INIT('SNACAUSE'),
SNAUCAUS CHAR(8) INIT('SNAUCAUS'),
SNAICAUS CHAR(8) INIT('SNAICAUS'),
SNAFCAUS CHAR(8) INIT('SNAFCAUS'),
SNAREACT CHAR(8) INIT('SNAREACT'),
SNADDATA CHAR(8) INIT('SNADDATA');

/********************************************************************
/* Constants for installation exits running under a DST */
********************************************************************/
DCL USERASIS FIXED BIN(31,0) INIT(0),
USERDROP FIXED BIN(31,0) INIT(4),
USERSWAP FIXED BIN(31,0) INIT(8),
USERLOG FIXED BIN(31,0) INIT(12),
USERLOGR FIXED BIN(31,0) INIT(16),
USERHCL FIXED BIN(31,0) INIT(20),
USERHCLR FIXED BIN(31,0) INIT(24),
USERNSL FIXED BIN(31,0) INIT(28),
USERNSLR FIXED BIN(31,0) INIT(32),
USERDINT FIXED BIN(31,0) INIT(233),
USERVINT FIXED BIN(31,0) INIT(234),
USERVINP FIXED BIN(31,0) INIT(235),
USERVOUT FIXED BIN(31,0) INIT(236),
USERCINP FIXED BIN(31,0) INIT(237),
USERCOUT FIXED BIN(31,0) INIT(238),
USERXLOG FIXED BIN(31,0) INIT(240),
USERBINT FIXED BIN(31,0) INIT(241),
USERBOUT FIXED BIN(31,0) INIT(242);

/********************************************************************
/* Constants for CNMSMU and CNMMSMU */
********************************************************************/
DCL MDSMU CHAR(8) INIT('MDSMU '),

304 Customization: Using PL/I and C
DSIPHLB Control Block

This file contains a PL/I mapping of the NetView HLL control block (HLB). The HLB is built during command processor initialization and exists for the lifetime of the command processor.

******************************************************************************
/ * Constants for calls to CNMRGS
******************************************************************************
DCL REGMSAPPL CHAR(12) INIT('REGMSAPPL ');
DCL DEREGMSAPPL CHAR(12) INIT('DEREGMSAPPL ');
DCL REGOMSERVD CHAR(12) INIT('REGOMSERVD ');
DCL DEREGOMSERVD CHAR(12) INIT('DEREGOMSERVD ');
DCL PRI_DEFAULT CHAR(8) INIT(' ');

DCL ALERT CHAR(8) INIT('ALERT ');
DCL EP_ALERT CHAR(8) INIT('EP_ALERT ');
DCL OPS_MGMT CHAR(8) INIT('OPS_MGMT ');
DCL EP_OPS CHAR(8) INIT('EP_OPS ');
DCL MS_CAPS CHAR(8) INIT('MS_CAPS ');
DCL COS_NETOP CHAR(8) INIT('COS_NETOP ');
DCL LINK_NETOP CHAR(8) INIT('LINK_NETOP ');

******************************************************************************
/ * Constants for calls to CNMMEMO and CNMMEMR
******************************************************************************
DCL INCL CHAR(6) INIT('INCL ');
DCL NOINCL CHAR(6) INIT('NOINCL ');

******************************************************************************
/ * Constants for calls to CNMHRGS
******************************************************************************
DCL REGAPPL CHAR(10) INIT('REGAPPL ');
DCL DEREGAPPL CHAR(10) INIT('DEREGAPPL ');

******************************************************************************
/ * Licensed Materials - Property of IBM
******************************************************************************
/ * 5697-B82
/ * (C) Copyright IBM Corp. 1989, 1997. All rights reserved.
/ * US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
/ *
DSIPHLB

/* STATUS = NetView Version 2 Release 4 */
/* */
/* FUNCTION = This file contains a PL/I mapping of DSIHLB, an */
/* internal control block. The HLB is built during command */
/* processor initialization and exists for the lifetime of the */
/* command processor. */
/* */
/* NOTES = see below */
/* */
/* DEPENDENCIES = none */
/* */
/* RESTRICTIONS = none */
/* */
/* REGISTER CONVENTIONS = not applicable */
/* */
/* PATCH LABEL = not applicable */
/* */
/* MODULE TYPE = structure map */
/* */
/* PROCESSOR = PL/I */
/* */
/* EXTERNAL REFERENCES = none */
/* */
/* CHANGE ACTIVITY */
/* */
/* $P0=PCN13854,BASE,02/27/93,DGK: Add zero flag for base code */
/***************************************************************************/
DCL 1 DSIHLB BASED(HLBPTR),
    3 HBLNEN FIXED (31), /* Length of HLB */
    3 HBLWKAT PTR, /* Pointer to WKA for API modules*/
    3 HBLHLLS PTR, /* Pointer to HLLS */
    3 HBLTIB PTR, /* Pointer to TIB */
    3 HBLUSERT PTR, /* User-defined field. Values can */
                      /* be assigned in an HLL command */
                      /* processor written in PL/I. */
    3 HBLRC FIXED (31), /* Return code from last API call*/
    3 HBLLENGTH FIXED (31), /* Length of data returned if */
                           /* HBLRC = 0. Otherwise, length */
                           /* of data that would have been */
                           /* returned if truncation had */
                           /* not occurred. */
    3 HBLSENSE BIT (32), /* Sense code from CNMI */
    3 HBLRSRV BIT (32), /* Reserved */
    3 HBLFFDCA CHAR (48); /* First failure data capture */

DSIPORIG Control Block

This file defines the mapping of the origin block of the request that caused the
execution of the procedure currently running.

/***************************************************************************/
/* */
/* NAME = DSIPORIG */
/* */
/* DESCRIPTIVE NAME = HLL PL/I Origin Block Mapping */
/* */
/***************************************************************************/
/* Licensed Materials - Property of IBM */
/* * 5697-B02 */
/* *(C) Copyright IBM Corp. 1989, 1997. All rights reserved. */
/* */
/* US Government Users Restricted Rights - Use, duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/* */
/***************************************************************************/
/* */
/* STATUS = NetView Version 2 Release 4 */
/* */
DSIPORIG

/*
** FUNCTION = This file defines the mapping of the origin block of the request that caused the execution of the procedure currently running.
**
** NOTES = see below
**
** DEPENDENCIES = none
**
** RESTRICTIONS = none
**
** REGISTER CONVENTIONS = not applicable
**
** PATCH LABEL = not applicable
**
** MODULE TYPE = structure map
**
** PROCESSOR = PL/I
**
** EXTERNAL REFERENCES = none
**
** CHANGE ACTIVITY
**
** $P0=PCN13854,BASE,02/27/93,DGK: Add zero flag for base code */
********************************************************************/
DCL 1 ORIG_BLOCK BASED,
3 ORIG_BLOCK LENGTH FIXED BINARY(31,0),
3 ORIG_DUMMY1 CHAR(8), /* Reserved */
3 ORIG_DOMAIN CHAR(8), /* Origin domain ID */
3 ORIG_TASK CHAR(8), /* Origin task ID */
3 ORIG_PROCESS CHAR(8),
3 ORIG_MSG_TYPE CHAR, /* Message type from HDRMTYPE */
3 ORIG_LINE_TYPE CHAR, /* Line type */
3 ORIG_DUMMY2 CHAR(2); /* Reserved */

DSIPHLLS Control Block

This file defines entry points for HLL service routines for PL/I.

/*******************************************************************************
**
** NAME = DSIPHLLS
**
** DESCRIPTIVE NAME = PL/I Definitions for HLL Service Routines
**
** Licensed Materials - Property of IBM
**
** 5697-B82
**
** (C) Copyright IBM Corp. 1989, 1997. All rights reserved.
**
** US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
**
** STATUS = NetView Version 3 Release 2
**
** FUNCTION = Define entry points for HLL service routines for PL/I. A macro definition is also provided for each HLL service routine.
**
** High-Level Language Service Routine Address Table
**
** This DSECT must correspond exactly to PART I of DSIHLLAR which contains the actual addresses of the service routines, or, in XA, the address of a branch instruction that branches to a Linkage Assist Routine (LAR) that insures */
*******************************************************************************/
/* the service routine is called with AMODE=31. */
/* */
/* NOTES = see below */
/* */
/* DEPENDENCIES = none */
/* */
/* RESTRICTIONS = none */
/* */
/* REGISTER CONVENTIONS = not applicable */
/* */
/* PATCH LABEL = not applicable */
/* */
/* MODULE TYPE = structure map */
/* */
/* PROCESSOR = PL/I */
/* */
/* EXTERNAL REFERENCES = none */
/* */
/* CHANGE ACTIVITY */
/* */
/* *********************************************/
DCL 1 DSIPHLLS BASED(HLBHLLS),
  3 HLLSHEAD CHAR(28), /* Skip over header information */
  3 HLLSLINK POINTER, /* PTR to Linkage Service Routine*/
  3 HLLSFIL1 CHAR(8),
  3 CNMCMD ENTRY(PTR, CHAR(*) VARYING)
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL2 CHAR(4),
  3 CNMVARS ENTRY(PTR, CHAR(8), CHAR(*) VARYING, FIXED BIN(31),
                CHAR(*) VARYING,CHAR(8))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL3 CHAR(4),
  3 CMMNAMS ENTRY(PTR, CHAR(8), PTR, CHAR(*) VARYING,
               FIXED BIN(31), FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL4 CHAR(4),
  3 CNMGETD ENTRY(PTR, CHAR(8), CHAR(*) VARYING,
               FIXED BIN(31), FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL5 CHAR(4),
  3 CNMSMSG ENTRY(PTR, CHAR(*) VARYING, CHAR(8), CHAR(8), CHAR(8))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL6 CHAR(4),
  3 CMMINFC ENTRY(PTR, CHAR(8), CHAR(*) VARYING, FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL7 CHAR(4),
  3 CMMINFI ENTRY(PTR, CHAR(8), FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL8 CHAR(4),
  3 CNMGETA ENTRY(PTR, CHAR(8), CHAR(*) VARYING,
               FIXED BIN(31), FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFIL9 CHAR(4),
  3 CNNMEMO ENTRY(PTR, FIXED BIN(31), CHAR(8), CHAR(8))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFILA CHAR(4),
  3 CNNMEMR ENTRY(PTR, FIXED BIN(31), CHAR(*) VARYING,
               FIXED BIN(31), CHAR(6))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFILB CHAR(4),
  3 CNNMEMC ENTRY(PTR, FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFILC CHAR(4),
  3 CNMCNMI ENTRY(PTR, CHAR(8), CHAR(*) VARYING, CHAR(8),
               FIXED BIN(31))
       OPTIONS(ASM,INTER,RETCODE),
  3 HLLSFILD CHAR(4),
Appendix A. PL/I Control Blocks and Include Files
CHAR(1), FIXED BIN(15))
3 HLLSFILU CHAR(4),
3 CNMPMDB ENTRY(PTR,PTR,PTR,CHAR(16))
   OPTIONS(ASM,INTER,RETCODE),
3 HLLSFILV CHAR(4),
3 CNMSSYM ENTRY(PTR,PTR,PTR,FIXED BIN(31),CHAR(8))
   OPTIONS(ASM,INTER,RETCODE),
3 HLLSFILW CHAR(4);

/*********************************************************************/
/* Macro definitions */
/* Each of the HLL service routines has a macro definition which */
/* inserts the HLB pointer (HLBPTR) into the parameter list. */
/*********************************************************************/
%DCL CNMCOMMAND ENTRY;
%CNMCOMMAND:PROC(DATA) STATEMENT RETURNS(CHAR);
DCL (DATA,RTNSTR) CHAR;
RTNSTR = 'CALL CNMCMD(HLBPTR,'||DATA||');';
RETURN(RTNSTR);
%END;

%DCL CNMVARPOOL ENTRY;
%CNMVARPOOL:PROC(FUNC,DATA,LENG,NAME,POOL) STATEMENT RETURNS(CHAR);
DCL (FUNC,DATA,LENG,NAME,POOL,RTNSTR) CHAR;
IF ¬PARMSET(LENG) THEN LENG = '0';
IF ¬PARMSET(DATA) THEN DATA = '0';
RTNSTR = 'CALL CNMVARS(HLBPTR,'||FUNC||','||DATA||','||LENG||','
           ||NAME||','||POOL||');';
RETURN(RTNSTR);
%END;

%DCL CNMNAMESTR ENTRY;
%CNMNAMESTR:PROC(FUNC,STRPTR,NAME,LENG,CLASS) STATEMENT RETURNS(CHAR);
DCL (FUNC,STRPTR,NAME,LENG,CLASS,RTNSTR) CHAR;
IF ¬PARMSET(LENG) THEN LENG = '0';
IF ¬PARMSET(STRPTR) THEN STRPTR = '(NULL())';
IF ¬PARMSET(CLASS) THEN CLASS = '0';
RTNSTR = 'CALL CNMNAMS(HLBPTR,'||FUNC||','||STRPTR||','||NAME||','
           ||LENG||','||CLASS||');';
RETURN(RTNSTR);
%END;

%DCL CNMGETDATA ENTRY;
%CNMGETDATA:PROC(FUNC,DATA,LENG,ORIGIN,QUEUE,LINE) STATEMENT RETURNS(CHAR);
DCL (FUNC,DATA,LENG,ORIGIN,QUEUE,LINE,RTNSTR) CHAR;
IF ¬PARMSET(DATA) THEN DATA = '0';
IF ¬PARMSET(ORIGIN) THEN ORIGIN = '0';
IF ¬PARMSET(LENG) THEN LENG = '0';
IF ¬PARMSET(LINE) THEN LINE = '0';
RTNSTR = 'CALL CNMGETD(HLBPTR,'||FUNC||','||DATA||','||LENG||','
           ||ORIGIN||','||QUEUE||','||LINE||');';
RETURN(RTNSTR);
%END;

%DCL CNMSENDSM Entry;
%CNMSENDSM:PROC(DATA,MSGTYPE,DESTTYPE,DEST) STATEMENT RETURNS(CHAR);
DCL (DATA,MSGTYPE,DESTTYPE,DEST,RTNSTR) CHAR;
IF ¬PARMSET(DEST) THEN DEST = '0';
RTNSTR = 'CALL CNMSMSG(HLBPTR,'||DATA||','||MSGTYPE||','
           ||DESTTYPE||','||DEST||');';
RETURN(RTNSTR);
%END;

%DCL CNMINFOC ENTRY;
%CNMINFOC:PROC(ITEM,DATA,LENG) STATEMENT RETURNS(CHAR);
DCL (ITEM,DATA,LENG,RTNSTR) CHAR;
RTNSTR = 'CALL CNMINFOC(HLBPTR,'||ITEM||','||DATA||','||LENG||');';
RETURN(RTNSTR);
%END;

%DCL CNMINFOI ENTRY;
%CNMINFOI:PROC(ITEM,DATA) STATEMENT RETURNS(CHAR);
DCL (ITEM,DATA,RTNSTR) CHAR;
RTNSTR = 'CALL CNMINFOI(HLBPTR,'||ITEM||','||DATA||');';
RETURN(RTNSTR);
%END;

%DCL CNMGETATTR ENTRY;
%CNMGETATTR:PROC(ITEM,DATA,LENG,QUEUE) STATEMENT RETURNS(CHAR);
DCL (ITEM,DATA,LENG,QUEUE,RTNSTR) CHAR;
RTNSTR = 'CALL CNMGETA(HLBPTR,'||ITEM||','||DATA||','||LENG||','||QUEUE||');';
RETURN(RTNSTR);
%END;

%DCL CNMOPENMEM ENTRY;
%CNMOPENMEM:PROC(TOKEN,DATASET,MEMBER) STATEMENT RETURNS(CHAR);
DCL (TOKEN,DATASET,MEMBER,RTNSTR) CHAR;
RTNSTR = 'CALL CNMOPEN(HLBPTR,'||TOKEN||','||DATASET||','||MEMBER||');';
RETURN(RTNSTR);
%END;

%DCL CNMREADMEM ENTRY;
%CNMREADMEM:PROC(TOKEN,DATA,LENG,INCL) STATEMENT RETURNS(CHAR);
DCL (TOKEN,DATA,LENG,RTNSTR,INCL) CHAR;
IF ¬PARMSET(INCL) THEN INCL = '''NOINCL''';
RTNSTR = 'CALL CNMREAD(HLBPTR,'||TOKEN||','||DATA||','||LENG||',',
||INCL||')';
RETURN(RTNSTR);
%END;

%DCL CNMCLOSMEM ENTRY;
%CNMCLOSMEM:PROC(TOKEN) STATEMENT RETURNS(CHAR);
DCL (TOKEN,RTNSTR) CHAR;
RTNSTR = 'CALL CNMCLM(HLBPTR,'||TOKEN||');'
RETURN(RTNSTR);
%END;

%DCL CNM1 ENTRY;
%CNM1:PROC(FUNC,DATA,DEST,TIMEOUT) STATEMENT RETURNS(CHAR);
DCL (FUNC,DATA,DEST,TIMEOUT,RTNSTR) CHAR;
IF ¬PARMSET(TIMEOUT) THEN TIMEOUT = '0';
RTNSTR = 'CALL CNM1C(HLBPTR,'||FUNC||','||DATA||','||DEST||','||TIMEOUT||')';
RETURN(RTNSTR);
%END;

%DCL CNMSSCAN ENTRY;
%CNMSSCAN:PROC(DATA,FORMAT,COUNT,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10) STATEMENT RETURNS(CHAR);
DCL (DATA,FORMAT,COUNT,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,RTNSTR) CHAR;
IF ¬PARMSET(P1) THEN P1 = '0';
IF ¬PARMSET(P2) THEN P2 = '0';
IF ¬PARMSET(P3) THEN P3 = '0';
IF ¬PARMSET(P4) THEN P4 = '0';
IF ¬PARMSET(P5) THEN P5 = '0';
IF ¬PARMSET(P6) THEN P6 = '0';
IF ¬PARMSET(P7) THEN P7 = '0';
IF ¬PARMSET(P8) THEN P8 = '0';
IF ¬PARMSET(P9) THEN P9 = '0';
DSIPHLLS

