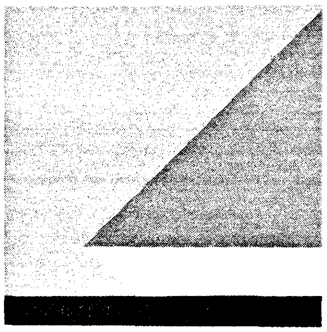
IBM

NetView™

SC31-6037-0

Customization: Using PL/I and C

Release 3



File Number
S370/4300/30XX-50

Program Number
5665-362 (MVS/XA)

First Edition (May 1989)

This edition applies to Release 3 of the NetView™ licensed program, which runs under the following operating systems:

MVS/XA (NetView program number 5665-362)
MVS/ESA in compatibility mode (NetView program number 5665-362)

Contents

Figures

Tables

About This Book

NetView Customization: Using PL/I and C describes how a system programmer can tailor or supplement the NetView™ program to satisfy unique requirements or operating procedures.

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

This book primarily contains general-use programming interfaces, which allow the customer to write programs that use the services of NetView. However, the book also provides the following types of information, which are explicitly identified where they occur:

Installation exits and other *product-sensitive interfaces* are provided to allow the customer installation to perform tasks such as product tailoring, monitoring, modification or diagnosis. They are dependent on the detailed design or implementation of the product. Such interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

The following is a list of common terms used in this book.

Command List
A list of commands and statements designed to perform a specific function for the user. Command lists can be written in REXX or in NetView command list language.

Command Procedure
Either a command processor written in a high-level language (HLL) or a command list. See command processor and command list for further explanation.

Command Processor
A user-written module designed to perform a specific function. Command processors, which can be written in assembler or a high-level language (HLL), are invoked as commands.

High-Level Language (HLL)
A programming language that does not reflect the structure of any particular computer or operating system. For NetView Release 3, the high-level languages are PL/I and C.

NetView Command List Language
An interpretive language unique to NetView that is used to write command lists.

™ NetView is a trademark of International Business Machines Corporation.

REXX
Restructured Extended Executor Language. An interpretive language used to write command lists.

User Exit Routine
A user-written routine that receives control at predefined user exit points. User exit routines can be written in assembler or a high-level language (HLL).

Who Should Use This Book

This book is intended for experienced system programmers who are knowledgeable in PL/I or C. It is assumed that the user is already familiar with the functions that NetView provides.

Secondary users include operators, network planners, designers, and systems analysts, as well as IBM marketing representatives and instructors.

Where To Find More Information

The following list shows all of the publications in the NetView Release 3 library, arranged according to related tasks. For more information on these and other related publications, see "Bibliography" on page 337.

Evaluation and Education	
Network Program Products General Information	GC30-3350
Bibliography and Master Index for NetView, NCP, and VTAM	GC31-6081
Learning about NetView: Operator Training (PC Diskettes)	SK2T-0292
Planning	
Network Program Products Planning	SC30-3351
NetView Storage Estimates (PC Diskettes)	SK2T-1988
Console Automation Using NetView: Planning	SC31-6058
Installation and Administration	
NetView Installation and Administration Guide	SC31-6018
NetView Administration Reference	SC31-6014
Network Program Products Samples	SC30-3352
NetView Tuning Guide	SC31-6079
Customization	
NetView Customization Guide	SC31-6016
NetView Customization: Writing Command Lists	SC31-6015
NetView Customization: Using PL/I and C	SC31-6037
NetView Customization: Using Assembler	SC31-6078
Operation	
NetView Operation Primer	SC31-6020
NetView Operation	SC31-6019
NetView Command Summary	SX75-0026
Diagnosis	
NetView Problem Determination and Diagnosis	LY43-0001
NetView Resource Alerts Reference	SC31-6024
NetView Problem Determination Supplement for Management Services Major Vectors 0001 and 0025	LD21-0023

Part 1. Overview

Chapter 1. NetView High-Level Language Services

Before reading this chapter, you should have read the chapters on designing user-written functions and the NetView™ customization facilities in the *NetView Customization Guide*. You should also have NetView experience as well as programming experience in PL/I or C.

To use this manual most effectively, you should have in mind a specific command processor or user exit routine that you want to write in PL/I or C. For example, you might want to write a command processor to run under a Data Services Task (DST) to store data in a VSAM file or a command processor to run on an Operator Station Task (OST) to display information on an operator's screen. The *NetView Customization Guide* contains information to help you decide the command processors and user exit routines you need to write in order to build your application and the appropriate language to use for each of these routines.

This chapter discusses the NetView services available to you for designing your command processor or user exit routine. The following is the list of services discussed in this chapter:

- Synchronous Command Execution
- Sending Commands (Asynchronous Command Execution)
- Client/Server Request/Response Handling
- Operator Interaction
- Data Access
- Communications Network Management Interface
- NetView Partitioned Data Sets
- NetView Storage Access
- User-Defined Lock Management
- Parsing Character Strings
- Scope Checking
- NetView Message Logging
- Debugging Support

Synchronous Command Execution

High-Level Language (HLL) command processors may invoke any NetView command, including simple commands, command lists, REXX command procedures, assembler command processors, NetView applications such as Session Monitor, and 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.

™ NetView is a trademark of International Business Machines Corporation.

Sending Commands (Asynchronous Command Execution)

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

Client/Server Request/Response Handling

NetView currently supports server tasks that service and reply to requests from one or more operator tasks. The current level of support is accomplished by:

- Allowing the requesting command processor to wait pseudo-synchronously for the reply. This means that the requesting command processor suspends processing while waiting for the reply. The task is not suspended and may continue processing other commands. The suspended command processor resumes processing after receiving the reply.
- Allowing the requests and replies to be sent over NetView-to-NetView cross domain operator sessions.
- Correlating the reply with the correct activation of the requesting HLL command processor. This in turn allows multiple active instances of the requesting command processor under a single operator task.

Operator Interaction

The following describes operator interaction in line mode and full-screen mode.

Line Mode Output

HLL command processors and most user exit routines can send output to the following destinations:

- A NetView operator
- The operating system console
- Another task
- The authorized receiver
- A group of operators defined by the NetView ASSIGN command.

Multi-line messages may be sent as a single unit (MLWTO) so an operator will receive them in a sequence without messages from other sources interspersed. This type of output will appear on the command facility screen or on the operating system console.

Line Mode Input

HLL command processors running under an OST, NNT, or PPT task may 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 may continue to run while waiting for operator input. To avoid forcing the operator to know what language a command procedure is written in, the GO command is used to provide input to a command procedure written in the NetView command list language, REXX, and High-Level Language.

Full-Screen Input/Output

HLL command processors may 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. The capability to ROLL among NetView applications, including HLL command processors, is also available. HLL command processors are treated like command lists when determining ROLL groups. The capability to asynchronously update a panel while it is being displayed is provided.

Data Access

The following is an overview describing data access techniques available to an HLL command processor or user exit routine. These techniques include:

- Message Trapping
- Message Automation
- Command List Variable Access
- NetView Information
- VSAM Files
- Data Queue Manipulation

Message Trapping

Command procedures frequently 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 multi-line messages.

Message Automation

NetView allows HLL command processors to be invoked upon receipt of messages and provides the command processor with access to both the command, and to the message that invoked it. Multi-line messages are supported here as well. NetView also provides services to alter the contents of the messages.

Command List Variable Access

Command procedures frequently store data in task global and common global command list variables which can be accessed by other command procedures. HLL command processors and user exit routines can access and update these variables.

NetView Information

HLL command processors and user exit routines may query certain information (such as domain ID, message attributes, etc.) about the current NetView environment. The information provided by this function is similar to that provided by control variables in the NetView command list language and control block fields in Assembler.

VSAM Files

HLL data services command processors may read, write, update, and delete records in VSAM files associated with the task under which the command processor is running. All requests are pseudo-synchronous allowing other command processors to execute while file I/O requests are executing.

Data Queue Manipulation

This function allows an HLL command processor or user exit routine to manipulate HLL data queues. Each HLL command processor and user exit routine has a set of queues from which it may receive data. There is a queue for each of the following types of input data:

- Input from a NetView operator
- Operator messages trapped for processing
- Data from another HLL command processor or user exit routine
- Initial data associated with a message that caused an HLL command processor or user exit routine to be executed.
- Data solicited over the CNMI.

Communications Network Management Interface

HLL data services command processors can send and receive data over the Communications Network Management Interface (CNMI). The CNMI is used to forward commands to and collect data from devices in the network. For example, RTM data is collected from PU2 control units using the CNMI. Unsolicited data received over the CNMI may be processed by HLL command processors as well. Solicited requests are pseudo-synchronous.

NetView Partitioned Data Sets

HLL command processors and user exit routines have read access to the NetView partitioned data sets. This allows you to write a program that uses the information that is in the NetView partitioned data sets. This function is completely synchronous.

Dynamic File Allocation/Deallocation

NetView provides facilities to dynamically allocate/deallocate files by the use of NetView ALLOCATE/FREE commands. (Refer to the *NetView Operation* manual for more details). Once allocated, these files may be accessed using the file I/O facilities present in the language being used.

NetView Storage Access

An HLL command processor or user exit routine will be able to allocate and free a named storage pool from the NetView subtask under which it is running. A storage pool is composed of a primary storage block and related secondary blocks. Once a pool is allocated, individual storage cells within the pool may be accessed as needed by the HLL command processor or user exit routine. The storage should be referenced only from the task associated with it.

Storage Copying

This function allows HLL command processors and user exit routines to make a copy of any area of virtual storage in the NetView address space in which the HLL command processor or user exit routine is running. If a request is made to copy an area of storage that is not currently addressable, a return code is generated instead of an 0C4 ABEND. This function is useful for debugging. The user must exercise great care to insure that the storage to which the copy is made belongs to their program.

Named Storage

This function allows HLL command processors or user exit routines to obtain an area of virtual storage and associate a name with it, so that other HLL command processors and user exit routines running under the same task may access this area of storage. This function can be used by transaction-oriented applications to save data across transactions.

User-Defined Lock Management

HLL command processors and user exit routines will be able to 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 PL/I language. This service is similar to the SSCANF function available in the C language. It is intended to facilitate the parsing of commands and messages.

Scope Checking

HLL command processors and user exit routines can invoke the NetView scope checking facility 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 user exit routines can send message output to the following logs:

- The network log
- An external log (such as SMF)
- A sequential log

Debugging Support

The NetView HLL API provides two debugging aids for users: an interactive debugger that displays the parameters and results of all HLL API service routine invocations, and a continuous First Failure Data Capture trace for ABEND debugging. In addition, the NetView internal trace may be used (see *NetView Problem Determination and Diagnosis* for further detail).

Remote Interactive Debugger (RID)

The remote interactive debugger (RID) allows NetView HLL service routine calls to be trapped and displayed to the programmer. RID is implemented using NetView commands and messages so that debugging procedures may be created using NetView command list language, REXX, or HLL command procedures. In addition, since NetView provides facilities to route commands and messages to remote systems, RID may be used from one system to debug an HLL command processor or user exit routine running on another system.

RID operates at the subtask level, so using RID to stop an HLL command processor or user exit routine running under one subtask will not effect other subtasks in the same NetView address space.

First Failure Data Capture Trace (FFDCT)

Each HLL command processor or user exit routine maintains an eight-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 will give some indication of what was going on before the ABEND. Refer to the First Failure Data Capture Trace in the *NetView Problem Determination and Diagnosis* manual for further detail.

Chapter 2. HLL User Exit Routines

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

This chapter contains *product-sensitive programming interfaces* provided by NetView. Installation exits and other *product-sensitive interfaces* are provided to allow the customer installation to perform tasks such as product tailoring, monitoring, modification or diagnosis. They are dependent on the detailed design or implementation of the product. Such interfaces should be used only for these specialized purposes. Because of their dependencies on detail design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Overview of User Exit Routines

NetView provides two types of user exits for which you may write routines:

- Global user exits (DSIEXnn), which apply to all NetView tasks. The global user exit routines are loaded when NetView starts. See Table 1 on page 11 for a list of user exits.
- Data Services Task (DST) user exits (XITnn and BNJPALEX), which apply only to DST subtasks. The DST user exit routines are loaded when their DST starts. Each DST can have its own set of user exit routines.

Note: DST user exits should not be used under the Network Product Support (NPS) task named DSIGDS.

You should avoid coding user exits for frequently executed functions, such as VSAM I/O, since performance can be degraded significantly.

Each user exit handles a particular event, such as the reception of data from the system console. When that event occurs, NetView passes control to the appropriate user exit routine for processing. After processing, the user exit routine returns control and passes a return code to NetView. Optionally, up to 10 DST exit routines can be concatenated. If the first exit did not indicate USERDROP, NetView then calls the next one in the sequence. This process continues until the last DST exit has returned control to NetView. For more information on input to the user exit routines see Chapter 4 on page 27 or Chapter 8 on page 91.

Coding Restrictions

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

- CNMCMD
- CNMCNMI
- CNMKIO

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 will only be invoked in the mainline environment if written in HLL. If written in assembler these exit routines can be invoked in both the mainline and IRB exit environments. See *NetView Customization: Using Assembler* for additional information on exits running in the IRB exit environment.

General Return Codes

Unless otherwise noted, user exit routines can pass the following return codes to NetView 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:

Return Code	**Meaning**
USERASIS (0)	Use the message as presented to the user 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 on how to delete messages, see "Deleting Messages."
USERSWAP (8)	Replace the message with the modified CMDBUF (*Cmdbuf*). For more information on how to replace messages, see "Replacing Messages."

Deleting Messages

To delete a message entirely, use return code USERDROP.

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

Replacing Messages

To replace a message, use return code USERSWAP and set the input CMDBUF (*Cmdbuf*) data to the desired data. The replacement data must be less than or equal in length to the original CMDBUF (*Cmdbuf*) data; otherwise it will be truncated to the original length of the CMDBUF (*Cmdbuf*) data.

User exit DSIEX02A provides a more flexible interface for replacing messages using CNMALTD. See "CNMALTD (CNMALTDATA) — Alter Data On A Queue" on page 190.

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

The following table lists all HLL user exits and the task environments they can be called under.

Table 1. User Exit Environments

Exit	Description	Applicable Tasks
BNJPALEX	Not supported in HLL	
DSIEX01	Not supported in HLL	
DSIEX02A	Message Output this Domain or Message Output Cross-Domain	NNT, OST, PPT NNT, OST, CNMCSSIR
DSIEX03	Input Before Command Processing HDRTYPEX Cross-Domain Return Command Receive	NNT, OST, PPT NNT
DSIEX04	Log Output for Buffers not Processed by DSIEX02A	Main Task or Any Subtask
DSIEX05	Before VTAM Command Invocation*	NNT, OST, PPT
DSIEX06	Solicited VTAM Messages*	NNT, OST, PPT
DSIEX07	Cross-Domain Command Send	NNT, OST
DSIEX09	Output to the System Console	Main Task or Any Subtask
DSIEX10	Input from the System Console	Main Task
DSIEX11	Unsolicited VTAM Messages*	PPT
DSIEX12	Logon Validation	NNT, OST
DSIEX13	OST/NNT Message Receiver	NNT, OST, PPT
DSIEX14	Before Logoff	NNT, OST
DSIEX16	Not supported in HLL	
XITBN	BSAM Empty File	DST
XITBO	BSAM Output	DST
XITCI	CNM Interface Input	DST
XITCO	CNM Interface Output	DST
XITDI	DST Initialization	DST
XITVI	VSAM Input	DST
XITVN	VSAM Empty File	DST
XITVO	VSAM Output	DST
XITXL	External Logging	DST

Note:

* When using NetView POI only. Does not include VTAM messages from other sources; for example MVS/XA SSI. You can process these messages in DSIEX02A.

User Exits

NetView provides a number of user exits. This section describes each of these user exits. For a discussion of message flows and interception points in OST, NNT, and PPT tasks see *NetView Customization: Using Assembler.*

DSIEX01: Input from the Operator

This exit is only available through assembler language. See *NetView Customization: Using Assembler*.

DSIEX02A: Output to the Operator

Description: NetView calls DSIEX02A 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 since logging options may be specified in either DSIEX02A or in the message automation table.

Example of Use: Since 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.

If your messages will be translated (such as to Kanji), changes to the message text may affect the translations. (See *NetView Installation and Administration Guide* for more information.)

Coding Considerations: Message automation is invoked after this exit routine has been called; therefore, any changes made for messages in this user exit may affect message automation. Message automation is not invoked for a message that has been deleted by this exit routine.

Do not use the USERSWAP return code to replace messages. Use the CNMALTD service. See "CNMALTD (CNMALTDATA) – Alter Data On A Queue" on page 190.

DSIEX02A is supported only in 31-bit addressing mode.

CNMSMSG can be issued from DSIEX02A but only with the destination type of 'TASK'. The message resulting from the CNMSMSG call will not redrive DSIEX02A.

DSIEX03: Input Before Command Processing

Description: All regular commands call DSIEX03. Regular commands include the following:

- Commands issued by a command procedure
- Commands received from another subtask
- Commands used to start the hardcopy log at logon
- Commands used as the initial command
- Commands entered as simulated terminal input
- Commands resulting from the message automation table
- Commands entered for an MVS console operator task
- Commands entered from a terminal
- Commands received as HDRTYPEX messages from an NNT
- Commands 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 screen are passed to DSIEX01 and DSIEX03. The remaining command types previously listed are passed to DSIEX03.

Example of Use: You can use DSIEX03 to restrict usage of particular, regular commands if your conditions are more complex than those provided by scope checking.

Coding Considerations: None.

DSIEX04: Log Output

Description: NetView 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 since logging options may be specified either in DSIEX02A or in the message automation table.

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

Coding Considerations: DSIEX04 can run under any subtask that initiates message logging. Be sure that any HLL services you request are supported by the subtask under which the routine is running. To determine the subtask you are running under see TASK in "CNMINFC (CNMINFOC) – Query NetView Character Information" on page 209.