IF ¬PARMSET(P10) THEN P10 = '0';
RTNSTR = 'CALL CNMSCAN(HLBPTR,'||DATA||','||FORMAT||','||COUNT||','|
        ||P1||','||P2||','||P3||','||P4||','||P5||','||P6||','|
        ||P7||','||P8||','||P9||','||P10||')';
RETURN(RTNSTR);
%END;

%DCL CNMKEYIO ENTRY;
%CNMKEYIO:PROC(FUNC,DATA,LENG,KEY,OPTIONS) STATEMENT RETURNS(CHAR);
IF ¬PARMSET(DATA) THEN DATA = 'ABCD';
IF ¬PARMSET(LENG) THEN LENG = '0';
IF ¬PARMSET(KEY) THEN KEY = 'X';z;
IF ¬PARMSET(OPTIONS) THEN OPTIONS = '0';
DCL (FUNC,DATA,LENG,KEY,OPTIONS,RTNSTR) CHAR;
RTNSTR = 'CALL CNMKIO(HLBPTR,'||FUNC||','||DATA||','||LENG||','|
        ||KEY||','||OPTIONS||')';
RETURN(RTNSTR);
%END;

%DCL CNMSCOP ENTRY;
%CNMSCOP:PROC(VERB,KEYWORD,VALUE) STATEMENT RETURNS(CHAR);
DCL (VERB,KEYWORD,VALUE,RTNSTR) CHAR;
IF ¬PARMSET(VERB) THEN VERB = 'X'z;
IF ¬PARMSET(KEYWORD) THEN KEYWORD = 'X'z;
IF ¬PARMSET(VALUE) THEN VALUE = 'X'z;
RTNSTR = 'CALL CNMSCOP(HLBPTR,'||VERB||','||KEYWORD||','||VALUE||')';
RETURN(RTNSTR);
%END;

%DCL CNMCOPYSTR ENTRY;
%CNMCOPYSTR:PROC(FROM,TO,LENG,COPYTYPE) STATEMENT RETURNS(CHAR);
DCL (FROM,TO,LENG,COPYTYPE,RTNSTR) CHAR;
RTNSTR = 'CALL CNMCPS(HLBPTR,'||FROM||','||TO||','||LENG||','||COPYTYPE||')';
RETURN(RTNSTR);
%END;

%DCL CNMLOCK ENTRY;
%CNMLOCK:PROC(FUNC,NAME,SCOPE,OPTION) STATEMENT RETURNS(CHAR);
DCL (FUNC,NAME,SCOPE,OPTION,RTNSTR) CHAR;
IF ¬PARMSET(SCOPE) THEN SCOPE = '0';
IF ¬PARMSET(OPTION) THEN OPTION = '0';
RTNSTR = 'CALL CNMLK(HLBPTR,'||FUNC||','||NAME||','||SCOPE||','||OPTION||')';
RETURN(RTNSTR);
%END;

%DCL CNMSTRPOOL ENTRY;
%CNMSTRPOOL:PROC(FUNC,TOKEN,NAME,LENG,PRICELLS,SECCELLS,CLASS) |
STATEMENT RETURNS(CHAR);
DCL (FUNC,TOKEN,NAME,LENG,PRICELLS,SECCELLS,CLASS,RTNSTR) CHAR;
IF ¬PARMSET(TOKEN) THEN TOKEN = '0';
IF ¬PARMSET(LENG) THEN LENG = '0';
IF ¬PARMSET(PRICELLS) THEN PRICELLS = '0';
IF ¬PARMSET(SECCELLS) THEN SECCELLS = '0';
IF ¬PARMSET(CLASS) THEN CLASS = '0';
RTNSTR = 'CALL CNMPOOL(HLBPTR,'||FUNC||','||TOKEN||','||NAME||','|
        ||LENG||','||PRICELLS||','||SECCELLS||','||CLASS||')';
RETURN(RTNSTR);
%END;

%DCL CNMLDATA ENTRY;
%CNMLDATA:PROC(FUNC,DATA,ORIGIN,QUEUE,LINE) STATEMENT RETURNS(CHAR);
DCL (FUNC,DATA,ORIGIN,QUEUE,LINE,RTNSTR) CHAR;
IF ¬PARMSET(DATA) THEN DATA = 'X'z;
IF ¬PARMSET(ORIGIN) THEN ORIGIN = 'X'z;
RTNSTR = 'CALL CNMLDATA(HLBPTR,'||FUNC||','||DATA||','||ORIGIN||')';

312  Customization: Using PL/I and C
DSIPHLLS

|QUEUE||',','|LINE|');
%END;

%DCL CNMSTRELL ENTRY;
%CNMSTRELL:PROC(FUNC,TOKEN,STRPTR) STATEMENT RETURNS(CHAR);
DCL (FUNC,TOKEN,STRPTR,RTNSTR) CHAR;
RTNSTR = 'CALL CNMCELL(HLBPTR,'||FUNC||',','|TOKEN|',','|STRPTR|');
RETURN(RTNSTR);
%END;

%DCL CNMSENDTR ENTRY;
%CNMSENDTR:PROC(TRANSID,TRTYPE,DESTNET,DESTDOM,DESTTASK,VOCAB,CORR,
PARMS,RTNSTR) CHAR;
IF ¬PARMSET(DESTNET) THEN DESTNET = ''' ''';
IF ¬PARMSET(DESTDOM) THEN DESTDOM = ''' ''';
IF ¬PARMSET(VOCAB) THEN VOCAB = ''' ''';
IF ¬PARMSET(CORR) THEN CORR = ''' ''';
IF ¬PARMSET(PARMS) THEN PARMS = '(NULL())';
RTNSTR = 'CALL CNMSENDT(HLBPTR,'||TRANSID||','||TRTYPE||','||DESTNET||','||DESTDOM||','||DESTTASK||','||VOCAB||','
|CORR||',||PARMS||');
RETURN(RTNSTR);
%END;

%DCL CNMGETPARM ENTRY;
%CNMGETPARM:PROC(METHOD,HNAME,PNAME,INDEX,DATA,LENG,TYPE,QUEUE,
CURSOR,RTNSTR) CHAR;
IF ¬PARMSET(HNAME) THEN HNAME = '''''';
IF ¬PARMSET(PNAME) THEN PNAME = '''''';
IF ¬PARMSET(INDEX) THEN INDEX = '0';
IF ¬PARMSET(TYPE) THEN TYPE = '''''';
IF ¬PARMSET(CURSOR) THEN CURSOR = '0';
RTNSTR = 'CALL CNMGTP(HLBPTR,'||METHOD||','||HNAME||','||PNAME||','
|INDEX||',||DATA||','||LENG||','||TYPE||',||QUEUE||','
|CURSOR||');
RETURN(RTNSTR);
%END;

%DCL CNMST rhetoric ENTRY;
%CNMCODE2TXT:PROC(DATA,LENG,TABLE,CODE) STATEMENT RETURNS(CHAR);
DCL (DATA,LENG,TABLE,CODE,RTNSTR) CHAR;
RTNSTR = 'CALL CNMC2T(HLBPTR,'||DATA||','||LENG||','||TABLE||','
|CODE||');
RETURN(RTNSTR);
%END;

%DCL CNMSENDMU ENTRY;
%CNMSENDMU ENTRY;

Appendix A. PL/I Control Blocks and Include Files
313
IF ¬PARMSET(DESTNET) THEN DESTNET = '?????
IF ¬PARMSET(DESTLU) THEN DESTLU = '?????
IF ¬PARMSET(DESTAPPL) THEN DESTAPPL = '?????
IF ¬PARMSET(MUTYPE) THEN MUTYPE = '0';
IF ¬PARMSET(PRI) THEN PRI = '?????
RTNSTR = 'CALL CNMSMU (HLBPTR,' || DATATYPE || ',
        DATA,
        SUPPCORREL,
        CORRELAREA,
        TIMEOUT,
        SYNCH,
        REPLYCMD,
        ORIGAPPL,
        DESTNET,
        DESTLU,
        DESTAPPL,
        MUTYPE,
        PRI);
RETURN (RTNSTR);
%END;

%DCL CNMREGIST ENTRY;
%CNMREGIST:PROC(TYPE,APPL,COMMAND,FPCATEGORY,FOCALPOINT,REPLACE,
NOTIFY,PRI)
STATEMENT RETURNS(CHAR);
DCL (TYPE,APPL,COMMAND,FPCATEGORY,FOCALPOINT,REPLACE,
NOTIFY,PRI,RTNSTR) CHAR;
IF ¬PARMSET(COMMAND) THEN COMMAND = '?????
IF ¬PARMSET(FPCATEGORY) THEN FPCATEGORY = '?????
IF ¬PARMSET(FOCALPOINT) THEN FOCALPOINT = '?????
IF ¬PARMSET(REPLACE) THEN REPLACE = '?????
IF ¬PARMSET(NOTIFY) THEN NOTIFY = 'NONE
IF ¬PARMSET(PRI) THEN PRI = '?????
RTNSTR = 'CALL CNMRGS(HLBPTR,' || TYPE || ',' || APPL || ',
        COMMAND,' || FPCATEGORY || ',
        FOCALPOINT,' || REPLACE || ',
        NOTIFY,' || PRI || ');'
RETURN(RTNSTR);
%END;

%DCL CNMAUTOTAB ENTRY;
%CNMAUTOTAB:PROC(DATA) STATEMENT RETURNS(CHAR);
DCL (DATA,RTNSTR) CHAR;
RTNSTR = 'CALL CNMAUTO(HLBPTR,' || DATA || ');'
RETURN(RTNSTR);
%END;

%DCL CNMHREGIST ENTRY;
%CNMHREGIST:PROC(TYPE,APPL,COMMAND,LOGMODE,REPLACE,NOTIFY,PRI)
STATEMENT RETURNS(CHAR);
DCL (TYPE,APPL,COMMAND,LOGMODE,REPLACE,NOTIFY,PRI,RTNSTR) CHAR;
IF ¬PARMSET(COMMAND) THEN COMMAND = '?????
IF ¬PARMSET(LOGMODE) THEN LOGMODE = '?????
IF ¬PARMSET(REPLACE) THEN REPLACE = 'YES
IF ¬PARMSET(NOTIFY) THEN NOTIFY = 'NONE
IF ¬PARMSET(PRI) THEN PRI = '?????
RTNSTR = 'CALL CNMHREGS(HLBPTR,' || TYPE || ',
        COMMAND,' || LOGMODE || ',
        REPLACE,' || NOTIFY || ',
        PRI');'
RETURN(RTNSTR);
%END;

%DCL CNMHSENDMU ENTRY;
%CNMHSENDMU:PROC(DATATYPE, DATA, SUPPCORREL, CORRELAREA,
TIMEOUT, SYNCH, REPLYCMD, ORIGAPPL, DESTNET,
DESTLU, DESTAPPL, MUTYPE, PRI)
STATEMENT RETURNS(CHAR);
DCL (DATATYPE, DATA, SUPPCORREL, CORRELAREA, TIMEOUT,
DSIPCNM Control Block

This file defines the HLL return codes for PL/I.

/**
 * NAME = DSIPCNM
 * DESCRIPTIVE NAME = HLL PL/I Return Codes
 */

DSIPHLLS

SYNCH, REPLYCMD, ORIGAPPL, DESTNET, DESTLU, DESTAPPL,
MUTYPE, PRI, RTNSTR) CHAR;

IF ¬PARMSET(DATATYPE) THEN DATATYPE = '''MDSMU ''';
IF ¬PARMSET(SUPPCORREL) THEN SUPPCORREL = '''''';
IF ¬PARMSET(CORRELAREA) THEN CORRELAREA = '''''';
IF ¬PARMSET(TIMEOUT) THEN TIMEOUT = '0';
IF ¬PARMSET(SYNCH) THEN SYNCH = '''NO ''';
IF ¬PARMSET(REPLYCMD) THEN REPLYCMD = '''''';
IF ¬PARMSET(ORIGAPPL) THEN ORIGAPPL = '''''';
IF ¬PARMSET(DESTNET) THEN DESTNET = '''''';
IF ¬PARMSET(DESTLU) THEN DESTLU = '''''';
IF ¬PARMSET(DESTAPPL) THEN DESTAPPL = '''''';
IF ¬PARMSET(MUTYPE) THEN MUTYPE = '0';
IF ¬PARMSET(PRI) THEN PRI = '''''';
RTNSTR = 'CALL CNMHSMU (HLBPTR,'||DATATYPE||','||DATA||','
      ||SUPPCORREL||','||CORRELAREA||','
      ||TIMEOUT||','
      ||SYNCH||','||REPLYCMD||','||ORIGAPPL||','
      ||DESTNET||','||DESTLU||','||DESTAPPL||','
      ||MUTYPE||','||PRI||');
RETURN (RTNSTR);

%END;

%DCL CNMOPREP ENTRY;
%CNMOPREP:PROC(ACB,TIB,RESPONSE,FUNCTION,WAITF,WAITT)
STATEMENT RETURNS(CHAR);
DCL (ACB,TIB,RESPONSE,FUNCTION,WAITF,WAITT,RTNSTR) CHAR;
IF ¬PARMSET(ACB) THEN ACB = '(NULL())';
IF ¬PARMSET(TIB) THEN TIB = '(NULL())';
IF ¬PARMSET(RESPONSE) THEN RESPONSE = '(NULL())';
IF ¬PARMSET(FUNCTION) THEN FUNCTION = '(NULL())';
IF ¬PARMSET(WAITF) THEN WAITF = 'N';
IF ¬PARMSET(WAITT) THEN WAITT = '120';
RTNSTR = 'CALL CNMQAPI (HLBPTR,'||ACB||','||TIB||','
      ||RESPONSE||','||FUNCTION||','
      ||WAITF||','||WAITT||');
RETURN (RTNSTR);

%END;

%DCL CNMPRSMDB ENTRY;
%CNMPRSMDB:PROC(MDB,SOURCE,CORREL) STATEMENT RETURNS(CHAR);
DCL (MDB,SOURCE,CORREL,RTNSTR) CHAR;
IF ¬PARMSET(SOURCE) THEN SOURCE = '(NULL())';
RTNSTR = 'CALL CNMPMDB(HLBPTR,'||MDB||','||SOURCE||','||CORREL||');
RETURN(RTNSTR);

%END;

%DCL CNMSUBSYM ENTRY;
%CNMSUBSYM:PROC(SOURCE,TARGET,LENG,COPYTYPE) STATEMENT RETURNS(CHAR);
DCL (SOURCE,TARGET,LENG,COPYTYPE,RTNSTR) CHAR;
RTNSTR = 'CALL CNMSUBS(HLBPTR,'||SOURCE||','||TARGET||','||LENG||',
      ||COPYTYPE||');
RETURN(RTNSTR);

%END;
DSIPCNM

****************************************************************************/
/* */
/* Licensed Materials - Property of IBM */
/* */
/* 5697-B82 */
/* */
/* (C) Copyright IBM Corp. 1977, 1997. All rights reserved. */
/* */
/* US Government Users Restricted Rights - Use, duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/* */
*******************************************************************************/
/* */
/* STATUS = NetView Version 3 Release 2 */
/* */
/* FUNCTION = This file defines the HLL return codes for PL/I. */
/* */
/* NOTES = see below */
/* */
/* DEPENDENCIES = none */
/* */
/* RESTRICTIONS = none */
/* */
/* REGISTER CONVENTIONS = not applicable */
/* */
/* PATCH LABEL = not applicable */
/* */
/* MODULE TYPE = constants */
/* */
/* PROCESSOR = PL/I */
/* */
/* EXTERNAL REFERENCES = none */
/* */
/* CHANGE ACTIVITY */
/* */
/* */
****************************************************************************/
/* */
/* */
%DCL CNM_GOOD FIXED; %CNM_GOOD =0;
%DCL CNM_BAD_INVOCATION FIXED; %CNM_BAD_INVOCATION =4;
%DCL CNM_TOO_MANY FIXED; %CNM_TOO_MANY =8;
%DCL CNM_BAD_SYNTAX FIXED; %CNM_BAD_SYNTAX =12;
%DCL CNM_BAD_DDNAME FIXED; %CNM_BAD_DDNAME =16;
%DCL CNM_NOT_FOUND FIXED; %CNM_NOT_FOUND =20;
%DCL CNM_NO_STORAGE FIXED; %CNM_NO_STORAGE =24;
%DCL CNM_IOERROR FIXED; %CNM_IOERROR =28;
%DCL CNM_BAD_TOKEN FIXED; %CNM_BAD_TOKEN =32;
%DCL CNM_END_FILE FIXED; %CNM_END_FILE =36;
%DCL CNM_DATA_TRUNC FIXED; %CNM_DATA_TRUNC =40;
%DCL CNM_NOT_IN_ASYNC FIXED; %CNM_NOT_IN_ASYNC =44;
%DCL CNM_BAD_RULENG FIXED; %CNM_BAD_RULENG =48;
%DCL CNM_BAD_FUNC FIXED; %CNM_BAD_FUNC =52;
%DCL CNM_BAD_TIMEOUT FIXED; %CNM_BAD_TIMEOUT =56;
%DCL CNM_NEED_PRID FIXED; %CNM_NEED_PRID =60;
%DCL CNM_TIME_OUT FIXED; %CNM_TIME_OUT =64;
%DCL CNM_BAD_QUEUE FIXED; %CNM_BAD_QUEUE =68;
%DCL CNM_BAD_INDEX FIXED; %CNM_BAD_INDEX =72;
%DCL CNM_QUEUE_EMPTY FIXED; %CNM_QUEUE_EMPTY =76;
%DCL CNM_BAD_ORIGBLOCK FIXED; %CNM_BAD_ORIGBLOCK =80;
%DCL CNM_BAD_LENGTH FIXED; %CNM_BAD_LENGTH =84;
%DCL CNM_BAD_LINETYPE FIXED; %CNM_BAD_LINETYPE =88;
%DCL CNM_NOT_MLWTO FIXED; %CNM_NOT_MLWTO =92;
%DCL CNM_BAD_LINETYPE FIXED; %CNM_BAD_LINETYPE =96;
%DCL CNM_NOCUR_LINE FIXED; %CNM_NOCUR_LINE =100;
%DCL CNM_REPEAT_NAME FIXED; %CNM_REPEAT_NAME =104;
%DCL CNM_NO_EXTNAME FIXED; %CNM_NO_EXTNAME =108;
%DCL CNM_BAD_CLASS FIXED; %CNM_BAD_CLASS =112;
%DCL CNM_BAD_MSGTYP FIXED; %CNM_BAD_MSGTYP =116;
%DCL CNM_bad_DESTYP FIXED; %CNM_bad_DESTYP =120;
%DCL CNM_TYP_CONFLICT FIXED; %CNM_TYP_CONFLICT =124;
%DCL CNM_BAD_OPTION FIXED; %CNM_BAD_OPTION =128;
%DCL CNM_COMMAND_NA FIXED; %CNM_COMMAND_NA =132;
%DCL CNM_KEYWORD_NA FIXED; %CNM_KEYWORD_NA =136;
%DCL CNM_VALUE_NA FIXED; %CNM_VALUE_NA =140;
%DCL CNM_BAD_COMMAND FIXED; %CNM_BAD_COMMAND =144;
%DCL CNM_BAD_KEYWORD FIXED; %CNM_BAD_KEYWORD =148;
%DCL CNM_NO_TRAP FIXED; %CNM_NO_TRAP =152;
%DCL CNM_BAD_POOL FIXED; %CNM_BAD_POOL =156;
%DCL CNM_BAD_ADDR FIXED; %CNM_BAD_ADDR =160;

%DCL CNM_BAD_TASKNAME FIXED; %CNM_BAD_TASKNAME =164;
%DCL CNM_BAD_MODNAME FIXED; %CNM_BAD_MODNAME =168;
%DCL CNM_BAD_ID FIXED; %CNM_BAD_ID =172;
%DCL CNM_BAD_COMBO FIXED; %CNM_BAD_COMBO =176;
%DCL CNM_TVB_INUSE FIXED; %CNM_TVB_INUSE =180;
%DCL CNM_RID_INUSE FIXED; %CNM_RID_INUSE =184;
%DCL CNM_RID_SELF FIXED; %CNM_RID_SELF =188;
%DCL CNM_BAD_PRI_COUNT FIXED; %CNM_BAD_PRI_COUNT =192;
%DCL CNM_BAD_SEC_COUNT FIXED; %CNM_BAD_SEC_COUNT =196;
%DCL CNM_DUPL_KEY FIXED; %CNM_DUPL_KEY =200;
%DCL CNM_NOT_IN_POOL FIXED; %CNM_NOT_IN_POOL =204;
%DCL CNM_LOCKED FIXED; %CNM_LOCKED =208;
%DCL CNM_LOCK_INUSE FIXED; %CNM_LOCK_INUSE =212;
%DCL CNM_LOG_INACTIVE FIXED; %CNM_LOG_INACTIVE =216;
%DCL CNM_TASK_INACTIVE FIXED; %CNM_TASK_INACTIVE =220;
%DCL CNM_TIME_OUT_WAIT FIXED; %CNM_TIME_OUT_WAIT =224;
%DCL CNM_GO_ON_WAIT FIXED; %CNM_GO_ON_WAIT =228;
%DCL CNM_MSG_ON_WAIT FIXED; %CNM_MSG_ON_WAIT =232;
%DCL CNM_OPINPUT_ON_WAIT FIXED; %CNM_OPINPUT_ON_WAIT =236;
%DCL CNM_DATA_ON_WAIT FIXED; %CNM_DATA_ON_WAIT =240;
%DCL CNM_NO_TRAP_SET FIXED; %CNM_NO_TRAP_SET =244;
%DCL CNM_NO_PREV_WAIT FIXED; %CNM_NO_PREV_WAIT =248;
%DCL CNM_BAD_CSTYPE FIXED; %CNM_BAD_CSTYPE =252;
%DCL CNM_CELL_ALREADY_FREE FIXED; %CNM_CELL_ALREADY_FREE =256;
%DCL CNM_INVALID_MESSAGE FIXED; %CNM_INVALID_MESSAGE =260;
%DCL CNM_NO_LINES_IN_MSG FIXED; %CNM_NO_LINES_IN_MSG =264;
%DCL CNM_NO_DATA FIXED; %CNM_NO_DATA =268;
%DCL CNM_INVALID_MESSAGE FIXED; %CNM_INVALID_MESSAGE =272;
%DCL CNM_PARM_MISSING FIXED; %CNM_PARM_MISSING =276;
%DCL CNM_NO_MORE_DATA FIXED; %CNM_NO_MORE_DATA =280;
%DCL CNM_BAD_CURSOR FIXED; %CNM_BAD_CURSOR =284;
%DCL CNM_REQ_TOO_LONG FIXED; %CNM_REQ_TOO_LONG =288;
%DCL CNM_BAD_TASK FIXED; %CNM_BAD_TASK =292;
%DCL CNM_BAD_TRAN_HDR FIXED; %CNM_BAD_TRAN_HDR =296;
%DCL CNM_PROCESSING_ERROR FIXED; %CNM_PROCESSING_ERROR =300;
%DCL CNM_USER_ENDED FIXED; %CNM_USER_ENDED =304;
%DCL CNM_OAPPL_NOT_REG FIXED; %CNM_OAPPL_NOT_REG =308;
%DCL CNM_BAD_POOL FIXED; %CNM_BAD_POOL =312;
%DCL CNM_SAME_APPL FIXED; %CNM_SAME_APPL =316;
%DCL CNM_BAD_SSTYPE FIXED; %CNM_BAD_SSTYPE =320;
%DCL CNM_BAD_DATA_TYPE FIXED; %CNM_BAD_DATA_TYPE =324;
%DCL CNM_SAMENAME FIXED; %CNM_SAMENAME =328;
%DCL CNM_BAD_DATA FIXED; %CNM_BAD_DATA =332;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =336;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =340;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =344;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =348;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =352;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =356;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =360;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =364;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =368;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =372;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =376;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =380;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =384;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =388;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =392;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =396;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =400;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =404;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =408;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =412;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =416;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =420;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =424;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =428;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =432;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =436;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =440;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =444;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =448;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =452;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =456;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =460;
%DCL CNM_BAD_SAPPL FIXED; %CNM_BAD_SAPPL =464;
DSIPCNM

%DCL CNM_BAD_SYNC    FIXED; %CNM_BAD_SYNC    =468;
%DCL CNM_BUSY        FIXED; %CNM_BUSY        =472;
%DCL CNM_BAD_APP_NAME FIXED; %CNM_BAD_APP_NAME =476;
%DCL CNM_APP_NAME_RSTD FIXED; %CNM_APP_NAME_RSTD =480;
%DCL CNM_BAD_FPCAT_NAME FIXED; %CNM_BAD_FPCAT_NAME =484;
%DCL CNM_BAD_FPCAT_CHOICE FIXED; %CNM_BAD_FPCAT_CHOICE =488;
%DCL CNM_BAD_REG_TYPE FIXED; %CNM_BAD_REG_TYPE =492;
%DCL CNM_BAD_FOCALPT_VALUE FIXED; %CNM_BAD_FOCALPT_VALUE =496;
%DCL CNM_CANT_BE_FOCALPT FIXED; %CNM_CANT_BE_FOCALPT =500;
%DCL CNM_BAD_REPLACE_VALUE FIXED; %CNM_BAD_REPLACE_VALUE =504;
%DCL CNM_SYNC_CMD_MISSING FIXED; %CNM_SYNC_CMD_MISSING =508;
%DCL CNM_BAD_PRI_VALUE FIXED; %CNM_BAD_PRI_VALUE =512;
%DCL CNM_SYNTAX_ERROR FIXED; %CNM_SYNTAX_ERROR =520;
%DCL CNM_INVALID MEMBER FIXED; %CNM_INVALID MEMBER =524;
%DCL CNM_INVALID_NEST FIXED; %CNM_INVALID_NEST =528;
%DCL CNM_SYSTEM_ERROR FIXED; %CNM_SYSTEM_ERROR =532;
%DCL CNM_BAD_INCLUDE FIXED; %CNM_BAD_INCLUDE =536;
%DCL CNM_BAD_CHAR FIXED; %CNM_BAD_CHAR =540;
%DCL CNM_BAD_COUNT FIXED; %CNM_BAD_COUNT =544;
%DCL CNM_BAD_OMAPPL FIXED; %CNM_BAD_OMAPPL =548;
%DCL CNM_REQ_CANCELED FIXED; %CNM_REQ_CANCELED =552;
%DCL CNM_TASK_NO AUTH FIXED; %CNM_TASK_NO AUTH =556;
%DCL CNM_BAD_OM_MDSM FIXED; %CNM_BAD_OM_MDSM =560;
%DCL CNM_BAD_SEQUENCE FIXED; %CNM_BAD_SEQUENCE =564;
%DCL CNM_BAD_MDB FIXED; %CNM_BAD_MDB =568;
%DCL CNM_BAD_SOURCE_OBJ FIXED; %CNM_BAD_SOURCE_OBJ =572;
%DCL CNM_BAD_LOGMODE FIXED; %CNM_BAD_LOGMODE =580;
%DCL CNM_BAD_MQS FIXED; %CNM_BAD_MQS =1000;
%DCL CNM_BAD_EXCMS FIXED; %CNM_BAD_EXCMS =2000;
%DCL CNM_BAD_LOAD FIXED; %CNM_BAD_LOAD =12000;
%DCL CNM_BAD_LCS FIXED; %CNM_BAD_LCS =13000;
%DCL CNM_BAD_CDS FIXED; %CNM_BAD_CDS =14000;
%DCL CNM_BAD_ESTAE FIXED; %CNM_BAD_ESTAE =15000;
%DCL CNM_BAD_PAS FIXED; %CNM_BAD_PAS =16000;
%DCL CNM_BAD_SNTXS FIXED; %CNM_BAD_SNTXS =17000;
%DCL CNM_BAD_MRLBD FIXED; %CNM_BAD_MRLBD =18000;
%DCL CNM_BAD_ZCSMS FIXED; %CNM_BAD_ZCSMS =20000;
%DCL CNM_BAD_ENQ FIXED; %CNM_BAD_ENQ =21000;
%DCL CNM_BAD_TRAN_PARAM FIXED; %CNM_BAD_TRAN_PARAM =22000;

Customization: Using PL/I and C
DSIPPRM Control Block

This file defines the NetView Bridge parameter control block to PL/I HLL service routines.

 /*********************************************************************/
 /* */
 /* NAME = DSIPPRM */
 /* */
 /* DESCRIPTIVE NAME = Netview Bridge Parameter Control Block */
 /* */
 /*********************************************************************/
 /* Licensed Materials - Property of IBM */
 /* 5697-B82 */
 /* (C) Copyright IBM Corp. 1990, 1997. All rights reserved. */
 /* */
 /* US Government Users Restricted Rights - Use, duplication or */
 /* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
 /* */
 /*********************************************************************/
 /* STATUS = NetView Version 2 Release 2 */
 /* */
 /* FUNCTION = Provides the Netview Bridge parameter block to */
 /* PLI HLL service routines. */
 /* */
 /* */
 /* NOTES = none */
 /* */
 /* DEPENDENCIES = none */
 /* */
 /* RESTRICTIONS = none */
 /* */
 /* REGISTER CONVENTIONS = not applicable */
 /* */
 /* PATCH LABEL = not applicable */
 /* */
 /* MODULE TYPE = structure map */
 /* */
 /* PROCESSOR = PL/I */
 /* */
 /* EXTERNAL REFERENCES = none */
 /* */
 /* CHANGE ACTIVITY */
 /* $P0=PCN13854,BASE,02/27/93,DGK: Add zero flag for base code */
 /* */
 /*********************************************************************/
 DCL 1 DSIPPRM BASED,
 3 PRMLINK PTR, /* POINTS TO NEXT NODE */
 3 PRMPTR PTR, /* POINTS TO PARAMETER VALUE */
 3 PRMNAML FIXED BIN(15), /* LENGTH OF THE PARAMETER NAME */
 3 PRMNAME CHAR(31), /* PARAMETER NAME */
 3 PRMINDEX FIXED BIN(15), /* PARAMETER INDEX */
 3 PRMTYPE BIT(16), /* PARAMETER TYPE */
 3 PRMLENG FIXED BIN(15); /* LENGTH OF PARAMETER VALUE */
 /*********************************************************************/
 /* TYPE OF PARAMETER: EBCDIC AND INTEGER */
 /* EBCDIC PARAMETER TYPE IS '00EE'X */
 /* INTEGER PARAMETER TYPE IS '00BB'X */
DCL PRM_TYPEEBDC BIT(16) INIT ('00EE'BX),
PRM_TYPEINT BIT(16) INIT ('00BB'BX);
Appendix B. PL/I Samples

This appendix contains a table of the PL/I samples that are shipped with the NetView program in NETVIEW.V1R4M0.CNMSAMP. When a data set name is referred to in this appendix, two names are given, for example PTMPPLT (CNMS4200). The first name is the alias name, and the name in parentheses is in the NetView samples library. You can use either name to access the samples. DSICMD has definitions for the alias names to allow those names to be entered as commands.

To enter the member names as commands do the following:
1. Compile and link-edit the samples using the alias name.
2. Delete the * in column 1 of the appropriate CMDMDL statement in DSICMD 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.

Notes:
1. Refer to the prologues of the samples for information about how certain samples are related and special cases for installation exit routines.
2. Each alias name for PL/I begins with the letter P.
3. In PL/I the alias name is the same as the procedure name, which is limited to seven characters.

This appendix also contains a description of each sample, and coded samples of an installation exit routine and two command processors.

PL/I Samples Table

Table 21 lists the PL/I samples that are shipped with NetView. The table contains the function, the alias name, and the name of the member in NETVIEW.V1R4M0.CNMSAMP.

<table>
<thead>
<tr>
<th>Sample Function Description</th>
<th>PL/I Alias</th>
<th>Sample CNMSAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template for commands and installation exit routines</td>
<td>PTMPPLT</td>
<td>CNMS4200</td>
</tr>
<tr>
<td>Sample DSIEX03 to set a global variable</td>
<td>PEXIT3</td>
<td>CNMS4210</td>
</tr>
<tr>
<td>Uses CNMSMSG to send data</td>
<td>PSNDDAT</td>
<td>CNMS4211</td>
</tr>
<tr>
<td>Uses WAIT FOR DATA</td>
<td>PWATDAT</td>
<td>CNMS4212</td>
</tr>
<tr>
<td>Sample DSIEX02A changes a WTO to an MLWTO</td>
<td>PEXIT2A</td>
<td>CNMS4213</td>
</tr>
<tr>
<td>Uses CNMCNMI to forward RU's to a PU</td>
<td>PCNMI</td>
<td>CNMS4214</td>
</tr>
<tr>
<td>Uses CNMKIO for I/O to VSAM</td>
<td>PKEYIO</td>
<td>CNMS4215</td>
</tr>
<tr>
<td>HLL command using CNMSCOP for command authorization checking</td>
<td>PSCOPCK</td>
<td>CNMS4216</td>
</tr>
<tr>
<td>Display full screen VIEW panel</td>
<td>PFLVIEW</td>
<td>CNMS4217</td>
</tr>
</tbody>
</table>
PL/I Samples

Table 21. PL/I Samples Shipped with the NetView Program (continued)

<table>
<thead>
<tr>
<th>Sample Function</th>
<th>PL/I Alias</th>
<th>Sample CNMSAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activates LU and uses TRAP and WAIT to determine if activation is successful</td>
<td>PACTLU</td>
<td>CNMS4218</td>
</tr>
<tr>
<td>Uses CNMSMSG to log text to a sequential log</td>
<td>PSEQLOG</td>
<td>CNMS4219</td>
</tr>
<tr>
<td>DST initialization exit USERVSAM DST</td>
<td>PXITDI</td>
<td>CNMS4220</td>
</tr>
<tr>
<td>Primes VSAM empty data set for USERVSAM DST</td>
<td>PXITVN</td>
<td>CNMS4221</td>
</tr>
<tr>
<td>Sends a request to USERVSAM DST</td>
<td>PSNDDST</td>
<td>CNMS4222</td>
</tr>
<tr>
<td>Processes VSAM requests under USERVSAM DST</td>
<td>PDOVSAM</td>
<td>CNMS4223</td>
</tr>
<tr>
<td>Primes VSAM empty data set for PKEYIO</td>
<td>PPRIME</td>
<td>CNMS4224</td>
</tr>
<tr>
<td>Sends a software alert to an application over the high performance transport</td>
<td>PHSNDMU</td>
<td>CNMS4226</td>
</tr>
<tr>
<td>Uses CNMQAPI to access a RODM</td>
<td>PRODMCON</td>
<td>CNMS4230</td>
</tr>
<tr>
<td>Sends an MSU directly to the automation table for evaluation</td>
<td>PAUTOOTB</td>
<td>CNMS4231</td>
</tr>
<tr>
<td>Registers or deregisters an application as an MS application or an Operations Management served application</td>
<td>PREGISTR</td>
<td>CNMS4232</td>
</tr>
<tr>
<td>Sends a software alert to ALERT_NETOP from an MS application</td>
<td>PSENDMU</td>
<td>CNMS4233</td>
</tr>
<tr>
<td>Registers an application with the high performance transport</td>
<td>PHREGSTR</td>
<td>CNMS4236</td>
</tr>
<tr>
<td>Uses CNMPMDB to process MDB</td>
<td>PPRSMDB</td>
<td>CNMS4239</td>
</tr>
<tr>
<td>Uses the PIPE command to activate an LU, wait and trap the output messages</td>
<td>PACTPIP</td>
<td>CNMS4305</td>
</tr>
</tbody>
</table>

PL/I Samples Descriptions

Each sample includes a description of the function and the HLL service routines used.

PTMPPLT (CNMS4200)

This sample is a template for commands and installation exit routines in PL/I.

It appears in “Chapter 7. PL/I High-Level Language Services” on page 61.