Return Code Considerations: DSIEX04 may pass four other return codes in addition to USERASIS, USERDROP, and USERSWAP.

Return Code	**Meaning**
USERLOG	Write the message to the network or MVS system log only.
USERLOGR	Write the substituted message to the network or MVS system log only.
USERHCL	Write the message to the hardcopy log only.
USERHCLR	Write the substituted message to the hardcopy log only.

DSIEX05: Before VTAM Command Invocation

Description: NetView calls DSIEX05 when preparing to pass a command to VTAM through the POI interface; 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, not using the 'MVS' prefix, which are passed through NetView's POI.

Command passed to DSIEX05 have already been processed under DSIEX03 (and perhaps, DSIEX01).

DSIEX06: Solicited VTAM Messages

Description: NetView 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, not using the 'MVS' prefix, which are passed through NetView's POI.

Message automation is invoked after this exit routine has been called. Therefore, any changes made to messages in this user exit may affect message automation. Message automation is not invoked for a message that has been deleted by this exit routine.

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

DSIEX07: Cross-Domain Command Send

Description: NetView 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.

Coding Considerations: None.

DSIEX09: Output to the System Console

Description: NetView calls DSIEX09 when a message is written to the system console operator using macro DSIWCS. See DSIWCS in *NetView Customization: Using Assembler*. The message has not been formatted for transmission.

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

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

DSIEX10: Input from the System Console

Description: NetView 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 could then be expanded in the user exit routine.

Coding Considerations: DSIEX10 can only be called from the main task, not from a subtask.

DSIEX10 is not called for commands entered by an operator using an MVS console operator task (OST). DSIEX03 is called instead.

DSIEX11: Unsolicited VTAM Messages

Description: NetView calls DSIEX11 when an unsolicited VTAM message is received via the POI interface. In addition, when VTAM's PPOLOG=YES modify or start option is used, copies of the messages are presented to DSIEX11. This user 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 NetView.

Coding Considerations: Message automation is invoked after this exit routine has been called; therefore, any changes made for messages in this user exit can affect message automation. Message automation will not be invoked for a message that has been deleted by this exit routine.

DSIEX12: Logon Validation

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

Example of Use: You can use DSIEX12 to perform additional checking of authorization and environmental customization. DSIEX12 can also send messages to other operators.

Coding Considerations: If the user exit routine issues a return code of zero, 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 OST and NNT tasks including unattended operator and MVS console operator tasks.

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

```
OFFSET       LENGTH      FUNCTION
------       ------      --------
  0             8        OPERATOR ID NAME
  8             8        OPERATOR LU NAME
 12             8        PASSWORD
 20             8        HARDCOPY DEVICE NAME
 28             8        PROFILE NAME
 36             8        NEW PASSWORD
```

DSIEX13: OST/NNT Message Receiver

Description: NetView calls DSIEX13 when either a message buffer or a user-defined internal function request (IFRCODUS) is received through macro DSIMQS. DSIEX13 is called within the message receiver for subtask-to-subtask communication. A message buffer is any non-HDRTYPEI (IFR) buffer. See IFRCODUS and DSIMQS in *NetView Customization: Using Assembler*.

Example of Use: You can use DSIEX13 in conjunction with IFRCODUS to initiate a user function with a buffer. Code DSIEX13 to perform the user function specified by IFRCODUS.

Coding Considerations: When DSIEX13 returns, these buffers are written to the operator terminal unless return code USERDROP is issued. The messages are logged after user exit DSIEX02A is called.

DSIEX14: Before Logoff

Description: NetView calls DSIEX14 when an OST or NNT subtask 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 terminate.

The subtask cannot communicate with the operator's terminal at this point. It is possible, however, to write to the system console and to write entries to the log.

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

Coding Considerations: Because there is no buffer associated with logoff processing, DSIEX14 does not receive an input buffer (the length of the command buffer will be zero).

Return Code Considerations: NetView ignores any return code received from this user exit routine.

DSIEX16: Post-Message Automation Table Exit

Available through an assembler interface only. See *NetView Customization: Using Assembler*.

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. You should code this user exit only if you wish to write your own BSAM subtask using DST as a base.

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

Return Code Considerations: 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 data base.

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

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

You should avoid coding user exits for frequently executed functions, such as BSAM I/O, since performance can be degraded significantly.

XITCI: CNM Interface Input

Description: The DST calls XITCI after CNM data is received.

Example of Use: You can use XITCI to modify CNM interface input data (Deliver RU).

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

If a substitute buffer is returned, the data must be a valid SNA request unit (RU). See *Systems Network Architecture Product Formats* for a discussion of RU formats.

XITCI invoked under the DSICRTR subtask allows access to unsolicited CNM data prior to NetView routing (except for cross domain alerts, which are only accessible under the BNJDSERV subtask). XITCI invoked under a DST other than DSICRTR will allow access to unsolicited CNM data particular to the invoking DST.

Network Services Request Units are routed as follows:

Request	Header Value	Responsible Subtask Name
RECMS	X'010381'	BNJDSERV
RECFMS	X'410384'	BNJDSERV and AAUTSKLP
ROUTE-INOP	X'410289'	AAUTSKLP
CNM	X'810814'	AAUTSKLP
NMVT	X'41038D'	

NMVT Request Units are routed based upon the Major Vector Key:

Major Vector Key	Responsible Subtask Name
X'0000'	BNJDSERV
X'0001'	BNJDSERV
X'0010'	AAUTSKLP
X'0025'	BNJDSERV
X'006F'	DSIGDS
X'0080'	AAUTSKLP
X'13FF'	BNJDSERV

If the data is a cross domain alert, the first 44 bytes of the data are mapped by the Focal Point Transfer RU (see *NetView Customization: Using Assembler* for a complete mapping) and the remainder of the data is the actual NMVT. The first two bytes of the Focal Point Transfer RU contain the length of the entire buffer (FPT RU + NMVT). The next two bytes contain the header id which is always X'1040'. The 16th byte contains the length of the originating domain id and the 17th through 24th bytes contain the actual originating domain id. When returning a substitute buffer do not modify the Focal Point Transfer RU (the first 44 bytes); replace only the NMVT portion of the buffer (it must be replaced with a valid NMVT).

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 subtask.

If a substitute buffer is returned, the data must be a valid SNA request unit (RU). See *Systems Network Architecture Technical Overview* for a discussion of RU formats.

XITDI: Data Services Task (DST) Initialization

Description: The DST calls XITDI for each statement read by the DST during initialization. When end-of-file is reached, this user exit is entered and the length of the input command buffer is zero. You can code up to 10 module names for each user-written exit routine. See Chapter 3 on page 21 for more information on 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 Code Considerations: XITDI can prevent the DST from processing a definition statement by passing return code USERDROP.

When 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 data base, 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 only use the service facilities available to the DST subtask.

You should avoid coding user exits for frequently executed functions, such as VSAM I/O, since 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. NetView provides its own XITVN for VSAM logs generated under DST. You should code this user exit only if you wish to write your own VSAM subtask using DST as a base.

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

Notes:

1. Only VSAM key-sequenced data sets (KSDS) are supported.
2. Do not replace NetView provided XITVN exits for the DSILOG and DSITRACE subtasks.

Return Code Considerations: 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 data base via 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 subtask. The text portion is mapped by DSILOGDS when using this exit for the DSILOG task.

You should avoid coding user exits for frequently executed functions, such as VSAM I/O, since 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 parameter. For example, session monitor performs external logging of response time and configuration data.

Example of Use: Write user defined data to a user defined log.

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

The following offsets (in byte values) can be used to access the CMDBUF (*Cmdbuf*).

OFFSET	LENGTH	NAME	FUNCTION
0	2	ELBLENG	Unsigned length of header
2	2	ELBRLENG	Unsigned length of record
4	1	ELBTYPE	Log type
5	3	ELBLOG	EBCDIC log type
8	4		Reserved by NetView
12			Start of record

Chapter 3. HLL Data Services Command Processors

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

A Data Services Task (DST) is a set of NetView interfaces built on top of the NetView optional task base. The NetView optional task is discussed in *NetView Customization: Using Assembler*. A DST provides a subtask processing module (DSIZDST) along with the following:

- An initialization exit interface
- A Data Services Command Processor (DSCP) interface that provides support for VSAM (via CNMKIO) and CNMI (via CNMCNMI).
- Various user exit interfaces. (see Table 1 on page 11)

Note: For further information on the TASK and DSTINIT statements referenced in this chapter see *NetView Administration Reference*.

Data Services Task Installation

A TASK statement for the DST subtask must be coded in the DSIDMN member of the DSIPARM data set. The TASK statement has the following format:

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

- MOD keyword - Must specify DSIZDST as the subtask processing module. DSIZDST is provided by NetView and provides the necessary initialization, processing, and termination routines to use the DSCP interfaces.
- TSKID keyword - The task name. Each task in NetView must have a unique task name.
- MEM keyword - Specifies the user-defined initialization member found in DSIPARM to be used by this task. The initialization dataset member must contain DSTINIT statements to provide various initialization parameters required by DSIZDST. The statements will be discussed under their respective interfaces. User-defined statements can also be processed by the initialization exit.
- PRI keyword - Specifies the relative task priority (1-9). 1 is the highest task priority that can be assigned, and 9 is the lowest.
- INIT keyword - Specifies whether the task is to be started during NetView initialization (INIT=Y) or through the START command only (INIT=N).

Initialization

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.

- DSTINIT Keywords - Related to initialization.
  - FUNCT - Specifies which DST services will be 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 keyword - Specifies the name of the user provided initialization exit. The exit is called with the standard NetView user exit interface as documented in Chapter 2 on page 9 and is called once for every statement in the specified initialization member (MEM keyword of TASK statement). When End-Of-File has been reached, the length of CMDBUF (*Cmdbuf*) will be zero. For each statement (except End-Of-File condition), the standard user exit return codes will cause the following actions:

  USERASIS (0) The statement will be processed by the NetView DST module (DSIZDST). If it is not a valid DSTINIT statement, DSIZDST will reject it with an error message and continue processing.

  USERDROP (4) The statement will not be processed by DSIZDST. This return code should be used if your user exit is going to process the statement (you can define your own initialization statements).

  USERSWAP (8) The swapped buffer will be processed by DSIZDST. If the swapped buffer does not contain a valid DSTINIT statement, it will be rejected by DSIZDST and processing will continue.

  When returning from the last call (for End-Of-File), any non-zero return code will terminate the DST. This should only be done if the initialization process has failed.

Data Services Command Processors

Command processors that run under DSTs are called data services command processors. They must be defined as TYPE=D (DST only) or TYPE=RD (Regular or DST). The following services are available to data services command processors (DSCP).

CNM Data Services

An APPL definition with AUTH=CNM must be defined to VTAM for the DST (use the TSKID name as the APPL name). The DST provides access to both solicited and unsolicited CNM data. CNMCNMI can be issued to solicit CNM data from the Network. An HLL DSCP can be defined 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 which is to receive the particular type of data. When a DST is defined with CNMI services, an ACB is opened with an ACB name (the application name) equivalent to the task name as defined by the TSKID parameter in the DST TASK definition statement (the one exception is Hardware Monitor whose CNMI DSTs 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 will be passed to the unsolicited data services command processor for that DST.

  - DSTINIT Keywords - Related to 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 may issue the CNMKIO macro.
    - DSRBU - Specifies the number of unsolicited Data Services Request Blocks (DSRBS) which are to be allocated to this DST. If unsolicited CNM data is not going to be processed by this DST, then this value should be set to zero; otherwise it should be set to one.
  - HLL DSCP Interface

    When the unsolicited HLL DSCP receives control, CNMDBUF (*Cmdbuf*) will contain the unsolicited data RU.

- Solicited CNM Data Interface

  CNMCNMI can be used by an HLL DSCP to acquire Communications Network Management data from the network.

  - DSTINIT Keywords - Related to the solicited CNM Data Interface.
    - DSRBO - Specifies the number of solicited DSRBs that will be required by this task and limits the number of concurrent CNMCNMI and/or CNMKIO requests. This value must be at least 1 (a DSCP will not be called unless a solicited DSRB is available) and no greater than 862.

VSAM Services

The CNMKIO service routine can be invoked by a DSCP to perform I/O for a specified VSAM data set.

- DSTINIT Keywords - Related to CNMKIO service routine.
  - DSRBO - Specifies the number of solicited DSRBS that will be required by this task and limits the number of concurrent CNMCNMI and/or CNMKIO service routine. This value must be at least 1 and no greater than 862, (a DSCP will not be called unless a solicited DSRB is available).
  - PDDNM - Specifies the DD name 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 DD name of the secondary data set to be used by VSAM services. The NetView SWITCH command is used to control which data set is currently the active data set.
  - SPASS - Specifies the VSAM password to be used when the secondary data set ACB is opened.
  - MACRF - Specifies local resource sharing.
  - XITVN - Specifies a user exit to receive control when an empty VSAM data set has been opened for processing. This exit allows you to put an initialization record into the data set.
  - XITVI - Specifies a user exit to receive control upon input from the VSAM data set before the input record is passed to the requesting SCP.
  - XITVO - Specifies a user exit to receive control before output of a record to the VSAM data set.

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

The CNMSMSG service routine can be used to schedule a DSCP and, in conjunction with the WAIT command, can wait for the DSCP to send back the results of the scheduled work. For samples of data services, command processors, and user exit routines, see Appendix B on page 263 for PL/I, and Appendix D on page 295 for C.

Part 2. Coding Your PL/I Program

Chapter 4. Coding Your PL/I Program - Interfaces and Restrictions

This chapter provides necessary information for coding HLL command processors and user exits in PL/I. The appropriate interfaces and language dependent restrictions are discussed.

Initial Parameters

Three parameters are passed to an HLL program upon invocation. Chapter 5 contains a sample template for coding the main procedure statement and the initial parameter declarations in PL/I. The descriptions of the initial parameters are as follows:

HLBPTR
A 4-byte pointer field containing the address of the HLB control block (DSIPHLB). 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.

CMDBUF
A varying length character string that contains the command or message that drove this program.

If this program was driven as a user exit (other than DSIEX02A), this string contains the message that drove this exit. If driven as DSIEX02A, CMDBUF will not contain any useful information. The user will have to retrieve the message from the Initial Data Queue (IDATAQ).

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

HLL Run-Time Options

HLL run-time options can be specified by declaring and initializing the external variable named HLLOPTS. If the user does not code HLL run-time options, the default HLL run time options are assumed. The default value for HLLOPTS is zero. The following bits are defined in HLLOPTS:

Bit Position	Field Name	Description
0	HLL_QUEUED_INPUT	Determines if an HLL program will accept QUEUEd input. Refer to the QUEUE command in the *NetView Operation* manual for further detail. 0 = HLL program will NOT accept QUEUEd input 1 = HLL program will accept QUEUEd input
1	HLL_NO_CANCEL	Determines if an HLL program will terminate on CANCEL/RESET. Refer to RESET command in the *NetView Operation* manual for further detail. 0 = Cancellable 1 = Non-cancellable
2-31		RESERVED for internal use. Do not assign any values to these fields.

The following example illustrates how the default HLL run-time options can be overridden in an HLL program written in PL/I. In this case, the user has chosen to make this PL/I program non-cancellable.

```
 DCL HLLOPTS BIT(32) STATIC EXTERNAL
     INIT('01000000000000000000000000000000'B);
```

PL/I Run-time Options

PL/I run-time options can be specified by declaring and initializing the external variables named PLIOPTS, ISASIZ and HEAPSIZ. If the user does not code PL/I run-time options, the default PL/I run-time options are assumed. Values for the PL/I run-time options are displayed on the entry screen (HAPIENTR) into your PL/I program (ID=PLIENTRY) when monitoring execution of your program using the Remote Interactive Debugger (RID). This is explained in detail in Chapter 11 on page 167.

Run-time options are passed to PL/I using the PLICALLB entry point conventions. Refer to the *PL/I Programming Guide* for a detailed description of each option. The following bits are defined in PLIOPTS.

Bit Position	**Field Name**
0	REPORT
1	**NOREPORT**
2	SPIE
3	**NOSPIE**
4	STAE
5	**NOSTAE**
6	COUNT
7	**NOCOUNT**
8	FLOW
9	**NOFLOW**
10	**KEEPHEAP**
11	FREEHEAP
12	**ANYHEAP**
13	BELHEAP
14-31	Map to the bits defined for PLICALLB entry point conventions.

PL/I programs must run with the NOSTAE and NOSPIE options when running in the NetView environment. Running with the STAE or SPIE options will cause unpredictable results in cases where error recovery is necessary. The following example illustrates how the default PL/I run-time options (PLIOPTS) can be overridden in an HLL program written in PL/I. In this case, the user has chosen to run this program with the REPORT option on. Note that the other PL/I run-time options defined in PLIOPTS (NOSTAE, NOSPIE, etc.) are still specified.

```
DCL   PLIOPTS   BIT(32)   STATIC EXTERNAL
      INIT ('10010101011010000000000000000000'B);
```

External variables ISASIZ and HEAPSIZ correlate to the ISASIZE and HEAP(*size* parameter) run-time options discussed in the *PL/I Programming Guide*. The default values for ISASIZ (4000) and HEAPSIZ (512) are obtained from the NetView Constants Module (CNMS0055). These values can be tailored for your environment. Refer to the *NetView Installation and Administration Guide* for details on how to change the default values in the constants module.

The following example illustrates how the default values for ISASIZ and HEAPSIZ can be overridden in the user's program.

```
 DCL (ISASIZ     INIT(4096),       /* Override ISA size             */
      HEAPSIZ    INIT(256))        /* Override HEAP size            */
                 STATIC EXTERNAL FIXED BIN(31);
```

To achieve optimum performance, it is recommended to run with the REPORT option until accurate ISA and HEAP sizes are determined. Refer to the run-time storage section of the *PL/I Programming Guide* for further details.