PEXIT3 (CNMS4210)

This is a sample DSIEX03 that sets a task global variable. This global variable contains the value of the last time that a command other than PSNDDAT was entered under an operator station task (OST). PWATDAT and PSNDDAT are used to interrogate this value. The HLL service routines used in this sample are CNMINFC (CNMINFOC) and CNMVARS (CNMVARPOOL).

PSNDDAT (CNMS4211)

This sample uses CNMSMSG to send data. The sample is part of an example of sending messages containing a request, waiting for the response, and parsing the results.
The example finds the last time that a command was entered on a given OST. A task global variable is set by PEXIT3 every time a command is entered on an OST. PWATDAT uses CNMSMSG to issue a PSNDDAT on the task in question. PWATDAT then goes into a wait state. PSNDDAT retrieves the value of the global variable and uses CNMSMSG to send the data back to the task that issued the PWATDAT. PWATDAT breaks out of the wait state and parses and displays the data.

The HLL service routines used in this sample are CNMVAR5 (CNMVARPOOL), CNMSMSG (CNMSENDDMSG), and CNMINFC (CNMINFOC).

**PWATDAT (CNMS4212)**

This sample uses the WAIT FOR DATA request. The sample is part of an example of sending messages containing a request, waiting on the response, and parsing the results.

The example finds the last time that a command was entered on a given OST. A task global variable is set by PEXIT3 every time a command is entered on an OST. PWATDAT uses CNMSMSG to issue a PSNDDAT on the task in question. PWATDAT then goes into a wait state. PSNDDAT retrieves the value of the global variable and uses CNMSMSG to send the data back to the task that issued the PWATDAT. PWATDAT breaks out of the wait state and parses and displays the data.

The HLL service routines used in this sample are CNMSMSG (CNMSENDDMSG), CNMSCAN (CNMSSCAN), CNMCMD (CNMCOMMAND), and CNMGETD (CNMGETDATA).

**PEXIT2A (CNMS4213)**

This sample exit converts a write-to-operator (WTO) to a multiline write-to-operator (MLWTO) by adding two lines to the single-line WTOs that drive the exit. The HLL service routines used in this sample are CNMGETD (CNMGETDATA) and CNMALTD (CNMALTDATA).

**PCNMI (CNMS4214)**

This sample uses CNMCNMI to send FORWARD RUs to a PU requesting that the product set ID be returned. Any data returned is sent as a message to the operator. The prolog of the sample contains instructions for setup.

NetView provides the CNMCNMI service routine for use in communicating with devices in the network through the communications network management interface (CNMI). You can access any data that is returned using the CNMGETD service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN), CNMCNMI (CNMI), CNMGETD (CNMGETDATA), and CNMSMSG (CNMSENDDMSG).

**PKEYIO (CNMS4215)**

This sample illustrates how to code a NetView HLL command processor that allows input/output (I/O) to a VSAM file through the CNMKIO service routine. The command processor must execute on a data services task (DST). To run this
PL/I Samples

command, use the EXCMD command or the CNMSMSG service routine (using type set to COMMAND). The prologue of the sample explains how to set up the command processor.

The HLL service routines used in this sample are CNMKIO (CNMKEYIO) and CNMSMSG (CNMSENDSMSG).

PSCOPCK (CNMS4216)
This sample illustrates the command authorization checking capabilities provided by the NetView program. It authorizes keywords and values using the PSCOPCK command. You must set up the following elements:

- Operator ID
- Operator classes that can access the command
- Operator profile

Refer to the prologue of the sample for more information. This command yields a message if the operator is not authorized to use the keyword and value specified when invoking the command.

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN), CNMSCOP (CNMSCOPECK), and CNMSMSG (CNMSENDSMSG).

PFLVIEW (CNMS4217)
This sample illustrates the use of the full-screen VIEW command processor.

The HLL service routines used in this sample are CNMCMD (CNMCOMMAND) and CNMVARS (CNMVARPOOL).

PACTLU (CNMS4218)
This sample illustrates how to issue a VTAM command to activate a logical unit (LU), trap the VTAM messages that result, and respond depending on the messages received.

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN), CNMCMD (CNMCOMMAND), CNMGETD (CNMGETDATA), and CNMSMSG (CNMSENDSMSG).

PSEQLOG (CNMS4219)
This sample uses CNMSMSG to log text to a sequential log. The prolog of the sample contains instructions for setup.

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN), CNMINFC (CNMINFC), and CNMSMSG (CNMSENDSMSG).

PXITDI (CNMS4220)
This sample illustrates the DST initialization exit that is used by the USERVSAM DST. The HLL service routines used in this sample are CNMVARS (CNMVARPOOL) and CNMSMSG (CNMSENDSMSG).

PXITVN (CNMS4221)
This sample primes a VSAM empty data set for the USERVSAM DST.
PSNDDST (CNMS4222)
This sample sends a PUT or GET request to the sample HLL data services
command processor PDOVSAM to store and retrieve a given value for a specified
key (key and value limited to 11 characters in length). The sample also allows a
specified NetView program command list language variable (defined by the caller)
to be set to the retrieved value.

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN),
CNMSMSG (CNMSENDMSG), CNMVARS (CNMVARPOOL), CNMGETD
(CNMGETDATA), and CNMCMD (CNMCOMMAND).

PDOVSAM (CNMS4223)
This sample is an HLL data services command processor that runs under the
sample data services task (task ID USERVSAM). It processes PUT or GET requests
sent by the PSNDDST sample, and writes or reads an 11-character value associated
with an 11-character key to the sample DST's VSAM data set. The prologue of
PDOVSAM contains instructions on installing the sample USERVSAM data
services task.

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN),
CNMSMSG (CNMSENDMSG), and CNMKIO (CNMKEYIO).

PPRIME (CNMS4224)
This sample primes a VSAM empty data set for PKEYIO.

PHSNDMU (CNMS4226)
This sample sends a software alert to an application over the high performance
transport.

The HLL service routines used are CNMHSMU (CNMHSENDMU) and
CNMSMSG (CNMSENDMSG).

PRODMCON (CNMS4230)
This sample invokes CNMQAPI which enables the user to access a RODM under
the control of NetView. The coded example shows a RODM CONNECT function.

The HLL service routines used in this sample are CNMSMSG (CNMSENDMSG)
and CNMQAPI (CNMOPREP).

PAUTOTB (CNMS4231)
This sample sends an MSU directly to the automation table for evaluation. This
sample provides a method of testing automation table statements without sending
an alert through the hardware monitor.

The HLL service routines used in this sample are CNMAUTO (CNMAUTOTAB),
CNMSMSG (CNMSENDMSG), and CNMSCAN (CNMSSCAN).

PREGSTR (CNMS4232)
This sample registers or deregisters an application as an MS application on an
operations management served application.

You must provide the following:
PL/I Samples

- Application name
- Command name
- Logmode
- Replace option
- Registration type

Refer to the prologue of the sample for more information. This command sends a message to the invoking operator that reports the return code from the registration or deregistration.

The HLL service routines used in this sample are CNMRGS (CNMREGIST), CNMSMSG (CNMSENDMSG), and CNMSCAN (CNMSSCAN).

**PSENDMU (CNMS4233)**

This sample sends a software alert to ALERT_NETOP from an MS application.

The HLL service routines used in this sample are CNMSMU (CNMSENDMU) and CNMSMSG (CNMSENDMSG).

**PHREGSTR (CNMS4236)**

This sample registers an application with the high performance transport.

You must provide the following:
- Application name
- Command name
- Replace option
- Registration type
- Focal point category
- Focal point option
- Notify option

Refer to the prologue of the sample for more information. This command sends a message to the invoking operator that reports the return code from the registration or deregistration.

The HLL service routines used in this sample are CNMHRGS (CNMHREGIST), CNMSMSG (CNMSENDMSG), and CNMSCAN (CNMSSCAN).

**PPRSMDB (CNMS4239)**

This HLL sample sends a Message Data Block (MDB) to NetView for processing. The MDB is an architected structure for message data. The MDB structure is mapped by the MVS control block IEAVM105.

The HLL services used in this sample are CNMPMDB (CNMPRSMDB) and CNMSMSG (CNMSENDMSG).

**PACTPIP (CNMS4305)**

This sample illustrates how to use the PIPE command to issue a VTAM command. The VTAM command activates a logical unit (LU), traps the resulting VTAM messages, and responds depending on the messages received.

The HLL service routines used in this sample are CNMSCAN (CNMSSCAN), CNMVAR (CNMVARPOOL), CNMCM (CNMCOMMAND), and CNMSMSG (CNMSENDMSG).
Appendix C. C Language Control Blocks and Include Files

This appendix describes the C control blocks and include files needed to write command processors and installation exits in the C language.

DSIC Include File

DSIC is required and must be included by all HLL programs written in C. DSIC includes all of the external HLL control blocks and include files needed to run C programs in the NetView environment.

```c
#include "dsiccons.h" /* Constants */
#include "dsicvarc.h" /* Varying length char structure */
#include "dsichlb.h" /* Mapping of HLB */
#include "dsicorig.h" /* Mapping of Origin block */
#include "dsicprm.h" /* NetView bridge parameter block */
#include "dsiccall.h" /* HLL function definitions */
#include "dsiccnm.h" /* HLL return code constants */
```
This file contains the definitions for constants that are helpful when coding HLL modules in C.

```c
#define ZERO 0x00
#define TRAPQ 1
#define OPERQ 2
#define DATAQ 3
#define IDATAQ 4
#define CNMIQ 5
#define MDSMUQ 6
```

Constants for calls to Cnmaltd

```c
#define INSLINE "INSLINE 
#define REPLINE "REPLINE 
#define DELLINE "DELLINE 
```

Constants for Cnmcpys and Cnmsubs
/* Constants for calls to Cmgetd */
#define GETFIRST "GETFIRST"
#define GETMSG "GETMSG 
#define GETNEXT "GETNEXT 
#define GETLINE "GETLINE 
#define PEEKBFR "PEEKBFR 
#define PEEKLINE "PEEKLINE" 
#define FLUSHNXT "FLUSHNXT" 
#define FLUSHLIN "FLUSHLIN" 
#define FLUSHGRP "FLUSHGRP" 
#define FLUSHMSG "FLUSHMSG" 
#define FLUSHQ "FLUSHQ "

/* Constants for Cnmnmi */
#define SENDRESP "SENDRESP"
#define SENDRPLY "SENDRPLY"

/* Constants for Cnmlock */
#define UNLOCK "UNLOCK 
#define LOCK "LOCK 
#define TEST "TEST 
#define WAIT "WAIT 
#define NOWAIT "NOWAIT 

/* Constants for Cnmams, Cmmpool and Cnmcell */
#define ALLOC "ALLOC 
#define FREE "FREE 
#define LOCATE "LOCATE 
#define REALLOC "REALLOC 

#define RESIDENT 0 
#define STORAG31 1 
#define STORAG24 2 

/* Constants for Cnmmsmsg */
#define MSG "MSG 
#define MSG_C "MSG_C " 
#define MSG_L "MSG_L 
#define MSG_D "MSG_D 
#define MSG_E "MSG_E 
#define MSG_F "MSG_F 
#define COMMAND "COMMAND 
#define REQUEST "REQUEST 
#define DATA "DATA 
#define OPER "OPER 
#define TASK "TASK 
#define SYSOP "SYSOP 
#define NETVLOG "NETVLOG 
#define EXTLLOG "EXTLLOG 
#define SEQLOG "SEQLOG 
#define AUTHRCV "AUTHRCV 
#define OPCLASS "OPCLASS 

Appendix C. C Language Control Blocks and Include Files 329
#define NULLCHAR " "

/**************************************************************/
/* Constants for Cnmvars */
/**************************************************************/
#define PUT "PUT 
#define DCL "DCL 
#define GET "GET 
#define LOCAL "LOCAL 
#define TGLOBAL "TGLOBAL "
#define CGLOBAL "CGLOBAL "
#define CALLER "CALLER 

/**************************************************************/
/* Constants for Cnmkio */
/**************************************************************/
#define GET_EQ "GET_EQ "
#define GET_EH "GET_EH "
#define GET_NEXT "GET_NEXT"
#define GET_PREV "GET_PREV"
#define ERASE "ERASE 
#define ENDEQ "ENDEQ 
#define UPDATE "UPDATE "
#define NOUPDATE "NOUPDATE"
#define DIRECT "DIRECT 

/**************************************************************/
/* Constants for Cnmgetp */
/**************************************************************/
#define BYHEADER 'H'
#define BYNAME 'P'
#define BYNEXT 'N'

/**************************************************************/
/* Constants for Cnmc2t */
/**************************************************************/
#define SNAALERT "SNAALERT"
#define SNACAUSE "SNACAUSE"
#define SNAUCAUS "SNAUCAUS"
#define SNAICAUS "SNAICAUS"
#define SNAFCAUS "SNAFCAUS"
#define SNAREACT "SNAREACT"
#define SNADDATA "SNADDATA"

/**************************************************************/
/* Constants for installation exits running under a DST */
/**************************************************************/
#define USERASIS 0
#define USERDROP 4
#define USERSWAP 8
#define USERLOG 12
#define USERLOGR 16
#define USERHCL 20
#define USERHCLR 24
#define USERDINT 233
#define USERVINT 234
#define USERVINP 235
#define USERVOUT 236
#define USERCINP 237
#define USERCOUT 238
#define USERXLOG 240
#define USERBINT 241
#define USERBOUT 242

/**************************************************************/
/* Constants for Cnmvlc and Cnmnvlc */
/**************************************************************/
DSICVARC Structure Type

DSICVARC is a structure type that represents varying length character strings for use in NetView HLL service routine invocations.

/**/ /* DSICVARC Structure Type */

DSIVARCH is a structure type that represents varying length character strings for use in NetView HLL service routine invocations.
DSICVARC

/*
 CONCATENATION = DSICVARCH is a structure type which represents
 varying length character strings for use in NetView High
 Level Language service routine invocations.
 The structure consists of two parts:
 short int size - A 2 byte field which holds the size of
 the character string. The end of string character (\0) is not included in this
 size but MUST delimit the character string.
 char *buffer - A character string delimited by the end
 of string character (\0).
 NOTES = see below
 DEPENDENCIES = none
 RESTRICTIONS = none
 REGISTER CONVENTIONS = not applicable
 PATCH LABEL = not applicable
 MODULE TYPE = structure map
 PROCESSOR = C
 EXTERNAL REFERENCES = none
 CHANGE ACTIVITY
 typedef struct {
  short int size; /* Length of buffer */
  char buffer[256]; /* Varying length buffer */
 } Dsivarch;

DSICHLB Control Block

This file contains a C mapping of the NetView HLL control block (HLB). The HLB is built during command processor initialization and exists for the lifetime of the command processor.

/*
 NAME = DSICHLB
 DESCRIPTIVE NAME = HLL C Mapping of DSIHLB

Licensed Materials - Property of IBM
(C) Copyright IBM Corp. 1989, 1997. All rights reserved.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*/
DSICHLB

This file maps the origin block of the request that initiated the current procedure.

DSICORIG Control Block

This file maps the origin block of the request that initiated the current procedure.
DSICORIG

/*
  * FUNCTION = This file defines the mapping of the origin block of the request that caused the execution of the procedure currently running.
  *
  * NOTES = see below
  *
  * DEPENDENCIES = none
  *
  * RESTRICTIONS = none
  *
  * REGISTER CONVENTIONS = not applicable
  *
  * PATCH LABEL = not applicable
  *
  * MODULE TYPE = structure map
  *
  * PROCESSOR = C
  *
  * EXTERNAL REFERENCES = none
  *
  * CHANGE ACTIVITY
  */

typedef struct {
    int Orig_block_length;
    char Orig_dummy1[8]; /* Reserved */
    char Orig_domain[8]; /* Origin domain id */
    char Orig_task[8]; /* Origin task id */
    char Orig_process[8];
    char Orig_msg_type; /* Message type from HDRMTYPE */
    char Orig_line_type; /* Line type */
    char Orig_dummy2[2]; /* Reserved */
} Dsiorig;

DSICPRM Control Block

This file defines the NetView Bridge parameter control block for C.

getStatusCode

/* Licensed Materials - Property of IBM */
/* 5697-B82 */
/* (C) Copyright IBM Corp. 1990, 1997. All rights reserved. */
/* */
/* US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/* */

/status = NetView Version 2 Release 2
/* */
/* function = This file defines the NetView Bridge Parameter Block for C */
/* */
/* notes = see below */
/* */
/* dependencies = none */
/* */
/* restrictions = none */
typedef struct dsicprm {
    struct dsicprm *prmlink;
    void *prmptr;
    unsigned short int prmnaml;
    char prmname[31];
    short int prmindex;
    unsigned short int prmtype;
    unsigned short int prmleng;
} Dsicprm;

#define prm_ebdc 0X00ee
#define prm_int 0X00bb

DSICCALL Control Block

This file defines the service routines for C.