Parameters Passed to HLL Service Routines

There are four different types of parameters that can be passed to HLL service routines. Each of the parameters described throughout Chapter 12 fall into one of these categories:

- Pointer Variables
- Integer Variables
- Fixed Length Character Strings
- Varying Length Character Strings

A discussion of each of these parameter types follows. This section describes how each of these parameter types can be declared, initialized and passed to the HLL service routines. Examples and recommendations for writing HLL programs in PL/I have been provided in this chapter. Note that these examples are not complete. They have been included here to emphasize how the HLL service routine parameters should be declared, initialized and passed. For complete examples of user written HLL programs, see the HLL samples shipped with NetView. Refer to Appendix B on page 263.

Pointer Variables

A pointer variable is a 4-byte pointer field containing an address. All HLL service routines require at least one argument of this type, HLBPTR. HLBPTR is required for all HLL service routine invocations. The value of HLBPTR is calculated by NetView and passed to the HLL command processor or user exit. Therefore, it only needs to be declared in PL/I. The user should NEVER assign a value to this variable. This is the only parameter of this type which does not have to be assigned by the user.

Note: The user does not need to specify the HLBPTR parameter when coding the HLL service routine invocation in the PL/I macro format. When an HLL service routine is invoked using the PL/I macro format, HLBPTR is inserted for the user before the HLL service routine is actually invoked.

If an HLL service routine is expecting an address in a pointer field, the user is responsible for assigning a value to that pointer field before invoking the HLL service routine. HLBPTR is the only exception to this rule. In PL/I, it is advised to use the ADDR function when passing pointer variables to HLL service routines rather than creating a separate pointer variable for this purpose. This will ensure that the pointer variable has been assigned a value before invoking the HLL service routine.

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

2  DCL HLBPTR PTR;                  /* HLB pointer MUST BE DECLARED! */
3  DCL SRCPTR PTR;                  /* Source pointer                */
4  DCL DSTPTR PTR;                  /* Destination pointer           */

5  DCL DSTLEN FIXED BINARY(31,0);   /* Length of Destination         */

6  DCL SRCBUF CHAR(255) VARYING;    /* Source buffer                 */
7  DCL DSTBUF CHAR(255) VARYING;    /* Destination buffer            */

8  SRCPTR = ADDR(SRCBUF);           /* Address of source buffer      */
9  DSTPTR = ADDR(DSTBUF);           /* Address of destination buffer */

10 DSTLEN = LENGTH(DSTBUF);         /* Length of destination buffer  */
11 SRCBUF = (255)'A';               /* Initialize source buffer      */
12 DSTBUF = (255)' ';               /* Initialize destination buffer */

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

Figure 1. Using Pointer Variables in PL/I

2

HLBPTR is declared as a pointer (PTR) variable to be used in the CNMCPYS invocation. The user did not assign a value to HLBPTR. HLBPTR is specified for this invocation because the user has chosen to invoke CNMCPYS using the PL/I call format rather than the PL/I macro format of the invocation. Chapter 12 contains examples of how to invoke HLL service routines using the PL/I macro format.

3

SRCPTR is declared as a pointer (PTR) variable.

8

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

13

Both HLBPTR and SRCPTR have been passed as parameters to CNMCPYS.

Replacing **13** with the following step illustrates the use of the ADDR function in PL/I. Using the ADDR function eliminates the need to declare pointer (PTR) variables and is advisable whenever possible. Note the use of a character constant instead of the VARTOVAR variable.

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

Note: If the ADDR function is used to represent a pointer to a varying length character string, warning message IEL0548I will be generated at compile time.

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, etc.. Figure 2 illustrates the use of integer variables in the PL/I environment.

```
1  DCL HLBPTR PTR;                    /* HLB pointer MUST BE DECLARED! */
2  DCL SPNAME CHAR(8) VARYING INIT('POOLNAME'); /* Subpool name      */
3  DCL SPFUNC CHAR(8);                /* Subpool function              */
4  DCL SPTOKEN  FIXED BIN(31,0);      /* Subpool token (returned)      */
5  DCL SPLENG   FIXED BIN(31,0);      /* Cell size                     */
6  DCL SPPRICNT FIXED BIN(31,0);      /* Number of cells in primary    */
7  DCL SPSECCNT FIXED BIN(31,0);      /* Number of cells in secondary  */
8  DCL SPCLASS  FIXED BIN(31,0);      /* Class of storage              */

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

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

Figure 2. Using Integer Variables in PL/I

4

SPTOKEN is declared as a 4-byte integer (FIXED BIN(31,0)).

5

SPLENG is declared as a 4-byte integer (FIXED BIN(31,0)).

10

SPTOKEN is initialized to zero. A value will be returned in SPTOKEN upon successful completion of the CNMPOOL invocation.

11

SPLENG is assigned a value of 256 to be used in the call to CNMPOOL.

15

SPTOKEN and SPLENG have been passed to CNMPOOL. The value of SPTOKEN will be returned to the user upon successful completion of the call to CNMPOOL.

Fixed Length Character Strings

The majority of the HLL service routines require the user to pass one or more fixed length character strings as arguments. Most of these fixed length character strings, except *adorigin* and *gdorigin*, are eight characters long. These exceptions are discussed below.

PL/I constants for most of the fixed length character strings have been provided in DSIPCONS. DSIPCONS is optional and can be tailored to the specific needs of the user. The following steps correlate to the steps outlined in Figure 2.

3
: SPFUNC is declared as an 8-byte character field (CHAR(8)).

9
: Character string 'ALLOC' is assigned to SPFUNC to be used in the call to CNMPOOL.

15
: SPFUNC is passed to CNMPOOL. SPFUNC could have been initialized (see VARTOVAR in Figure 1 step **1**) or passed to CNMPOOL as a character constant as shown here. In all cases, it is important to note that PL/I automatically pads fixed length character fields with blanks.

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

The only fixed length character fields required for HLL services that are not 8 bytes in length are origin blocks. The mapping structure for an origin block resides in file DSIPORIG which is included by DSIPLI. There are two types of origin blocks used by the HLL service routines.

The first type of origin block (ORIGBLCK) is a 40-byte structure which must be declared by the user. This is a required initial parameter which was previously described in the 'Initial Parameter' section of this chapter. The user is responsible for declaring this 40-byte structure but should never need to alter it. Refer to the PL/I coding template in Chapter 5 on page 37 for an example of how to declare ORIGBLCK.

The second type of origin block (*adorigin*, *gdorigin*) is specified by the user. *adorigin* and *gdorigin* must be at least 38 bytes long and must map to the first 38 bytes of the origin block structure (DSIPORIG). The user MUST declare these origin blocks separately from the origin block which is required as an initial parameter. The initial parameter origin block (ORIGBLCK) should NOT be used in place of *adorigin* or *gdorigin*.

Varying Length Character Strings

Several of the HLL service routines require the user to pass a varying length character string as an argument. The following steps correlate to the steps outlined in Figure 2 on page 32.

2
: SPNAME is declared as a varying length character field with a maximum length of 8 bytes (CHAR(8) VARYING). SPNAME is also initialized in this step.

15
: SPNAME is passed to CNMPOOL.

Control Blocks and Include Files

There are a number of control blocks and include files that are required for execution of an HLL program (written in PL/I) in the NetView environment. DSIPLI is the main file that includes the rest of the files and is necessary to compile HLL programs written in PL/I. Optional include files have been provided to assist the user in coding and maintaining HLL programs. DSIPLI, DSIPCNM and DSIPCONS may be tailored to the user's needs.

Note: Tailoring files can lead to better performance in many cases. This is especially helpful in performance sensitive environments such as the user exit environment.

Appendix A on page 245 contains the following list of control blocks and include files:

DSIPLI	(Required) 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 compile and run PL/I programs in the NetView environment. Refer to the PL/I coding template in Chapter 5 on page 37 for usage.
DSIPCNM	(Optional) Declares HLL return code constants for PL/I.
DSIPCONS	(Optional) Declares constants that are helpful when coding High-Level Language programs in PL/I.
DSIPHLB	(Required) PL/I mapping of internal control block DSIHLB.
DSIPHLLS	(Required) PL/I definitions for HLL service routines.
DSIPORIG	(Required) PL/I mapping of the origin block of the request that caused the execution of the program currently running.

PL/I I/O Considerations

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

Each file referenced from your PL/I program correlates to a physical data set in auxiliary storage. Before opening a file for I/O, the user must ensure that the appropriate data set has been allocated. Allocation can be performed under TSO or by using the NetView ALLOCATE command described in *NetView Operation*. NetView also provides a FREE command to deallocate a data set.

If the data set is allocated from TSO, the user must also add a corresponding data definition (DD) statement to the NetView start up 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:

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

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

The following 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          */
       .
       .
       .
/*******************************************************************/
/* Check for error before opening file for I/O.  If UNDEFINEDFILE  */
/* condition is raised, issue an error message end exit program.   */
/*******************************************************************/
ON UNDEFINEDFILE(OUTFILE)
  BEGIN;
    CALL CNMSMSG(HLBPTR,'OUTPUT FILE IS UNDEFINED','MSG','OPER','');
    HLBRC = CNM_GOOD;
    STOP;
  END;

OPEN FILE(OUTFILE) OUTPUT;        /* Open file for output         */
       .
       .
       .
PUT SKIP FILE(OUTFILE) ...        /* Write to output file         */
CLOSE FILE(OUTFILE);              /* Close output file            */
```

If the user chooses to write to a common output file from two or more PL/I programs, access to the common file must be coordinated by the programs. This can be accomplished using NetView's CNMLK routine if desired. If access is not coordinated, the user may experience a system ABEND 213.

Special care should be taken when attempting to share open files between two or more HLL programs. Sharing of open files must be coordinated between the sharing programs. PL/I and C cannot share an open file. However, a C program can read a file created by PL/I.

If the user chooses to code a GET or PUT statement without the FILE option, the compiler will insert the file names SYSIN and SYSPRINT. By default, SYSIN and SYSPRINT are directed to the terminal. These defaults are not valid and will 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 12.

Refer to *PL/I Programming: Language Reference* and *PL/I Programming Guide* for a more detailed discussion on files and PL/I I/O.

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 executed when a particular PL/I error condition is raised. PL/I error conditions can be detected by the operating system or by PL/I. If a PL/I program is running with the NOSTAE or NOSPIE options, only the conditions detected by PL/I can be handled by ON-units. Since PL/I programs running in the NetView environment must run with the NOSTAE and NOSPIE options, the user will not be able to code ON-units for operating system detected conditions. While debugging a PL/I program in the NetView environment, it is allowable to run with the STAE and SPIE options until the run-time problems have been resolved. Most run-time errors are represented by diagnostic messages written to the SYSPRINT file. See *PL/I Programming Guide* for a complete discussion on error and condition handling.

Considerations for HLL Command Processors

It is necessary to code a CMDMDL statement in DSICMD for each HLL command processor that you have written. CMDMDL TYPE will be dependent on the functions that your command processor performs. Keep in mind that some of the HLL services are only useful when executed under a Data Services Task (DST). There is no support for HLL command processors running as immediate commands (TYPE=I). The CMDMDL statement is described in *NetView Administration Reference*.

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. This field should be checked after each HLL service routine invocation. It is recommended that this field be utilized when passing return codes between HLL programs.

For a complete list of HLL API return codes, see DSIPCNM in Appendix A. Refer to Chapter 12 for a list of return codes that apply to each HLL service routine.

PLIRETV and PLIRETC should not be used when passing return codes between HLL programs written in PL/I. Both of these routines could yield unpredictable results in the NetView environment. Normal termination of a PL/I program can be achieved by assigning a value to HLBRC and issuing a RETURN statement as shown here.

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

Restrictions for HLL Programs Written in PL/I

The following commands should not be used when coding PL/I programs to run in the NetView environment:

```
DISPLAY             Use NetView's CNMSMSG service routine.
WAIT, DELAY         Use NetView's WAIT command.

ON FIXEDOVERFLOW    These condition codes will not work
ON OVERFLOW         in NetView since they require SPIE
ON UNDERFLOW        and STAE.
ON ZERODIVIDE

PLIRETV             Return codes should be passed via
PLIRETC             HLBRC.
```

Chapter 5. PL/I High-Level Language Services

This chapter is an example-oriented discussion of commands and services provided by NetView in support of PL/I. The complete syntax and usage of each command and service routine can be found in Chapter 12.

Note: When you are compiling PL/I programs you will receive a warning message IEL0548I. This message should be ignored.

PL/I Sample Template

The following is a coding template sample to be used when coding HLL programs in PL/I. This template can be used, with your enhancements, to utilize NetView functions and commands. Further examples in this chapter should be used in conjunction with this template.

```
PTMPPLT: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/********************************************************************/
/*                                                                  */
/*   (C) COPYRIGHT IBM CORP. 1989                                   */
/*                                                                  */
/*   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 a user 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.         */
/*                                                                 */
/*   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.                                )        */
/*******************************************************************/

/*******************************************************************/
/* NetView High-Level Language include files                       */
/*******************************************************************/
%INCLUDE DSIPLI;                  /* Include the HLL macros        */

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

```
/*********************************************************************/
/* Other declarations                                                */
/*********************************************************************/
DCL ORIGIN   PTR;                 /* Pointer to the Origin Block   */
DCL ADDR     BUILTIN;             /* Builtin function              */

/*********************************************************************/
/* Initialization                                                    */
/*********************************************************************/
ORIGIN=ADDR(ORIGBLCK);            /* Address of the Origin Block   */

/*********************************************************************/
/* Execution                                                         */
/*********************************************************************/
HLBRC = CNM_GOOD;                 /* Successful completion         */
END PTMPPLT;
```

Data Queue Management

NetView utilizes several data and message queues to work in conjunction with HLL service routines. Information retrieved from these queues, by the GETDATA function, can be manipulated to enhance your network manageability. The following five queues are defined for data and message management.

TRAPQ	Queue 1	This queue enables the user to access messages placed on it after being trapped as a result of an issuance of the TRAP command for messages.
OPERQ	Queue 2	This queue enables the user to access operator input, entered by the GO or QUEUE command.
DATAQ	Queue 3	This queue enables the user to access DATA type messages placed on it by the send message HLL service routine (CNMSMSG).
IDATAQ	Queue 4	The initial data queue enables the user to access the message that invoked the HLL command processor by the message automation table or which drove DSIEX02A.
CNMIQ	Queue 5	This queue enables the user to access CNMI solicited data which was solicited by an issuance of the HLL CNMI service routine (CNMCNMI).

Sending Information

The following is an example of sending messages to different destinations.

```
/**********************************************************************/
/*                     SEND A MULTILINE MESSAGE TO USER               */
/**********************************************************************/
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','');

/**********************************************************************/
/*                     SEND A MULTILINE MESSAGE TO A TASK             */
/**********************************************************************/
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');

/**********************************************************************/
/*                     SEND A MESSAGE TO THE CONSOLE (only 1-liners)*/
/**********************************************************************/
CALL CNMSMSG(HLBPTR,'Hello Sysop    ','MSG','SYSOP','');

/**********************************************************************/
/*                     SEND A MESSAGE TO THE AUTHORIZED RECEIVER      */
/**********************************************************************/
CALL CNMSMSG(HLBPTR,'Hello Authrcvr ','MSG','AUTHRCV','');

/**********************************************************************/
/*                     SEND A MESSAGE TO THE NETWORK LOG              */
/**********************************************************************/
CALL CNMSMSG(HLBPTR,'This should only be in log','MSG','NETVLOG','');

/**********************************************************************/
/*                     SHOW THAT YOU CAN SEND TO SEQLOG               */
/**********************************************************************/
CALL CNMSMSG(HLBPTR,'test msg','MSG','SEQLOG','SQLOGTSK');

/**********************************************************************/
/*                     SHOW THAT YOU CAN SEND TO A GROUP              */
/**********************************************************************/
CALL CNMSMSG(HLBPTR,'hello group','MSG','OPCLASS','+GROUP1');
```

Parsing Input Strings

Parsing Input String Similar to NetView Command List Language

The following is an example of parsing the input string similar to the NetView command list language. It will parse the first 10 tokens individually, just as NetView command list language would parse them into &1, &2, etc.. It will also set an equivalent variable to &PARMSTR.

```
/********************************************************************/
/*                                                                  */
/*  Other Declarations                                              */
/*                                                                  */
/********************************************************************/
DCL (CLIST1,CLIST2,CLIST3,
     CLIST4,CLIST5,CLIST6,
     CLIST7,CLIST8,CLIST9,
     CLIST10,PARMSTR)
            CHAR(255) VARYING;
DCL PARMCNT FIXED    BIN(31,0);   /* Number of tokens parsed       */
/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/
CNMSSCAN                          /* Parse like NetView Command
                                     ...List Language parses       */
  DATA(CMDBUF)                    /* ...input is in cmdbuf         */
  FORMAT('%*S%S%S%S%S%S%S%S%S%S%S')/* ...parse each token by blanks */
                                  /* ...but skip the command       */
  COUNT(PARMCNT)                  /* ...number of tokens parsed    */
  P1(CLIST1)                      /* ...first token                */
  P2(CLIST2)                      /* ...next token                 */
  P3(CLIST3)                      /* ...next token                 */
  P4(CLIST4)                      /* ...next token                 */
  P5(CLIST5)                      /* ...next token                 */
  P6(CLIST6)                      /* ...next token                 */
  P7(CLIST7)                      /* ...next token                 */
  P8(CLIST8)                      /* ...next token                 */
  P9(CLIST9)                      /* ...next token                 */
  P10(CLIST10);                   /* ...last token                 */

CNMSSCAN                          /* Parse like NetView Command
                                     ...List Language parses       */
  DATA(CMDBUF)                    /* ...input is in cmdbuf         */
  FORMAT('%*S%{¬}')               /* ...skip over command, then    */
                                  /* ...put all parms in a target  */
  COUNT(PARMCNT)                  /* ...number of tokens parsed    */
  P1(PARMSTR);                    /* ...first token                */
```

Parsing Input String Similar to REXX

The following is an example of parsing input strings. Using the following example, PL/I will parse the first 4 tokens individually, and then put the rest of the input into the fifth token. The REXX language parses the following statement in this way.

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