//*************************************************************/
/* NAME = DSICCALL */
/* DESCRIPTIVE NAME = HLL C Service Routine Definitions */
/* Licensed Materials - Property of IBM */
/* 5697-B82 */
/* (C) Copyright IBM Corp. 1989, 1997. All rights reserved. */
/* US Government Users Restricted Rights - Use, duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
/* STATUS = NetView Version 3 Release 2 */
/* FUNCTION = This files defines the following service routines */
/* for C: */
/* 1. Preprocessor directives */
/* 2. Function declarations */
/* 3. Macro definitions */
/* NOTES = see below */
/* DEPENDENCIES = none */
/* RESTRICTIONS = none */
/* REGISTER CONVENTIONS = not applicable */
/* PATCH LABEL = not applicable */
/* MODULE TYPE = structure map */
/* PROCESSOR = C */
/* EXTERNAL REFERENCES = none */
/* CHANGE ACTIVITY */

/********************************************************************/
 ifndef __cplusplus
 /********************************************************************/
 /* Function declarations */
 /* Each of the HLL service routines has a function declaration */
 /* defining its parameter list. */
 /* Each of the HLL service routines uses OS linkage. */
 /********************************************************************/
 extern "OS" void Cnmaltd(Dsihlb **hlbptr, char *adfunc, void *adbuf,
 void *adorigin, int adqueue, int adindex);
 extern "OS" void Cnmcell(Dsihlb **hlbptr, char *pcfunc, int pctoken,
 void *pcstrptr);
 extern "OS" void Cnmcnmi(Dsihlb **hlbptr, char *cnfunc, void *cndata,
 char *cndest, int cntimout);
 extern "OS" void Cnmcpys(Dsihlb **hlbptr, void *csfrom,
 void *csto, int cslen,
 char *cstype);
 extern "OS" void Cnmgeta(Dsihlb **hlbptr, char *ganame, void *gadata,
 int gdatlen, int gqueue);
 extern "OS" void Cnmgetd(Dsihlb **hlbptr, char *gdfunc, void *gdbuf,
 int gdbuflen, void *gdorigin, int gdqueue, int gdindex);
 extern "OS" void Cnminfc(Dsihlb **hlbptr, char *icname, void *icdata,
 int idatlen);
 extern "OS" void Cnminfi(Dsihlb **hlbptr, char *iiname, int *inumb);
 extern "OS" void Cnmkio(Dsihlb **hlbptr, char *vsfunc, void *vsdata,
 int vsdatlen, void *vskey, char *vsoption);
 extern "OS" void Cnmlk(Dsihlb **hlbptr, char *lkfunc, void *lkname,
 char *lkscope, char *lkoption);
 extern "OS" void Cnmmemo(Dsihlb **hlbptr, int *motoken, char *moddname,
 char *momemnam);
 extern "OS" void Cnmmemr(Dsihlb **hlbptr, int mrtoken, void *mrdata,
 int mrdatlen, char *incl);
 extern "OS" void Cnmmemc(Dsihlb **hlbptr, int mctoken);
 extern "OS" void Cnnnams(Dsihlb **hlbptr, char *nsfunc, void *nsptr,
 void *nsname, int *nsleng, int nsclass);
 extern "OS" void Cmmpool(Dsihlb **hlbptr, char *spfunc,
 int *sptoken, void *spname,
 int spleng, int *spprcnt, int *spseccnt, int *spclass);
 extern "OS" void Cmnsfld(Dsihlb **hlbptr, char *scmd,
 char *sckwd, char *scvalue);
 extern "OS" void Cmnmsg(Dsihlb **hlbptr, void *smtex, 
 char *smsgtype, 
 char *smdestyp, char *smdestid);
 extern "OS" void Cnmvars(Dsihlb **hlbptr, char *cvfunc,
 void *cvdata,
 int cvdatlen, void *cvname, char *cvpool);
 extern "OS" void Cmnsndt(Dsihlb **hlbptr, char *strtype,
 char *stdestnet, char *stdestdom, char *stdesttask,
 char *stvocab, char *stcorr,
 Dsicprm **stparms);
 extern "OS" void Cmgetp(Dsihlb **hlbptr, char *gmpmethod,
 char *gpmname,
void *gpnamep, int *gpindex, void *gpdata,
int gdatlen, short *gptype, int gpqueue,
int *gpcursor);

extern "OS" void Cnm2t(Dsihlb **hlbptr, void *trdata, int trdatlen,
char *trtable, int trcode);

extern "OS" void Cnmsmu(Dsihlb **hlbptr,
char *sudtype,
void *sudata,
void *susupcor,
void *susorar,
int sutmout,
char *susynch,
char *surepcp,
char *suoappl,
char *sudnet,
char *sudlu,
char *sudappl,
int sumutype,
char *supri);

extern "OS" void Cnmmgs(Dsihlb **hlbptr, char *rgtype,
char *rgappl, char *rgcmd,
char *rgfpcat, char *rgfocpt, char *rgrepl,
char *rgmotify, char *rgpri);

extern "OS" void Cnmauto(Dsihlb **hlbptr, void *data);

extern "OS" void Cnmhrgs(Dsihlb **hlbptr, char *hrtype,
char *hrapplname,
char *hrcmdname, char *hrlogmode, char *hrreplvalue,
char *hrnotfy, char *hrpri);

extern "OS" void Cnmhsmu(Dsihlb **hlbptr, char *hsdatatype,
void *hsdata,
void *hs Supbcorrel, void *hsorrelarea,
int hstimeout, char *hssynch,
char *hssrepycmd, char *hsorigappl, char *hsdestnet,
char *hsdestlu, char *hsdestappl, int hsmutype,
char *hspri);

extern "OS" void Cnmqapi(Dsihlb **hlbptr, void *qaacb, void *qatif,
void *qaresp, void *qfunc, char *qawaitf,
short *qawaitt);

extern "OS" void Cnmpmdb(Dsihlb **hlbptr, void *mdmdb,
void *mdsource, char *mdcorr);

extern "OS" void Cnmsubs(Dsihlb **hlbptr, void *sssource,
void *sstarget,
int sslen, char *sstype);

/**********************************************************************/
/* Function declarations */
/* Functions provided for use with varying length character */
/* strings. */
/**********************************************************************/
extern "C" void *Cnmvlc(void *vstring, short convert,
char *istring, ...);
extern "C" void *Cnmnvlc(void *vstring, short convert,
int length, char *istring);

#else

/**********************************************************************/
/* Preprocessor directives */
/* Each of the HLL service routines uses OS linkage. */
/**********************************************************************/
#pragma linkage(Cnmaltld, OS)
#pragma linkage(Cnmcell, OS)
#pragma linkage(Cnmcmd, OS)
#pragma linkage(Cnmcmni, OS)
#pragma linkage(Cnmcpsy, OS)
#pragma linkage(Cnmgeta, OS)
#pragma linkage(Cnmgetd, OS)
DSICCALL

#pragma linkage(Cnminfc, OS)
#pragma linkage(Cnmnfini, OS)
#pragma linkage(Cnmkio, OS)
#pragma linkage(Cnmlk, OS)
#pragma linkage(Cnmmemr, OS)
#pragma linkage(Cnmnmem, OS)
#pragma linkage(Cnmnams, OS)
#pragma linkage(Cnmpool, OS)
#pragma linkage(Cnmssps, OS)
#pragma linkage(Cnmsmsg, OS)
#pragma linkage(Cnmvars, OS)
#pragma linkage(Cnmsndt, OS)
#pragma linkage(Cnmpool, OS)
#pragma linkage(Cnmsmu, OS)
#pragma linkage(Cnmrgs, OS)
#pragma linkage(Cnm_auto, OS)
#pragma linkage(Cnmhrgs, OS)
#pragma linkage(Cnmhsmu, OS)
#pragma linkage(Cnmqapi, OS)
#pragma linkage(Cnmpmdb, OS)
#pragma linkage(Cnmsubs, OS)

/********************************************************************/
/* Function declarations */
/* Each of the HLL service routines has a function declaration */
/* defining its parameter list. */
/********************************************************************/
void Cnmaltd(Dsihlb **hlbptr, char *adfunc, void *adbuf,
 void *adorigin, int adqueue, int adindex);
void Cnmcell(Dsihlb **hlbptr, char *pcfunc, int pctoken,
 void *pcstrptr);
void Cnmcmd(Dsihlb **hlbptr, void *cmdstr);
void Cnmcmdmi(Dsihlb **hlbptr, char *cnfunc, void *cndata,
 char *cndest, int cntimout);
void Cnmcpys(Dsihlb **hlbptr, void *csfrom, void *csto, int cslen,
 char *ctype);
void Cnmgeta(Dsihlb **hlbptr, char *ganame, void *gadata,
 int gadatlen, int gqueue);
void Cnmgetd(Dsihlb **hlbptr, char *gdfunc, void *gdbuf,
 int gdbuflen, void *gdorigin, int gdqueue, int gdindex);
void Cnmninf(Dsihlb **hlbptr, char *icname, void *icdata,
 int icdatlen);
void Cnmnfini(Dsihlb **hlbptr, char *iiname, int *iinumb);
void Cnmkio(Dsihlb **hlbptr, char *vsfunc, void *vsdata,
 int vsdatlen, void *vskey, char *vsoption);
void Cnmkio(Dsihlb **hlbptr, char *lkfunc, void *lkname,
 char *lkscope, char *lkoption);
void Cnmnams(Dsihlb **hlbptr, char *nsfunc, void *nsptr,
 char *nsname, int *nslen, int nsclass);
void Cnmpool(Dsihlb **hlbptr, int *mrtoken, void *mrdata,
 int mrdatlen, char *incl);
void Cnmmemr(Dsihlb **hlbptr, int *mrtoken, void *mrddata,
 int mrdatlen, char *incl);
void Cnmnmemc(Dsihlb **hlbptr, int mctoken);
void Cnmnams(Dsihlb **hlbptr, char *nsfun, void *nsptr,
 void *nsname, int *nsleng, int nsclass);
void Cnmninf(Dsihlb **hlbptr, char *spfunc, int *spoken, void *spname,
 int splen, int spprcnt, int spsccnt, int spclass);
void Cnmnmsg(Dsihlb **hlbptr, char *sscmd, char *scwd, char *scvalue);
void Cnmnmsend(Dsihlb **hlbptr, char *strtype, char *strtype,
 char *stdestnet, char *stdestdom, char *stdesttask,
 char *stvocab, char *stcor, char *stvocabulary, char *stvar);
DSICPRM **stparm);
void Cnmgetp(Dsihlb **hlbptr, char *gpmethod, char *gpnameh,
    void *gpnamep, int *gpindex, void *gpdata,
    int gpdatlen, short *gptype, int gpqueue,
    int *gpcursor);
void Cnmc2t(Dsihlb **hlbptr, void *trdata, int trdatlen,
    char *trtable, int trcode);
void Cnmsmu(Dsihlb **hlbptr,
    char *sudtype,
    void *sudata,
    void *susupcor,
    void *sucorrar,
    int sutmeout,
    char *susynch,
    char *surepcp,
    char *suoaapl,
    char *sudnet,
    char *sudlu,
    char *sudappl,
    int sumutype,
    char *supri);
void Cnmmrgs(Dsihlb **hlbptr, char *rgtype, char *rgappl,
    char *rgcmd, 
    char *rgfpcat, char *rgfocpt, char *rgrepl,
    char *rgmotty, char *rgpri);
void Cnmamauto(Dsihlb **hlbptr, void *data);
void Cnmhrgs(Dsihlb **hlbptr, char *hrtype, char *hrapplname,
    char *hrcmdname, char *hrlogmode, char *hrreplvalue,
    char *hrmotfy, char *hrpri);
void Cnmhsmu(Dsihlb **hlbptr, char *hsdatatype, void *hsdata,
    void *hssupcorrel, void *hscorrelarea,
    int hstimeout, char *hssynch,
    char *hspocmd, char *hsorigappl, char *hsdestnet,
    char *hsdestapl, int hsmutype,
    char *hspri);
void Cnmqapi(Dsihlb **hlbptr, void *qaach, void *qatif,
    void *qaresp, void *qafunc, char *qawaitf,
    short *qawaitt);
void Cnmpmdb(Dsihlb **hlbptr, void *mdmdb,
    void *mdsource, char *mdcorr);
void Cnmsubs(Dsihlb **hlbptr, void *sssource, void *sstarget,
    int sslen, char *sstype);

/********************************************************************/
/* Function declarations */
/* Functions provided for use with varying length character */
/* strings. */
/********************************************************************/
#endif

/********************************************************************/
/* Macro definitions */
/* Each of the HLL service routines has a macro definition which */
/* inserts the HLB pointer (&Hlbptr) into the parameter list. */
/********************************************************************/
#define Cnmaltd(adfunc,adbuf,adorigin,adqueue,adindex) \
    Cnmaltd(&Hlbptr,adfunc,adbuf,adorigin,adqueue,adindex)
#define Cnmcell(pcfunc,pctoken,pcstrptr) \
    Cnmcell(&Hlbptr,pcfunc,pctoken,pcstrptr)
#define Cnmcnmi(cnfunc,cndata,cndest,cntimout) \
    Cnmcnmi(&Hlbptr,cnfunc,cndata,cndest,cntimout)
#define Cnmcpys(csfrom,csto,cslen,cstype) \
    Cnmcpys(&Hlbptr,csfrom,csto,cslen,cstype)
#define Cnmgeta(ganame,gadata,gadatlen,gaqueue) \
    Cnmgeta(&Hlbptr,ganame,gadata,gadatlen,gaqueue)
DSICCALL

Cnmgeta(&Hlbptr, ganame, gadata, gadatlen, gqueue)
#define Cnmgeta(gdfunc, gdbuf, gdbuflen, gdorigin, 
gqueue, gindex)
Cnmgeta(&Hlbptr, gdfunc, gdbuf, gdbuflen, gdorigin, 
gqueue, gindex)
#define Cnminfc(iename, icdata, icdatlen)
Cnminfc(&Hlbptr, iename, icdata, icdatlen)
#define Cnminfi(iename, inumb)
Cnminfi(&Hlbptr, iename, inumb)
#define Cnmkio(vsfunc, vsdata, vsdatlen, vskey, vsoption)
Cnmkio(&Hlbptr, vsfunc, vsdata, vsdatlen, vskey, vsoption)
#define Cnmkio1(lkfunc, lkname, lkscope, lkoption)
Cnmkio1(&Hlbptr, lkfunc, lkname, lkscope, lkoption)
#define Cnmkio2(motoken, moddname, momemnam)
Cnmkio2(&Hlbptr, motoken, moddname, momemnam)
#define Cnmmemr(mrtoken, mrdatalen, mrdatlen, incl)
Cnmmemr(&Hlbptr, mrtoken, mrdatalen, incl)
#define Cnmmemc(mctoken)
Cnmmemc(&Hlbptr, mctoken)
#define Cnmpool(spfunc, sptoken, spname, spleng, 
spprice, spsectc, spclass)
Cnmpool(&Hlbptr, spfunc, sptoken, spname, spleng, 
spprice, spsectc, spclass)
#define Cnmsmsg(smtext, smmsgtype, smdestyp, smdestid)
Cnmsmsg(&Hlbptr, smtext, smmsgtype, smdestyp, smdestid)
#define Cnmvars(cvfunc, cvdata, cvdatlen, cvname, cvpool)
Cnmvars(&Hlbptr, cvfunc, cvdata, cvdatlen, cvname, cvpool)
#define Cnmgetp(gpmethod, gpname, gpdata, 
gpdatlen, gpindex, gpqueue, gpcursor)
Cnmgetp(&Hlbptr, gpmethod, gpname, gpdata, 
gpdatlen, gpindex, gpqueue, gpcursor)
#define Cnmgetp1(trdata, trdatlen, trtable, trcode)
Cnmgetp1(&Hlbptr, trdata, trdatlen, trtable, trcode)
#define Cnmgetp2(trdata, trdatlen, trtable, trcode)
Cnmgetp2(&Hlbptr, trdata, trdatlen, trtable, trcode)
#define Cnmmsmu(sudtype, sudata, susupcor, sucorrar, 
sutimeout, sussynch, surepcp, suappl, 
sudnet, sudlu, sudappl, sudtyp, sudpri)
Cnmmsmu(&Hlbptr, sudtype, sudata, susupcor, sucorrar, 
sutimeout, sussynch, surepcp, suappl, 
sudnet, sudlu, sudappl, sudtyp, sudpri)
#define Cnmrgs(rgtype, rgappl, rgcmd, 
grpcat, rgocpt, rgreppl, rgnotify, rgpri)
Cnmrgs(&Hlbptr, rgtype, rgappl, rgcmd, 
grpcat, rgocpt, rgreppl, rgnotify, rgpri)
#define Cnmauto(data)
Cnmauto(&Hlbptr, data)
#define Cnmmrgs(hrtype, hrappname, hrappcmdname, 
hrlogmode, hrrepplvalue, hrnotfy, hrpri)
Cnmmrgs(&Hlbptr, hrtype, hrappname, hrappcmdname, 
hrlogmode, hrrepplvalue, hrnotfy, hrpri)
#define Cnmhsmu(hsdatatype, hsdatalen, hssuppcorrel, 
hscorelarea, hstimeout, hssynch, 
hsrplcmd, hssorigappl, hsddestnet, hsdstl, 
hsdstatpl, hsmutype, hspr)
Cnmhsmu(&Hlbptr, hsdatatype, hsdatalen, hssuppcorrel, 
hscorelarea, hstimeout, hssynch, 
hsrplcmd, hssorigappl, hsddestnet, hsdstl, 
hsdstatpl, hsmutype, hspr)
#define Cnmqapi(qaacb, qatif, qaresp, qafunc, qawaitf, qawaitt)
Cnmqapi(&Hlbptr, qaacb, qatif, qaresp, qafunc, qawaitf, qawaitt)
DSICCNM Control Block

This file defines the HLL return codes for C.