```
/*********************************************************************/
/*                                                                   */
/*  Other Declarations                                               */
/*                                                                   */
/*********************************************************************/
DCL (TOKEN1,TOKEN2,TOKEN3,
     TOKEN4,TOKEN5,MSGSTR)
             CHAR(255) VARYING;
DCL PARMCNT  FIXED    BIN(31,0);  /* Number of tokens parsed        */
/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/
CNMSSCAN                          /* Parse like TOKEN parses...     */
  DATA(CMDBUF)                    /* ...input is in cmdbuf          */
  FORMAT('%*S%S%S%S%S%S%{¬}')     /* ...parse each token by blanks  */
                                  /* ...but skip the command and    */
                                  /* ...last token gets rest of     */
                                  /* ...the input.                  */
  COUNT(PARMCNT)                  /* ...number of tokens parsed     */
  P1(TOKEN1)                      /* ...first token                 */
  P2(TOKEN2)                      /* ...next token                  */
  P3(TOKEN3)                      /* ...next token                  */
  P4(TOKEN4)                      /* ...next token                  */
  P5(TOKEN5);                     /* ...last token gets the rest
                                     ...of the input string         */

CNMSSCAN                          /* Parse like ARG parses...       */
  DATA(CMDBUF)                    /* ...input is in cmdbuf          */
  FORMAT('%*S%{¬}')               /* ...skip over command, then     */
                                  /* ...put all parms in a target   */
  COUNT(PARMCNT)                  /* ...number of tokens parsed     */
  P1(MSGSTR);                     /* ...first token                 */
```

Parsing Input String

The following is another example of parsing. Based upon the C SSCANF function, CNMSCAN allows for both parsing and conversion in a single step, and allows delimiting the input on any character desired.

```
/*********************************************************************/
/*                                                                   */
/*  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                           */
/*********************************************************************/
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 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 can 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                  */

/*********************************************************************/
/*  After executing, the variables have the following 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"                                                 */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
```

Synchronous Commands

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

```
/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/

/* Issue the VTAM command D NET,APPLS                                */

CALL CNMCMD(HLBPTR,                  /* Invoke the command...        */
            'D NET,APPLS');          /* ...text of the command to run */
```

Sending Commands

The following 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 VTAM command, or a NetView command.

You can use this process to execute commands under data services tasks (DST), other operator station tasks (OST), or the primary POI task (PPT).

```
/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/

/* Issue the LOGOFF command on a task called OPER1                   */

CALL CNMSMSG(HLBPTR,                  /* Send the command...         */
            'LOGOFF',                 /* ...text of the command to run */
            'COMMAND',                /* ...this is a command        */
            'TASK',                   /* ...run it on a task         */
            'OPER1');                 /* ...task name is OPER1       */
SELECT;
  WHEN(HLBRC=CNM_GOOD)
    CALL CNMSMSG(HLBPTR,              /* Inform user of success...   */
            'OPER1 logoff successfully scheduled', /* ...text of msg*/
            'MSG',                    /* ...this is a message        */
            'OPER',                   /* ...to the operator          */
            '');                      /* ...not used                 */
  WHEN(HLBRC=CNM_TASK_INACTIVE)
    CALL CNMSMSG(HLBPTR,              /* Inform user task not active...*/
            'OPER1 not active',       /* ...text of message          */
            'MSG',                    /* ...this is a message        */
            'OPER',                   /* ...to the operator          */
            '');                      /* ...not used                 */
  OTHERWISE
    CALL CNMSMSG(HLBPTR,              /* Inform user bad rc...       */
            'Unexpected RC from CNMSMSG', /* ...text of message      */
            'MSG',                    /* ...this is a message        */
            'OPER',                   /* ...to the operator          */
            '');                      /* ...not used                 */
 END;                                 /* of select                   */

 HLBRC=CNM_GOOD;                      /* Clear RC                    */
```

Waiting and Trapping

The following is an example of how to issue a command, trap the output of the command, and respond depending on the output that is encountered. It will activate the given LU and issue an appropriate message.

The syntax that it checks for is:

PACTLU luname

Where luname is the name of the LU to be activated

```
/*********************************************************************/
/*                                                                   */
/*  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       */
/*********************************************************************/
/*               Scan the input for the lu name 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 CNMCMD(HLBPTR,             /* Trap the following VTAM msgs  */
             ' TRAP AND SUPPRESS ONLY MESSAGES IST*');
    CALL CNMCMD(HLBPTR,' V NET,ACT,ID='||NODENAME); /* Activate node*/
    CALL CNMCMD(HLBPTR,' WAIT 10 SECONDS FOR MESSAGES'); /* Wait... */
    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          */

    /****************************************************************/
    /* Loop through messages until IST093I is found or no more      */
    /* ...messages are left                                         */
    /****************************************************************/
    DO WHILE(GETPTR->ORIG_BLOCK.ORIG_PROCESS¬='IST093I' &
             HLBRC=CNM_GOOD);
      CALL CNMCMD(HLBPTR,' WAIT CONTINUE'); /* Wait for next msg... */
      CALL CNMGETD(HLBPTR,          /* Get the next trapped msg...   */
              'GETMSG',             /* ...function is get a msg      */
               INBUF,               /* ...result goes here           */
               255,                 /* ...max input length           */
               GETBLOCK,            /* ...must provide a work area   */
               TRAPQ,               /* ...message is trapped         */
               1);                  /* ...get the top one on queue   */
    END;
    IF (HLBRC=CNM_GOOD              /* Did we find IST093I?          */
       GETPTR->ORIG_BLOCK.ORIG_PROCESS='IST093I') THEN
      CALL CNMSMSG(HLBPTR,          /* Inform user activation worked */
            'RESOURCE '||NODENAME||' NOW ACTIVE',
                                    /* ...text of message            */
            'MSG',                  /* ...single line message        */
            'OPER',                 /* ...to the operator            */
            '');                    /* ...not needed                 */
    ELSE                            /* IST093I not found, must be
                                       ...an error                   */
      CALL CNMSMSG(HLBPTR,          /* Inform user activation failed */
            'ERROR - ACTIVATION UNSUCCESSFUL',
                                    /* ...text of message            */
            'MSG',                  /* ...single line message        */
            'OPER',                 /* ...to the operator            */
            '');                    /* ...not needed                 */
  END;
ELSE                                /* Nodename not specified        */
  CALL CNMSMSG(HLBPTR,              /* Inform user need more args    */
          'ERROR - NODENAME NOT SPECIFIED',
                                    /* ...text of message            */
          'MSG',                    /* ...single line message        */
          'OPER',                   /* ...to the operator            */
          '');                      /* ...not needed                 */
```

Retrieving Information

The following gives an example of how an HLL command processor or user exit routine can retrieve information from NetView. Assembler language command processors and user exit routines need DSECTs to access information about NetView. HLL command processors and user exit routines can access some of this information as shown below. Many variables are available. Please refer to the command and service routine reference "CNMINFC (CNMINFOC) – Query NetView Character Information" on page 209 and "CNMINFI (CNMINFOI) – Query NetView Integer Information" on page 211 for an exhaustive list of the values supported.

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

/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/
  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 */
                                   /* ...that the terminal supports */
               'COLORS',           /* ...specify the variable       */
               IDATA);             /* ...result goes here           */
```

Command List Variable Access

The following example illustrates the capability of updating common global variables. This example simply increments a global variable named "GVARIABLE" by 1.

Task globals are updated and read the same way. The only difference is the pool name that is specified.

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

/*********************************************************************/
/*            Find 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   */
```

Using Locks

The previous example illustrated the capability of updating common global variables, but it did not 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. This example has been modified 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 lock method shown below or you may wish to run all the updates on a given task. Since only one process can occur on a task at a time, the updates will be serialized. Note that 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                  */
```