.BLLBBB
  /********************************************************************/
  */
  */  NAME = DSICCNM
  */
  */  DESCRIPTIVE NAME = HLL C Return Codes
  */
  */
  /********************************************************************/
  */
  */ Licensed Materials - Property of IBM
  */
  */  5697-B82
  */
  */  (C) Copyright IBM Corp. 1989, 1997. All rights reserved.
  */
  */
  */ US Government Users Restricted Rights - Use, duplication or
  */ disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
  */
  */
  /********************************************************************/
  */
  */ STATUS = NetView Version 3 Release 2
  */
  */
  */ FUNCTION = This file defines the HLL return codes for C.
  */
  */
  */ NOTES = see below
  */
  */
  */ DEPENDENCIES = none
  */
  */
  */ RESTRICTIONS = none
  */
  */
  */ REGISTER CONVENTIONS = not applicable
  */
  */
  */ PATCH LABEL = not applicable
  */
  */
  */ MODULE TYPE = constants
  */
  */
  */ PROCESSOR = C
  */
  */
  */ EXTERNAL REFERENCES = none
  */
  */
  */ CHANGE ACTIVITY
  */
  */
  /********************************************************************/
  
#define CNM_GOOD 0
#define CNM_BAD_INVOCATION 4
#define CNM TOO_MANY 8
#define CNM.BAD SYNTAX 12
#define CNM.BAD DDNAME 16
#define CNM.NOT FOUND 20
#define CNM.NO STORAGE 24
#define CNM.IOERROR 28
#define CNM.BAD TOKEN 32
#define CNM.END FILE 36
#define CNM.DATA TRUNC 40
#define CNM.NOT IN ASYNC 44
#define CNM.BAD RULENG 48
#define CNM.BAD FUNC 52
#define CNM.BAD TIMEOUT 56
#define CNM.NEED PRID 60
#define CNM.NEG_RESPONSE 64
DSICCNM

#define CNM_TIME_OUT 68
#define CNM_BAD_QUEUE 72
#define CNM_BAD_INDEX 76
#define CNM_QUEUE_EMPTY 80
#define CNM_BAD_ORIGINBLOCK 84
#define CNM_BAD_LENGTH 88
#define CNM_BAD_SOURCE 92
#define CNM_BAD_LINE 96
#define CNM_NO_CUR_LINE 100
#define CNM_DUPL_NAME 104
#define CNM_BAD_NAME 108
#define CNM_BAD_CLASS 112
#define CNM_BAD_MSGTYP 116
#define CNM_BAD_DESTYP 120
#define CNM_TYP_CONFLICT 124
#define CNM_BAD_OPTION 128
#define CNM_COMMAND_NA 132
#define CNM_KEYWORD_NA 136
#define CNM_VALUE_NA 140
#define CNM_BAD_COMMAND 144
#define CNM_BAD_KEYWORD 148
#define CNM_NO_TRAP 152
#define CNM_BAD_POOL 156
#define CNM_BAD_ADDR 160
#define CNM_BAD_TASKNAME 164
#define CNM_BAD_MODNAME 168
#define CNM_BAD_ID 172
#define CNM_BAD_COMBO 176
#define CNM_TVBB_INUSE 180
#define CNM_RID_INUSE 184
#define CNM_RID_SELF 188
#define CNM_BAD_PRI_COUNT 192
#define CNM_BAD_SEC_COUNT 196
#define CNM_DUPL_KEY 200
#define CNM_NOT_IN_POOL 204
#define CNM_LOCKED 208
#define CNM_LOCK_INUSE 212
#define CNM_LOG_INACTIVE 216
#define CNM_TASK_INACTIVE 220
#define CNM_TIME_OUT_WAIT 224
#define CNM_GO_ON_WAIT 228
#define CNM_MSG_ON_WAIT 232
#define CNM_OP_INPUT_ON_WAIT 236
#define CNM_DATA_ON_WAIT 240
#define CNM_NO_TRAP_SET 244
#define CNM_NO_PREV_WAIT 248
#define CNM_BAD_CSTYPE 252
#define CNM_BAD_CELL_ADDRESS 256
#define CNM_CELL_ALREADY_FREE 260
#define CNM_NO_LINES_IN_MSG 264
#define CNM_NO_DATA 268
#define CNM_INVALID_MESSAGE 272
#define CNM_PARM_MISSING 276
#define CNM_NO_MORE_DATA 280
#define CNM_BAD_CURSOR 284
#define CNM_REQ_TOO_LONG 288
#define CNM_BAD_TASK 292
#define CNM_BAD_TRAN_HDR 296
#define CNM_PROCESSING_ERROR 300
#define CNM_USER_ENDED 304
#define CNM_DST_ENDED 308
#define CNM_BAD_TABLE 312
#define CNM_BAD_CODE 316
#define CNM_BAD_SSTYPE 320
#define CNM_BAD_DATA_TYPE 400
#define CNM_BAD_DATA 404
#define CNM_SAME_APPL 408
#define CNM_SYNCH_NOT_COMP 412
#define CNM_OAPPL_NOT_REG 416
#define CNM_BAD_SAPPL 420
#define CNM_BAD_UOW 424
#define CNM_BAD_RTI 428
#define CNM_BAD_OAN 432
#define CNM_BAD_DAN 436
#define CNM_BAD_DAPPL 440
#define CNM_BAD_DNETID 444
#define CNM_BAD_DLU 448
#define CNM_BAD_DAPPL 452
#define CNM_BAD_OII 456
#define CNM_BAD_REPLY 460
#define CNM_BAD_MUTYPE 464
#define CNM_BAD_SYNCH 468
#define CNM_BUSY 472
#define CNM_BAD_APPL_NAME 476
#define CNM_APP_NAME_RSTD 480
#define CNM_BAD_FPCAT_NAME 484
#define CNM_BAD_FPCAT_CHOICE 488
#define CNM_BAD_REG_TYPE 492
#define CNM_BAD_FOCALPT_VALUE 496
#define CNM_CANT_BE_FOCALPT 500
#define CNM_BAD_REPLACE_VALUE 504
#define CNM_SYNCH_CMD_MISSING 508
#define CNM_BAD_PRI_VALUE 512
#define CNM_SYNTAX_ERROR 520
#define CNM_INVAL_MEMBER 524
#define CNM_INVAL_NEST 528
#define CNM_SYSTEM_ERROR 532
#define CNM_IMBED_IOERROR 536
#define CNM_BAD_INCLUDE 540
#define CNM_BAD_COUNT 544
#define CNM_SAME_OMAPPL 548
#define CNM_REQ_CANCELED 552
#define CNM_TASK_NO_AUTH 556
#define CNM_BAD_OM_MDSEM 560
#define CNM_BAD_SEQUENCE 564
#define CNM_BAD_MDB 568
#define CNM_BAD_SOURCE_OBJ 572
#define CNM_BAD_LOGMODE 576
#define CNM_LOGMODE_REQUIRED 580
#define CNM_SNACR_MISSING 584
#define CNM_BAD_NOTIFY 588
#define CNM_NO_REPOS 592
#define CNM_BAD_MQS 596
#define CNM_API_FAILURE 600
#define CNM_NETID_UNINIT 604
#define CNM_INVAL_PARMS 608
#define CNM_STRG_FAILURE 612
#define CNM_TIMEOUT 616
#define CNM_TIMEDOUT 620
#define CNM_SAF_UNEXPECTED 624
#define CNM_KWRD_VAL_NA 628
#define CNM_BAD_SEC_ENVIR 632
#define CNM_TBL_UNEXPECTED 636
#define CNM_NO_SEC_INFO 640
#define CNM_NO_OP_INFO 644
#define CNM_BAD_MQS 1000
#define CNM_DST_FAILURE 2000
#define CNM_BAD_EXCMS 3000
#define CNM_BAD_PUSH 4000
#define CNM_BAD_POP 5000
#define CNM_BAD_WLS 6000
#define CNM_BAD_PSS 7000

Appendix C. C Language Control Blocks and Include Files 343
DSICCNM

#define CNM_BAD_WTO 8000
#define CNM_BAD_CES 9000
#define CNM_BAD_DKS 10000
#define CNM_BAD_KVS 11000
#define CNM_BAD_LOAD 12000
#define CNM_BAD_LCS 13000
#define CNM_BAD_CDS 14000
#define CNM_BAD_ESTAE 15000
#define CNM_BAD_PAS 16000
#define CNM_BAD_SNTXS 17000
#define CNM_BAD_MRBLD 18000

#define CNM_BAD_ZCSMS 20000
#define CNM_BAD_ENQ 21000
#define CNM_BAD_TRAN_PARM 22000
#define CNM_FAIL_TO_AUTOMATE 24000
#define CNM_BAD_PRS 25000
#define CNM_BAD_OPR 25000

#define CNM_BAD_ZVSMS 100
Appendix D. C Samples

This appendix contains a table of the C samples that are shipped with the NetView program in NETVIEW.V5R1M0.CNMSAMP. When a data set name is referred to in this appendix, two names are given, for example CTMPPLT (CNMS4201). The first name is the alias name, and the name in parentheses is in the NetView samples library. You can use either name to access the samples. DSICMD has definitions for the alias names to enable those names to be entered as commands.

To enter the member names as commands:
1. Compile and link-edit the samples using the alias name.
2. Delete the * in column 1 of the appropriate CMDMDL statement in DSICMD 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.

Notes:
1. Refer to the prolog of the samples for information about how certain samples are related and special cases for installation exit routines.
2. Each alias name for C begins with the letter C.

This appendix also contains a description of each sample, and coded samples of an installation exit routine and two command processors.

C Language Samples Table

The following table refers to the C language samples that are shipped with NetView. The table contains the function, the alias name, and the name of the member in NETVIEW.V5R1M0.CNMSAMP.

<table>
<thead>
<tr>
<th>Sample Function</th>
<th>C Alias</th>
<th>Sample CNMSAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template for commands and installation exit routines</td>
<td>CTMPPLT</td>
<td>CNMS4201</td>
</tr>
<tr>
<td>Sample DSIEX03 to set a global variable</td>
<td>CEXIT3</td>
<td>CNMS4240</td>
</tr>
<tr>
<td>Uses Cnmsmsg to send data</td>
<td>CSNDDAT</td>
<td>CNMS4241</td>
</tr>
<tr>
<td>Uses WAIT FOR DATA</td>
<td>CWATDAT</td>
<td>CNMS4242</td>
</tr>
<tr>
<td>Sample DSIEX02A changes a WTO to an MLWTO</td>
<td>CEXIT2A</td>
<td>CNMS4243</td>
</tr>
<tr>
<td>Uses Cnmcnmi to forward RUs to a PU</td>
<td>CCNMI</td>
<td>CNMS4244</td>
</tr>
<tr>
<td>Uses Cnmkio for I/O to VSAM</td>
<td>CKEYIO</td>
<td>CNMS4245</td>
</tr>
<tr>
<td>HLL command using Cnmscop for command authorization checking</td>
<td>CSCOPCK</td>
<td>CNMS4246</td>
</tr>
<tr>
<td>Display full screen VIEW panel</td>
<td>CFLVIEW</td>
<td>CNMS4247</td>
</tr>
<tr>
<td>Activates LU and uses TRAP and WAIT to determine if activation is successful</td>
<td>CACTLU</td>
<td>CNMS4248</td>
</tr>
<tr>
<td>Uses Cnmsmsg to log text to a sequential log</td>
<td>CSEQLOG</td>
<td>CNMS4249</td>
</tr>
<tr>
<td>DST initialization exit for USERVSAM DST</td>
<td>CXITDI</td>
<td>CNMS4250</td>
</tr>
<tr>
<td>Primes VSAM empty data set for USERVSAM DST</td>
<td>CXITVN</td>
<td>CNMS4251</td>
</tr>
</tbody>
</table>
C Language Samples Descriptions

Each sample includes a description of the function and the HLL service routines used.

CTMPPLT (CNMS4201)

A template for commands and installation exit routines in C. It is included in "Chapter 9. C High-Level Language Services“ on page 133.

CEXIT3 (CNMS4240)

A sample DSIEX03 that sets a task global variable. This global variable contains the value of the last time that a command other than CSNDDAT was entered under an operator station task (OST). CWATDAT and CSNDDAT are used to interrogate this value. The HLL service routines used in this sample are Cnminfc and Cnmsmsg.

CSNDDAT (CNMS4241)

Uses Cnmsmsg to send data. It is part of an example of sending messages containing a request, waiting on the response, and parsing the results.

The example finds the last time that a command was entered on a given OST. A task global variable is set by CEXIT3 every time a command is entered on an OST. CWATDAT uses Cnmsmsg to issue a CSNDDAT on the task in question. CWATDAT then goes into a wait state. CSNDDAT retrieves the value of the global variable and uses Cnmsmsg to send the data back to the task that issued the CWATDAT. CWATDAT breaks out of the wait state (it has received the data it was waiting for), and parses and displays the data.

The HLL service routines used in this sample are Cnmsmsg, Cnminfc, and Cnmsmsg.

CWATDAT (CNMS4242)

Uses WAIT FOR DATA. It is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.
Its purpose is to find the last time that a command was entered on a given OST. A task global variable is set by CEXIT3 every time a command is entered on an OST. CWATDAT uses Cnmsmsg to issue a CSNDDAT on the task in question. CWATDAT then goes into a wait state. CSNDDAT retrieves the value of the global variable and uses Cnmsmsg to send the data back to the task that issued the CWATDAT. CWATDAT breaks out of the wait state (it has received the data it was waiting for), and parses and displays the data.

The HLL service routines used in this sample are Cnmsmsg, Cnmcmd, and Cnmgetd.

CEXIT2A (CNMS4243)

An exit that converts a write-to-operator (WTO) to an MLWTO by adding two lines to the single-line WTOs that are driving the exit. The HLL service routines used in this sample are Cnmgetd and Cnmaltd.

CCNMI (CNMS4244)

This sample uses Cnmcnmi to send RUs to a PU requesting that the product set ID be returned. Any data returned is sent as a message to the operator. The prolog of the sample contains instructions for setup.

NetView provides the Cnmcnmi service routine for communicating with devices in the network through the CNMI. You can access any data that is returned using the Cnmgetd service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

The HLL service routines used in this sample are Cnmcnmi, Cnmgetd, and Cnmsmsg.

CKEYIO (CNMS4245)

Shows how to code a NetView HLL command processor that allows input/output to a VSAM file through the Cnmkio routine. The command processor must execute on a DST. To run this command, use the EXCMD command or the Cnmsmsg service routine (with a type of COMMAND). The prolog of the sample explains how to set up a DST.

The HLL service routines used in this sample are Cnmkio and Cnmsmsg.

CSCOPCK (CNMS4246)

Shows the command authorization checking capabilities provided by the NetView program. It authorizes keywords and values using the CSCOPCK command. For the command, set up:

- Operator ID
- Operator classes that can access the command
- Operator profile

Refer to the prolog of the sample for more information. This command yields a message if the operator is not authorized to use the keyword and value specified when invoking the command.

The HLL service routines used in this sample are Cnmscop and Cnmsmsg.
C Samples

**CFLVIEW (CNMS4247)**
Illustrates the use of the full-screen VIEW command processor.

The HLL service routines used in this sample are `Cnmcmd` and `Cnmvars`.

**CACTLU (CNMS4248)**
Shows how to issue a VTAM command to activate an LU, trap the VTAM messages that result, and respond depending on the messages received.

The HLL service routines used in this sample are `Cnmcmd`, `Cnmgetd`, and `Cnmsmsg`.

**CSEQLOG (CNMS4249)**
Uses `Cnmsmsg` to log text to a sequential log. The prolog of the sample contains instructions for setup.

The HLL service routines used in this sample are `Cnminf` and `Cnmsmsg`.

**CXITDI (CNMS4250)**
Illustrates the DST initialization exit that is used by the USERVSAM DST. The HLL service routines used in this sample are `Cnmvars` and `Cnmsmsg`.

**CXITVN (CNMS4251)**
Prepares a VSAM empty data set for the USERVSAM DST.

**CSNDDST (CNMS4252)**
Sends a PUT or GET request to the sample HLL data services command processor CDOVSAM to store and retrieve a given value for a specified key (key and value limited to 11 characters in length). The sample also allows a specified NetView command list language variable (defined by the caller) to be set to the retrieved value.

The HLL service routines used in this sample are `Cnmsmsg`, `Cnmvars`, `Cnmgetd`, and `Cnmcmd`.

**CDOVSAM (CNMS4253)**
Is an HLL data services command processor that runs under the sample data services task (task ID USERVSAM). The command processor processes PUT or GET requests sent by the CSNDDST sample, and writes or reads an 11-character value associated with an 11-character key to the sample DST’s VSAM data set. The prolog of CDOVSAM contains instructions on installing the sample USERVSAM data services task.

The HLL service routines used in this sample are `Cnmsmsg` and `Cnmkio`.

**CPRIME (CNMS4254)**
Prepares a VSAM empty data set for CKEYIO.

**CHSNDMU (CNMS4256)**
Sends a software alert to an application over the high performance transport.

The HLL service routines used are `Cnmhsmu` and `Cnmsmsg`. 
CRODMCON (CNMS4260)
Invokes CNMQAPI which enables the user to access a RODM under the control of NetView. The coded example shows a RODM CONNECT function.

The HLL service routines used in this sample are Cnmmsg and Cnmqapi.

CAUTOTB (CNMS4261)
Sends an MSU directly to the automation table for evaluation. It provides a method of testing automation table statements without sending an alert through the hardware monitor.

The HLL service routines used in this sample are Cnmauto, Cnmmsg, and Cnmscan.

CREGISTR (CNMS4262)
Registers or deregisters an application as an MS application or an operations management served application.

You must provide the following:
• Application name
• Command name
• Logmode
• Replace option
• Registration type

Refer to the prolog of the sample for more information. This command sends a message to the invoking operator that reports the return code from the registration or deregistration.

The HLL service routines used in this sample are Cnmrgs and Cnmmsg.

CSENDMU (CNMS4263)
Sends a software alert to ALERT_NETOP from an MS application.

The HLL service routines used in this sample are Cnmmsmu and Cnmmsg.

CHREGSTR (CNMS4266)
Registers an application with the high performance transport.

You must provide the following:
• Application name
• Command name
• Replace option
• Registration type
• Focal point category
• Focal point option
• Notify option

Refer to the prologue of the sample for more information. This command sends a message to the invoking operator that reports the return code from the registration or deregistration.

The HLL service routines used in this sample are Cnmhrgs and Cnmmsg.
C Samples

**CPRSMDB (CNMS4269)**
This HLL sample sends an MDB to NetView for processing. The MDB is an
architected structure for message data. The MDB structure is mapped by the MVS
control block IEAVM105. The HLL service routines used in this sample are
Cnmpmdb and Cnmsmsg.

**CACTPIP (CNMS4405)**
Shows how to use the PIPE command to issue a VTAM command which activates
an LU, traps the resulting VTAM messages, and responds depending on the
messages received.

The HLL service routines used in this sample are Cnmvlc, Cnmvars, Cnmcmd, and
Cnmsmsg.
Index

Special Characters
(\0) end-of-string character 125
& (address) operator 122
\&HDRMTYPE variable 243
\&LINETYPE variable 243
\&MSGID variable 243
\&MSGORIGIN variable 243
\{| type specifier 279
\&PARMSTR variable 65
\&PAUSE function 80, 158
#pragma C variable 40

A
AAUTSKLP subtask 22
abbreviate command 17
access
command list variable
C example 151
PL/I example 76
data 4
data set 89, 171
servers across hosts 8
storage 88, 169
VSAM file 93
accessibility information xiv
ACTIONDL, CNMGETA attribute 230
ACTIONMG, CNMGETA attribute 230
AD/Cycle C/370 programs
achieving optimum performance 120
TRAP(OFF) option 120
AD/Cycle PL/I
preinitialized environment
restrictions 59
run-time options automatically set 50
run-time options
TRAP(OFF) 57
TRAP(ON) 57
adbuf operand 219
ADDR function
description 53
IEL05481 message 53
passing pointer variables 52
address of the HLB control block 119
addressing mode 43
adfunc operand 220
adindex operand 220
adorig operand 184, 420
adqueue operand 220
alert
high performance transport
C example 195
PL/I example 110
MS transport
C example 193
PL/I example 109
alias, C sample 345
ALLOCATE command 7, 56, 129
allocating files, PL/I 56
altering data
C example 167
PL/I example 86
API (application program interface) 4
application registration with high performance transport 106, 193
with MS transport 103, 192, 273
arg parameter in C 119
argy parameter in C 119
asynchronous command execution 3
panel update 4
atdata operand 221
ATTNID, CNMGETA attribute 230
AUTCONID 254
authorization checking command 82, 163
command, keyword, value 280
logon 18
authorized receiver 283
automating MSUs
C example 190
PL/I example 101
automating NetView messages 4
AUTOTOKE, CNMGETA attribute 230
B
BNJDSERV subtask 22
BNJNETOP data set 22
BNJPALLEX installation exit 13, 14
BNJPNL1 data set 264
BNJPNL2 data set 264
books
feedback xii
online xii
ordering xii
BSAM 21
building request units, CNMI 136
C
C control blocks, description 128
C examples
accessing command list variables 151
altering data 167
automating MSUs 190
CNMI 173
coding DSTD installation exits 181
coding installation exits 183
coding WAIT FOR DATA 185
coding command authorization checking 163
data set access 171
Hlbrc C return code field 12
invoking synchronous commands 140
keyed file access 178
locks 155
C examples (continued)
message processing 161
operator input 158
registering applications high performance transport 193
MS transport 192
operations management 192
registering applications, MS transport 192
registering applications, operations management 192
retrieving information 149
sending alerts, MS transport 193
sending commands 141
sending data, high performance transport 195
sending MDB to NetView 197
sending messages 139
storage access 169
translating code points 191
VIEW command processor 159
VSAM access 178
waiting, trapping 143
C I/O considerations 129
C include files
description 128
DSIC 128, 327
DSICCALL 129, 335
DSICCNM 129, 341
DSICCONS 129, 332
DSICORIG 129, 333
DSICPRM 129, 334
DSICVARC 129, 331
C language
command processor support AD/Cycle C/370 119
IBM C/370 V2R1M0 119
installation exit support AD/Cycle C/370 119
IBM C/370 V2R1M0 119
C module, coding
global interfaces 119
restrictions 131
C program
installation entries, program considerations 12
global interfaces 119
production environment 130
restrictions 119
test environment 130
C programs, coding initial parameters 119
C run-time considerations 130
C run-time options 119
C sample template 133
C samples
CACTLU (CNMS4248) 348
CACPPIP (CNMS4045) 146, 350
CAUTOTB (CNMS4261) 349
CCNMI (CNMS4244) 347
<table>
<thead>
<tr>
<th>C samples (continued)</th>
<th>CNM (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDOVSAM (CNMS4253) 14, 348</td>
<td>CNM request 22</td>
</tr>
<tr>
<td>CEXIT2A (CNMS4243) 13, 347</td>
<td>CNM987I message 206</td>
</tr>
<tr>
<td>CEXIT3 (CNMS4240) 13, 346</td>
<td>CNM988I message 206</td>
</tr>
<tr>
<td>CFLVIEW (CNMS4247) 348</td>
<td>CNMALTD (CNMALTDATA) service routine</td>
</tr>
<tr>
<td>CHREGSTR (CNMS4266) 349</td>
<td>description 219</td>
</tr>
<tr>
<td>CHSNDMU (CNMS4256) 348</td>
<td>CNMAUTO (CNMAUTOTAB) service routine 221</td>
</tr>
<tr>
<td>CKEYIO (CNMS4245) 14, 347</td>
<td>CNMC2T (CNMCODE2TXT) service routine 228</td>
</tr>
<tr>
<td>CPMR (CNMS4254) 14, 348</td>
<td>CNMCELL (CNMSTRECELL) service routine</td>
</tr>
<tr>
<td>CPRSMB (CMS4269) 350</td>
<td>description 222, 269</td>
</tr>
<tr>
<td>CREGSTR (CNMS4262) 349</td>
<td>syntax 222</td>
</tr>
<tr>
<td>CSDL (CNMS4260) 349</td>
<td>CNMCMDS (CNMCMDMAND) service routine</td>
</tr>
<tr>
<td>CSCOPCK (CNMS4246) 347</td>
<td>issuing operator commands 212</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMCMDS (CNMCMDMAND) service routine</td>
</tr>
<tr>
<td>CSEQLOG (CNMS4267) 348</td>
<td>description 223</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>excluding, installation exits 11</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 224</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMCNMLI (CNMNLIMI) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 224</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>example 90, 173</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>excluding, installation exits 11</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 224</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMCPYS (CNMCPYS) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 226</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>example 122</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMCSSIR 13</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMGEMET A (CNMGEMET ATR) service routine 229</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMGEMDATA (CNMGEMDATA) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 158, 240</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>example 173</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 240</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMHREGS (CNMHREGS) service routine 244</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMHSMU (CNMHSMU) service routine 247</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMI building request units 136</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>C example 173</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>interface required by DST 28</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>PL/I example 90</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMINFC (CNMINFC) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 75, 253</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 253</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMINFI (CNMINFI) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 75, 256</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 256</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMIQ access queue 64, 139, 243</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMKIO (CNMKIO) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 259</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>excluding, installation exits 11</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 259</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMLK (CNMLK) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 57, 261</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 261</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMMLMEMC (CNMMLMEMC) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 89, 172, 263</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 263</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMMEMR (CNMMEMR) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 89, 172, 263</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 265</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMNAMS (CNMNAMS) service routine 266</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>Cnmv C convert string 125, 127, 137</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMPDB (CNMPDB) service routine 268</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMPNLI data set 264</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMPOOL (CNMPOOL) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 269</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>example 124</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMQAPI (CNMQAPI) service routine 271</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMRCS (CNMRCS) service routine 273</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMSCAN (CNMSCAN) service routine 277</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMSCOP (CNMSCOP) service routine 280</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMSMSG (CNMSMSG) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 17, 26, 282</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>excluding, installation exits 11</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>syntax 282</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMMSU (CNMMSU) service routine 285</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNMSSM (CNMSSM) service routine</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>description 293</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>Cnv C convert string 125, 126, 136</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>cowntime operand 225</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>CNVTOHEX constant 127, 128</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>code</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>C program interfaces 119</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>restrictions 119</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>PL/I program environment 49</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>interfaces 49</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>restrictions 49</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>code points, translating</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>C example 191</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>PL/I example 102</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>code template 61, 133</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>coding DST installation exits</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>C example 181</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>concatenating 13</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>PL/I example 95</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>coding installation exits</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>C example 183</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>PL/I example 96</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>coding WAIT FOR DATA</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>C example 185</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>PL/I example 97</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>command authorization 82, 163, 280</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>command cancellation 215</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>command line, Cmdbuf 119</td>
</tr>
<tr>
<td>CSENDMU (CNMS4263) 349</td>
<td>command list variable, accessing 5, 151</td>
</tr>
</tbody>
</table>
command processor
environment 258
non-preinitialized environments, steps for implementing 34
PL/I support 49
preinitialized environments, steps for implementing 34
VIEW 159
commands:
abbreviation 17
ALLOCATE 7
authorization 280
CNMSCOP service routine 280
FREE 7
PIPE 214
sending with C 141
sending with PL/I 70
synchronous execution 3, 70, 140
syntax 17
verifying authorization 16
commands, HLL
GLOBALY 212
GO 212
overview 212
QUEUE 212, 214
RESET 212, 215
TRAP 212
WAIT 212
common global (CGLOBAL)
command list variable 5
pool 294, 296
variables 77, 152, 155
common global pool 295
communicating with devices 90, 173
communications network management interface (CNMI) 224
completion code 58, 131
concatenate DST installation exits 13
control blocks
C 128, 327
PL/I 55, 301
control line message 283
controlling locks 261
converting
DSICCONS 127, 128
hexadecimal 127, 128
initial parameters, sscanf 119
copying storage 7
correlarea operand 248
CPRIME (CNMS4254) C sample 14, 348
CPRSMSDB (CMS4269) C sample 350
CPU time (TOD Clock) 255
CREGISTR (CNMS4250) C sample 349
critical preinitialized programs 40
CROMDCON (CMS4260) C sample 349
cross-domain
alert 22
command 17
traffic, monitor 17
CSCOPCK (CNMS4246) C sample 347
CSENDMU (CNMS4263) C sample 349
CSEQLQG (CNMS4249) C sample 348
csfrom operand 226
cslen operand 227
CSNDAT (CNMS4241) C sample 13, 346
CSNDST (CNMS4252) C sample 348
csto operand 227
cstype operand 227
CTMPPLT (CNMS4201) C sample 346
CURCONID 254
current date function 254
CURSYS 254
Customer Support xiii
customization, check 18
cvdata operand 294
cvdaten operand 294
cvfunc operand 294
cvname operand 295
cvpool operand 295
CWATDAT (CNMS4242) C sample 13, 346
CXMLDI (CNMS4250) C sample 14, 348
CXMLTVN (CNMS4251) C sample 14, 348
D
d type specifier 279
data, requesting
C 186
PL/I 98
data, sending
C 188
PL/I 100
data access 4
data alteration 86, 167
data definition (DD) statement 56, 129
data queue 5
data queue management 63, 138
data set access
C example 171
closing 263
opening 263
PL/I example 89
data sets, access from HLL 7
DATAQ access queue 64, 139, 242
datatype operand 249
DATE 254
DATETIME2 254
datatype2 254
deallocate files
deallocating files 56
dynamically 7
PL/I 56
debug support
remote interactive debugger (RID) 8, 203
traces 8
DELAY, avoiding 58
deleting a message 12
deliver request unit (RU) 22
DESC, CNMGETA attribute 231
destappl operand 248
destlu operand 248
destnet operand 249
device communications 90, 173
disability information xiv
DISCORIG, mapping Origblk 119
DISPLAY, avoiding 58
domain ID 243, 254
DSCP 28
DSIC C include file 125, 128, 327
DSICCALL C include file 129, 335
DSICCONS C include file 129, 341
DSICCNS C include file 124, 129, 328
DSICES macro 224
DSICHLB C include file 332
DSICLDM data set 264
DSICMD data set member 281
DSICORIG C include file 129, 333
DSICPRM C include file 129, 334
DSICRTSUB task 22
DSICVARC C include file 125, 129, 331
DSIDKS macro 264
DSIE01 installation exit 13, 14
DSIE02A installation exit 13, 14
DSIE03 installation exit 13, 15
DSIE04 installation exit 13, 15
DSIE05 installation exit 13, 16
DSIE06 installation exit 13, 16
DSIE07 installation exit 13, 17
DSIE09 installation exit 13, 17
DSIE10 installation exit 13, 17
DSIE11 installation exit 13, 17
DSIE12 installation exit 13, 18
DSIE13 installation exit 13, 18
DSIE14 installation exit 13, 19
DSIE16 installation exit 13, 20
DSIE16B installation exit 13, 20
DSIE17 installation exit 13, 20
DSIE18 installation exit 13, 20
DSIE19 installation exit 13, 20
DSIE20 installation exit 13, 21
DSIE21 installation exit 13, 21
DSICGDS NPS task 11
DSIGET macro 214
DSILCS macro 224
DSILG task 25, 26
DSILLOGDS mapping function 26
DSIMQS macro 18, 214
DSIMSC data set 264
DSIOOPEN data set 264
DSIPARM data set 264
DSIPCNM PL/I include file 56, 315
DSIPCNS PL/I include file 55, 302
DSIPHLLB PL/I include file 55, 305
DSIPHLLS PL/I include file 56, 307
DSIPLI PL/I include file 55, 301
DSIPORIG PL/I include file 56, 306
DSIPFRM PL/I include file 56, 319
DSIPRF data set 264
DSIPSS macro 15
DSIPUSH macro 267
DSITIB control block 258
DSITRACE task 25
DSITVCTRL control block 257, 258
divarch, varying length character string 122, 125, 136
DSIVTAM data set 264
DSIWCS macro 17
DSIZCMS macro 210
DSRBO keyword 29
DST (data services task)
coding
C 181
PL/I 95
described 27
execute commands in C 141
DST (data services task) (continued)
  execute commands in PL/I 70
  execute HLL services 58
installation exit routine
  XITBN 14, 21
  XITBO 14, 21
  XITCI 14, 22
  XITCO 14, 24
  XITDI 14, 24
  XITVI 14, 25
  XITVN 14, 25
  XITVO 14, 26
  XITXL 14, 26
installation of 27, 226
  schedule commands under 30
DST initialization routine 11
  DSTINIT 27, 29
  dynamic file allocation, deallocation 7

E
  e-mail contact xiii
  edit
    log information 15
    system console messages 17
ELBLEN C external logging offset 26
ELBLOG external logging offset 26
ELBLEN C external logging offset 26
ELBUF external logging offset 26
empty dataset 21, 25
  end of multiline message 283
  end of string character (\0) 125
environment, PL/I 49
errors, PL/I run-time 57
examples
  C
    accessing command list variables 151
    altering data 167
    automating MSUs 190
    CNMI 173
    coding DST initialization exits 181
    coding installation exits 183
    coding WAIT FOR DATA 185
    command authorization checking 163
    data queue management 138
    data set access 171
    D5IEX02A installation exit 14
    invoking synchronous commands 140
    keyed file access 178
    message processing 161
    operator input 158
    retrieving information 149
    sending an MDB to NetView 197
    sending commands 141
    sending messages 139
    storage access 169
    translating code points 191
    using locks 155
    VIEW command processor 159
    VSAM access 178
    waiting, trapping 143
  PL/I
    altering data 86
    CNMI 90
examples (continued)
  PL/I (continued)
    code points, translating 102
    coding 95, 96, 97
    coding, template 61
    command authorization checking 82
    command list variable access 76
    data queue management 63
    dataset access 89
    invoking synchronous commands 70
    MDB, sending to NetView 112
    message processing 82
    MSUs, automating 101
    operator input 80
    parsing character strings 65, 66, 67
    registering applications 103, 106
    retrieving information 75
    sending alerts 109, 110
    sending commands 70
    sending messages 64
    storage access 88
    using locks 79
    VIEW command processor 80
    VSAM key 62
    preparing character strings 66
    resolving composite return codes 209
    external C variable, HLOOPTS
    HLL_NO_CANCEL run-time option 39
    HLL_QUEUE_INPUT run-time option 38
    syntax 38
    external log (EXTLOG operand) 26, 283
    external logging exit (XITXL) 14, 26
    external trace log 15
feedback about publications xiii
  file allocating or deallocating 7
  file I/O capability 56, 129
  FILE option 57
  first failure data capture trace 8
  fixed-length character strings 53, 124
  fixed-length variable, PL/I example 54
  focal point transfer RU header 23
  fopen 129
  format specifications 127
  forward RU 24
  FREE command 7, 56, 129
  full-screen
    capability 159
    input/output 4
    FUNCT, DSTINIT keyword 28
  gdindex operand 242
  gdorign operand 54, 124, 242
  gdbuf operand 240
  gdqueue operand 242
  GET statement 57
  getchar routine 130, 131
  GETDATA function 63, 138
  global installation exit 11
  global variables 212
  GLOBALV command 212
  GO command 4, 213
H
  HAPIENTR debug option 204, 207
  HAPIEXIT debug option 204, 207
  hardcopy log 15, 254
  HCOPY 254
  HDRMTYPE field 223
  hexadecimal conversation 127
  high-level language services, descriptions 3
  high performance transport
    API 244, 247
    CNGMGETD service routine 243
    CNMHRGS service routine 246
    CNMHSMU service routine 247, 251
    data queue management 64, 139
    description 6
    NetView bridge remote access 8
    PHREGSTR sample 326
    PHADNDMU sample 325
    registering applications
      C example 193
      PL/I example 106
    sending alerts
      C example 195
      PL/I example 110
    sending data
      C example 195
      PL/I example 110
    sending data
      C example 195
      PL/I example 110
    HLB control block 58, 131
    HLBLEN C storage field 229, 253, 260
    HLBLEN C storage field 229, 253, 260
    HLPTR
      C pointer field 122
      description 119
      initial parameter in C 119
      restrictions for Cnmcpsys invocation 122
    HLBPTIR
      initial parameter of PL/I program 49
      PL/I pointer field 52
      restrictions with pointer variables 52
    HLPBT
      return code field in C 12
    HLBRC
      resolving composite return codes 209
      return code field in PL/I 12, 58
    HLL command processors
      coding in C 119
      considerations for 58, 131
    HLL commands
      GLOBALV 212
      GO 212, 213
      overview 212
      QUEUE 212, 214
HLL commands (continued)
RESET 212, 215
TRAP 212, 218
WAIT 212, 218
HLL definition facilities
HLLENV command
DEFAULT keyword 38
preinitialized environments, defining 37
REGENVS and CRITENVS keywords 37
storage keywords 37
HLL installation exit routines
invocation restriction 3
overview 11
HLL_NO_CANCEL run-time option
description 39
HLL PL/I restrictions 58
HLL program
accepting queued input 38
terminate on cancel/reset 39
HLL_QUEUED_INPUT run-time option 38
HLL run-time options
default values 38
specifying, HLLOPTS 38
HLL service routines
CNMALTD (CNMALTDATA) 219
CNMAUTO (CNMAUTOTAB) 221
CNMCT2T (CNMCODE2TXT) 228
CNMCEL (CNMSTRCELL) 222
CNMCMID(CNMCOMMAND) 223
CNMCMNI (CNMI) 224
CNMCPY(S (CNMCPYESTR) 226
CNMGETA (CNMGGETATTR) 229
CNMGGETD (CNMGGETDATA) 240
CNMRHGCS (CNMRHREGIST) 244
CNMHSMU (CNMHSENDMNU) 247
CNMNF (CNMNINFO) 253
CNMINFI (CNMINFO) 256
CNMKIO (CNMKKEYIO) 259
CNMLK (CNMLOCK) 261
CNMMEMC (CNMCMEM) 263
CNMMEMO (CNMOPENMEM) 263
CNMMEMR (CNMMREADMM) 265
CNMMMDB (CNMMMDBPR) 268
CNMNAMS (CNMNAMSTR) 266
CNMPOOL (CNMSTRPOOL) 269
CNMQAPI (CNMQREPRE) 271
CNMBOS (CNMBREGIST) 273
CNMSCAN (CNMSCANN) 277
CNMSCOP (CNMSCOPECK) 280
CNMSMGS (CNMSENDMSG) 282
CNMSMU (CNMSENDMNU) 285
CNMVARS (CNMVARPOOL) 293
description 219
input parameters
fixed-length character strings 53
integer variables 53
pointer variables 52
varying-length character strings 54
HLLENV command
REGENVS and CRITENVS keywords
critical programs 38
programs that should run 37
HLLOPTS external C variable
HLL_NO_CANCEL run-time option 39
HLL_QUEUED_INPUT run-time option 38
syntax 38
hrapplname operand 244
hrcmdname operand 245
hrlogmod 245
hrreplvalue operand 246
hrtype operand 246
I/O considerations, PL/I
sharing files, PL/I and C 57
writing files, multiple programs 57
icdata operand 253
icdatlen operand 253
icname operand 254
IDATAQ access queue
description 64, 243
usage 119, 220
IEL05481 message 53
IFRAUGMT, CNMGETA attribute 231
IFRAUI3X, CNMGETA attribute 232
IFRAUIN3, CNMGETA attribute 231
IFRAUIND, CNMGETA attribute 231
IFRAUSB2, CNMGETA attribute 232
IFRAUSC2, CNMGETA attribute 232
IFRAUSRB, CNMGETA attribute 232
IFRAUSRC, CNMGETA attribute 232
IFRAUTA1, CNMGETA attribute 232
IFRAUWF1, CNMGETA attribute 233
iiname operand 257
iinumb operand 259
immediate command 58, 131, 223
include files - C 128, 327
include files - PL/I 55, 301
information, accessibility xiv
information, disability xiv
information retrieval
C 75
PL/I 149
initial command 15
initial data queue (IDATAQ) 119
initial operands, PL/I 49
input, line mode 4
input before command processing 13
input from the operator 14
input from the system console 17
input/output considerations
full-screen 4
preinitialized environments 57
installation exits (continued)
descriptions (continued)
BSAM output 14, 21
CNM input 14, 22
CNM interface output 14, 24
cross-domain command send 13, 17
DST initialization 14, 24
cryptographic key for DSTICPRF 21
external logging 14, 26
HDRTYPEX cross-domain return command receive 13
input, command processing 15
input, operator 14
input, system console 13, 17
input before command processing 13
log output 13, 15
logon validation 13, 18
message output, cross-domain 13
message output, this domain 13
OST/NNT message receiver 13, 18
output, operator 14
output, system console 13, 17
RUNCMD authorization 13, 20
SAW filtering 21
solicited VTAM messages 13, 16
unsolicited VTAM messages 13, 17
VSAM empty file 14, 25
VSAM input 14, 25
VSAM output 14, 26
DST
C 181
PL/I 95
for HLL 11
general return codes 12
names 13
BNJPALEX 13, 14
DSEIX01 13, 14
DSEIX02A 13, 14
DSEIX03 13, 15
DSEIX04 13, 15
DSEIX05 13, 16
DSEIX06 13, 16
DSEIX07 13, 17
DSEIX09 13, 17
DSEIX10 13, 17
DSEIX11 13, 17
DSEIX12 13, 18
DSEIX13 13, 18
DSEIX14 13, 19
DSEIX16 13, 20
DSEIX16B 13, 20
DSEIX17 13, 20
DSEIX18 13, 20
DSEIX19 13, 20
DSEIX20 13, 21
DSEIX21 14, 21
XITBN 14, 21
XITBO 14, 21
XITCI 14, 22
XITCO 14, 24
XITDI 14, 24
XITVI 14, 25
XITVN 14, 25
Index 355
installation exits (continued) names (continued) XITVO 14, 26 XITXL 14, 26 non-preinitialized environments, steps for implementing 34 overview 11 performance degradation 12 PL/I coding 96 PL/I support 49 preinitialized environments, steps for implementing 34 replacing messages 12 integer variable, PL/I example 53 integer variables 53, 123 interfaces C program 119 PL/I program 49 internal function request (IFR) 18 interruption request block (IRB) exit 12 invoking synchronous commands C example 140 PL/I example 70 IRB (interruption request block) exit 12 ISASIZ PL/I run-time option displayed by RID 206 istring operand 127, 128 J JOBNAME, CNMGETA attribute 233 JOBNUM, CNMGETA attribute 233 K KEY, CNMGETA attribute 233 key-sequenced data set (KSDS) 25 key-sequenced VSAM files 259 keyboard, shortcut keys xiv keyed file access C example 178 PL/I example 93 under a DST 259 keyword authorization checking 280 L label line message 283 LE/390 libraries 50 LEOPTS static external variable 51, 121 line mode input 4 link-edit JCL 43 PL/I program 43 lkfunc operand 262 lkname operand 262 lkoption operand 262 lkscope operand 262 local pool 295 lock management C example 155 controlling 261 PL/I example 79 user-defined locks 7 log output 15 MS transport registering applications C example 192 PL/I example 103 sending alerts C example 193 PL/I example 109 MS transport layer 6 MSGAUTH, CNMGETA attribute 234 MSGCAATTR, CNMGETA attribute 234 MSGCBGP, CNMGETA attribute 236 MSGCMISC, CNMGETA attribute 234 MSGCMVL, CNMGETA attribute 234 MSGCMSGT, CNMGETA attribute 235 MSGCOJB, CNMGETA attribute 235 MSGCPROD, CNMGETA attribute 235 MSGCSPLX, CNMGETA attribute 235 MSGCSYID, CNMGETA attribute 235 MSGDOMFL, CNMGETA attribute 235 MSGGDATE, CNMGETA attribute 235 MSGGFGPA, CNMGETA attribute 235 MSGGMID, CNMGETA attribute 236 MSGGMEM, CNMGETA attribute 237 MSGGSYID, CNMGETA attribute 237 MSGGTIME, CNMGETA attribute 237 MSGGWMID, CNMGETA attribute 237 MSGSCOB, CNMGETA attribute 238 MSGSTOKEN, CNMGETA attribute 238 MSGSTTYP, CNMGETA attribute 238 MSU automating C example 190 PL/I example 101 multiline messages 4, 161, 240 MVS console 18 MVS console operator task 18 MVS JOB name/number 233 MVS system 239 MVS system log 15 MVSLEVEL 254 MVTUFLD control block 258
PL/I include files

description 55
DSIPCNM 56, 315
DSIPCONS 55, 302
DSIPHBL 55, 305
DSIPHLLS 56, 307
DSIPLI 55, 301
DSIPORIG 56, 306
DSIPPM 56, 319

PL/I INCLUDE files, optional

DSIPCNM 56
DSIPCONS 55

PL/I INCLUDE files, required

DSIPHLLS 56
DSIPLI 55
DSIPORIG 56
DSIPPRM 56

PL/I installation exits, support for 49

PL/I program

coding template 61
commands to avoid
DELAY 58
DISPLAY 58
ON FIXEDOVERFLOW 58
 ON OVERFLOW 58
 ON UNDERFLOW 58
 ON ZERODIVIDE 58
PLIRECT 58
PLIRETV 58
WAIT 58

compiling 43
ending normally 58
environment 49
file I/O capability 56
FILE option, failure to code 57
I/O considerations
 sharing files, PL/I and C 57
 writing files, multiple programs 57
initial parameters
CMDBUF 49
HLBPTR 49
ORIGBLCK 49
installation exits, performance considerations 12
interfaces 49
link-editing 43
requesting data 98
restrictions 49
run-time errors
 NetView 57
 operating system 57
 running 45
 sending data 100
translating code points 102

PL/I run-time considerations 57

PL/I samples (continued)

PL/I run-time considerations 57

PL/I samples
PACTLU (CNMS4218) 324
PACTPPI (CNMS4305) 73, 326
PAUOTOB (CNMS4231) 325
PCNMI(CNMS4214) 323
PDOVSAM (CNMS4223) 14, 325
PEXIT2A (CNMS4213) 13, 323
PEXIT3 (CNMS4210) 13, 322
PFLVIEW (CNMS4217) 324
PHREGSTR (CNMS4236) 326
PHSNDMU (CNMS4226) 325
PL/I samples (continued)
PKEYIO (CNMS4215) 323
PPRIME (CNMS4224) 14, 325
PR3SMDBN (CNMS4239) 326
PREGSTR (CNMS4232) 325
PRODMCON (CNMS4230) 325
PSCOCPK (CNMS4216) 324
PSENDMU (CNMS4233) 326
PSEQLOG (CNMS4219) 324
PSNDDAT (CNMS4211) 13, 322
PSNDDST (CNMS4222) 325
PTMPPILT (CNMS4200) 322
PWATDAT (CNMS4212) 13, 323
PXITDI (CNMS4220) 14, 324
PXITVN (CNMS4221) 14, 324
table of 321
PL/I run-time considerations 57

PL/I run-time considerations

PL/I installation exits, support for 49

PL/I program

coding template 61
commands to avoid
DELAY 58
DISPLAY 58
ON FIXEDOVERFLOW 58
 ON OVERFLOW 58
 ON UNDERFLOW 58
 ON ZERODIVIDE 58
PLIRECT 58
PLIRETV 58
WAIT 58

compiling 43
ending normally 58
environment 49
file I/O capability 56
FILE option, failure to code 57
I/O considerations
 sharing files, PL/I and C 57
 writing files, multiple programs 57
initial parameters
CMDBUF 49
HLBPTR 49
ORIGBLCK 49
installation exits, performance considerations 12
interfaces 49
link-editing 43
requesting data 98
restrictions 49
run-time errors
 NetView 57
 operating system 57
 running 45
 sending data 100
translating code points 102

PL/I run-time considerations 57

PL/I samples
PACTLU (CNMS4218) 324
PACTPPI (CNMS4305) 73, 326
PAUOTOB (CNMS4231) 325
PCNMI(CNMS4214) 323
PDOVSAM (CNMS4223) 14, 325
PEXIT2A (CNMS4213) 13, 323
PEXIT3 (CNMS4210) 13, 322
PFLVIEW (CNMS4217) 324
PHREGSTR (CNMS4236) 326
PHSNDMU (CNMS4226) 325

preinitialized environments (continued)
 installation exits
 steps for implementing 35
 languages supported 33
 PL/I program 49
 PL/I run-time options
 entry points 50
 specifying 50
 programs that should run 37
 REGENVS and CRITENVS
 keywords 37
 restrictions 58
 storage keywords 37
 storage values 51
primary POI task (PPT) 70, 79, 141
print function 127, 131
process ID 255
PRODMCON (CNMS4230) PL/I sample 325
profile name (function) 18
program, defining cancelable, non-cancelable 39
PTYN, CNMGETA attribute 239
PSCOCPK (CNMS4216) PL/I sample 324
PSENDMU (CNMS4233) PL/I sample 326
PSEQLOG (CNMS4219) PL/I sample 324
PSNDDAT (CNMS4211) PL/I sample 13, 322
PSNDDST (CNMS4222) PL/I sample 325
PTMPPILT (CNMS4200) PL/I sample 322
publications
 feedback xii
 online xii
 ordering xii
PUT statement 57
putcchar routine 130, 131
PWATDAT (CNMS4212) PL/I sample 13, 323
PXITDI (CNMS4220) PL/I sample 14, 324
PXITVN (CNMS4221) PL/I sample 14, 324

Q
QUEUE command 214
queued input, HLL program 38

R
RECFMS 22
RECMS 22
registering applications
 high performance transport
 C example 193
 specifying 244
 PL/I example 106
MS transport
 C example 192
 description 273
 PL/I example 103
registering applications (continued)
operations management
C example 192
PL/I example 103
regular command 15
Remote NetView Bridge 8
remote systems, operating 6
replace messages 12
reply identifier 239
reply RU 225
replicmd operand 250
REPLYID, CNMGETA attribute 239
REPORT option 120
request handling 3
requesting data
C example 186
PL/I example 98
RESET command 215, 258
resetting and terminating HLL program 39
response handling 3
restrictions
C program 131
PL/I program 58
RETCODE exit value 207
retrieving information
C example 149
PL/I example 75
return codes
C 131
composite
major and minor 209
resolving 209
description 12
passed by installation exits 12
passed in C Hlbrc field 12
passed in PL/I HLBRC field 12
PL/I 58
simple 209
USERASIS 12
USERDROP 11, 12
USEREVNT 24
USEREXLG 23
USERHCL 16
USERHCLR 16
USERLOG 16
USERLOGR 16
USERNRL 16
USERSNSL 16
USERSNSLR 16
USERSWAP 12
REXX language 295
tgapppl operand 274
tgcmd operand 274
tgfoct operand 274
tgfpcat operand 275
tgrel operand 275
tgtype operand 276
RH header 225
RID
CONTINUE 203
description 203, 205
END 204
MODNAME 204
OPTION 204
RUN 204
STEP 204
ROUTCODE, CNMGETA attribute 239
ROUTE-INOP request 22
RTM 6
RU formats 22
run-time options
default values 38
specifying, HLOPTS 38
S
s type specifier 279
scmd operand 280
schedule commands under the DST 30
scvalwd operand 280
svalue operand 281
SDDNM keyword 29
sending alerts
high performance transport
C example 195
PL/I example 110
MS transport
C example 193
PL/I example 109
sending an MDB to NetView
C example 197
PL/I example 112
sending commands
C example 141
PL/I example 70
sending data
C example 188
PL/I example 100
sending messages
C example 139
description 282
PL/I example 64
serial number of screen update 258
serialize access 261
service routines, limitations, installation exits 11
SESSID, CNMGETA attribute 239
shortcut keys, keyboard xiv
signal function 130
simulated terminal input 15
single line message 283
smdestid operand 282
smdestyp operand 283
SMF 8
smmsgtyp operand 255, 283
SMSSID, CNMGETA attribute 239
smtext operand 283
solicited VTAM messages 13, 16
source operand 269
SPASS keyword 29
spclass operand 270
spfunc operand 124, 270
SPIE run-time option
C program 119, 130
PL/I V2R3M0 57
STARTIME 255
STCK instruction 254, 256
stdin 130
stdout 130
storage
copy 7
named 7
overlay 230, 254
pool
ALLOC 269
FREE 269
LOCATE 269
restriction 7
storage, managing 7
storage access
C example 169
PL/I example 88
string 279
substitute system symbolics 292
succorr operand 286
sudata operand 286
sudstapl operand 286
sudstlu operand 287
sudnet operand 287
sudtype operand 287
sumutype operand 288
suaappl operand 288
SUPPCHAR 255
surplcmd operand 288
suspcor operand 289
susynch operand 289
synchronous commands
execution 3
invoking, C 140
invoking, PL/I 70
synchronous operand 250
synonym, command 17
SYSCONID, CNMGETA attribute 239
SYSID, CNMGETA attribute 239
SYSPLEX, NetView 57
SYSPLEX
CNMINFC service routine
attribute 255
identifying command list source 255
identifying message source 235
SYSPRINT, NetView 57
system
message flags 234
message type 238
system ABEND 57, 130
system console 283
system symbolic substitution
access 5
T
TAF session ID 239
TARGET side 100
TASK 255
variables, global 212
VARTOVAR constant 122
varying length character string
Cnmmvic 137
Cnmmvc 136
copying null-terminated text into description 124, 136
verify operator authorization 16
VIEW command 80
VIEW command processor
C example 159
description 4
PL/I example 80
virtual storage, managing 7
VSAM
access
C example 178
PL/I example 93
data set 26
database 25
DST installation 28
empty file 14, 25
files 5, 260
I/O 25, 26, 178
input 14, 25
keyed file access 178
open failure 25
output 14, 26
services 29
vsdata operand 259
vsdatlen operand 259
vsfunc operand 260
vskey operand 260
vsoption operand 260
vstring operand 127, 128
VT 255
VTAM 255
command invocation 13, 16
component ID 256
level 255
messages, changing 16
VTCOMPID 256

W
WAIT
avoiding 58
data 57, 97, 185
waiting and trapping
C example 143
C example using PIPE 146
PL/I example 74
PL/I example using PIPE 73

X
x type specifier 279
XITBN installation exit 14, 21
XITBO installation exit 14, 21
XITCI installation exit 14, 22
XITCO installation exit 14, 24
XITDI installation exit 14, 24
XITVI installation exit 14, 25
XITVO keyword 29
XITXV installation exit 14, 25

U
u type specifier 279
unattended operator task 18
unsolicited VTAM messages 13, 17
user-defined lock, managing 7
user-defined services 30
USERASIS return code 12
USERDROP return code 11, 12
USEREVENT return code 24
USEREXLG return code 23
USERHCL return code 16
USERHCLR return code 16
USERLOG return code 16
USERLOGR return code 16
USERNL return code 16
USERNLS return code 16
USERSWAP return code 12

V
value authorization checking 280
variables (command list language)
\&HDRMTYPE 243
\&LINETYPE 243
\&MSGID 243
\&MSGORIGIN 243
access 5

360  Customization: Using PL/I and C