Operator Input

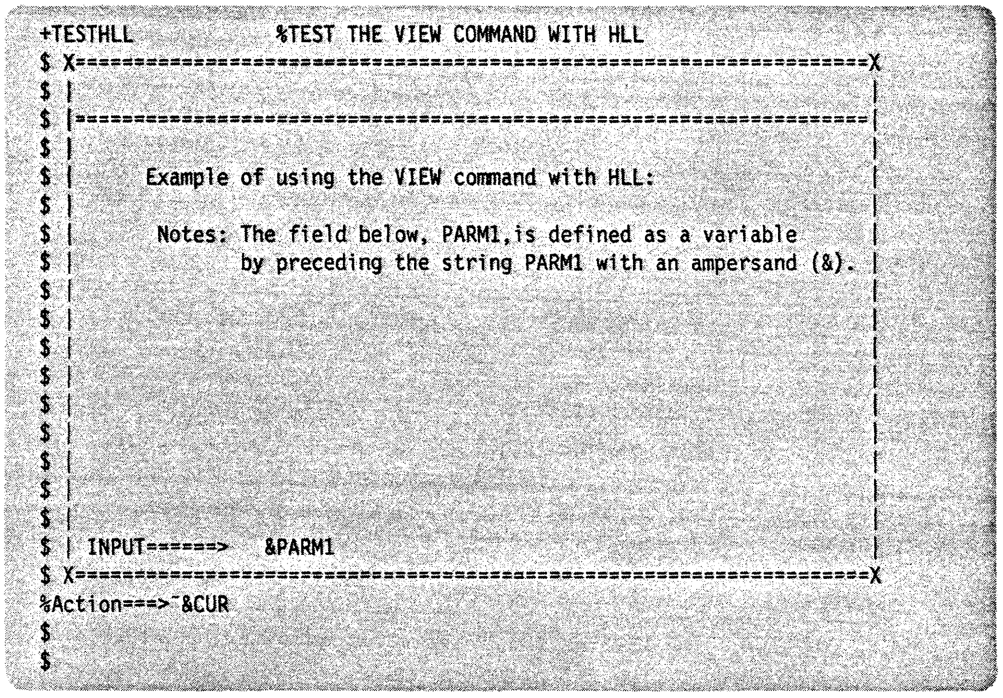
The following is an example of how to code an HLL command processor to accept operator input in single-line mode. The interface is similar to the &PAUSE function of the NetView command list language. Input is requested by the application using the WAIT FOR OPINPUT command, input is retrieved by the application using the CNMGETD service routine and the operator can respond by using the GO command.

```
/********************************************************************/
/*                                                                  */
/*  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:||DATA_IN,   /* ...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                   */
```

VIEW Command Processor

The following is an example of using the full-screen VIEW command processor. First it creates the local variable called PARM1, and the variable is initialized. The VIEW command processor is invoked, displaying a full-screen panel. The following panel is used as input by the VIEW command. For more information on VIEW see the *NetView Customization Guide*.

The panel that is invoked by the following example appears below:



```
+TESTHLL            %TEST THE VIEW COMMAND WITH HLL
$ X==============================================================X
$ |                                                              |
$ |==============================================================|
$ |                                                              |
$ |      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                                       |
$ X==============================================================X
%Action===>¬&CUR
$
$
```

```
/*********************************************************************/
/*                                                                   */
/*  Other Declarations                                               */
/*                                                                   */
/*********************************************************************/
DCL DATA_INCHAR(48) VAR;           /* Input buffer for results       */

/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/
CNMVARPOOL FUNC('DCL')             /* Declare to local pool...       */
                                   /* ...prior to invoking VIEW      */
           NAME('PARM1')           /* ...name is Parml               */
           POOL('LOCAL');          /* ...the pool is local           */

CNMVARPOOL FUNC('PUT')             /* Initialize PARM1...            */
           DATA('the contents of parml go here') /* ...data          */
           NAME('PARM1')           /* ...name of local variable      */
           POOL('LOCAL');          /* ...the pool is local           */

/* Invoke the VIEW command.  Give the task name as a unique          */
/*   name to go on the View Stack.                                   */

CNMCOMMAND DATA('VIEW '|| ORIGIN->ORIG_TASK||' TESTHLL NOMSG NOINPUT');
```

Message Processing

The following example lists the message attributes of a message. The invocation must be as a result of an entry in the message automation table, which is documented in the *NetView Administration Reference*. This example will function correctly for both single line messages and multiple line messages.

```
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 DATA_IN CHAR(12)  VAR;         /* Attribute result             */
DCL ATTR(12) CHAR(8) INIT(         /* 12 message attributes:       */
  'AREAID' ,'DESC'    ,'JOBNAME','JOBNUM','MCSFLAG' ,'MSGTYP',
  'REPLYID','ROUTCDE','SESSID' ,'SMSGID','SYSCONID','SYSID');
/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/
GETPTR=ADDR(GETBLOCK);             /* Address the Orig Block       */
CALL CNMGETD(HLBPTR,               /* Get the first line of the msg */
            'GETMSG',              /* ...function is get a msg     */
             INBUF,                /* ...result goes here          */
             256,                  /* ...max input length          */
             GETBLOCK,             /* ...must provide a work area  */
             IDATAQ,               /* ...message from automation   */
             1);                   /* ...get the first line of msg */
DO WHILE(HLBRC=CNM_GOOD            /* Loop through the messages... */
      HLBRC=CNM_DATA_TRUNC);         /* ...ignoring truncation     */
  DO I=1 TO 12;                    /* For 12 possible attributes... */
    CALL CNMGETA(HLBPTR,           /* Get the Ith attribute...     */
                 ATTR(I),          /* ...Ith member of array       */
                 DATA_IN,          /* ...result goes here          */
                 12,               /* ...at most 12 bytes          */
                 IDATAQ);          /* ...on the initial data queue */
    CALL CNMSMSG(HLBPTR,ATTR(I)||' = '||DATA_IN,'MSG','OPER','');
  END;
  CALL CNMSMSG(HLBPTR,'LINETYPE = '||GETPTR->ORIG_BLOCK.ORIG_LINE_TYPE,
             'MSG','OPER','');
  CALL CNMSMSG(HLBPTR,'HDRMTYPE = '||GETPTR->ORIG_BLOCK.ORIG_MSG_TYPE,
             'MSG','OPER','');
  CALL CNMSMSG(HLBPTR,'MSGID    = '||GETPTR->ORIG_BLOCK.ORIG_PROCESS,
             'MSG','OPER','');
  CALL CNMSMSG(HLBPTR,'MSGSTR   = '||INBUF,
             'MSG','OPER','');
  CALL CNMGETD(HLBPTR,             /* Get next line of message...  */
             'GETLINE',            /* ...function is get next line */
              INBUF,               /* ...result goes here          */
              255,                 /* ...max input length          */
              GETBLOCK,            /* ...must provide a work area  */
              IDATAQ,              /* ...message is from automation */
              1);                  /* ...get the next line         */
 END;                              /* Of DO WHILE                  */
 HLBRC=CNM_GOOD;                   /* Clear RC                     */
```

Scope Checking

The following is an example of the scope checking capabilities provided by NetView. In this example, the user is required to set up the following elements for the command (shown below):

1. operator id
2. operator classes that can access the command
3. operator profile

The command gives the return code that the scope check service routine returned to the operator.

The syntax that this command checks for is:

```
PSPCCKO PARMx(VALx)
```

The following is the setup for the scope check example.

In DSIPARM(DSICMD):

- Define the operator classes that can access the command, its keywords, and its keyword values.
- The example below says that the command PSPCCKO can be executed by operators in scope class 1 and 2. Scope class 1 can issue any keyword or keyword value, but scope class 2 cannot use the value of VAL1 with keyword PARM2, and scope class 2 cannot issue PARM3 at all.

```
PSPCCKO   CMDMDL    MOD=PSPCCKO,RES=N,TYPE=RD
          CMDCLASS  1,2
PARM2     KEYCLASS  1,2
VAL1      VALCLASS  1
PARM3     KEYCLASS  1
VAL1      VALCLASS  1
```

In DSIPARM(DSIOPF):

- Define the operator ids and the profiles that the operator ids can use.

```
JOE      OPERATOR  PASSWORD=USER
         PROFILEN  DSIPROF3
```

In DSIPRF(profilename):

- Define the operator class value that will correspond to the profile that the operator logs on with.

```
DSIPROF3   PROFILE
           OPCLASS 3
           END
```

```
/********************************************************************/
/*                                                                  */
/*  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                                                       */
/*                                                                  */
/********************************************************************/

/********************************************************************/
/*                 Scan the keyword and the value                   */
/********************************************************************/
CALL CNMSCAN(HLBPTR,               /* Parse the input ...           */
            CMDBUF,                /* ...command line is the input  */
                                         /* SYNTAX OF COMMAND IS:   */
                                   /*  CMDNAME KEYWORD(KEYVALUE)    */
                                   /*                               */
                                   /* Scan for the:                 */
            '%S'||                 /* ...command name               */
            '%*{ }'||              /* ...skip over leading blanks   */
            '%{¬(}'||              /* ...keyword up to "("          */
            '%*C'||                /* ...skip over "("              */
            '%{¬)}',               /* ...keyvalue up to ")"         */
            CNT,                   /* ...number strings parsed      */
            CMDNAMEV,              /* ...command goes here          */
            KEYWORDV,              /* ...keyword goes here          */
            KEYVALUEV);            /* ...keyvalue goes in here      */
CMDNAME=CMDNAMEV;                  /* Get fixed length value        */
KEYWORD=KEYWORDV;                  /* Get fixed length value        */
KEYVALUE=KEYVALUEV;                /* Get fixed length value        */
IF CNT=3 THEN                      /* Enough parms specified?       */
  CALL CNMSCOP(HLBPTR,             /* Scope check the input...      */
             CMDNAME,              /* ...the command                */
             KEYWORD,              /* ...the keyword                */
             KEYVALUE);            /* ...the value                  */

ELSE                               /* Not enough parms specified    */
  HLBRC=CNM_BAD_INVOCATION;        /* Set rc                        */
```

```
/*********************************************************************/
/*                 Inform user of the return code results...        */
/*********************************************************************/
 SELECT;
   WHEN(HLBRC=CNM_GOOD)
     DO;                            /* Operator                     */
                                    /*      has                     */
                                    /*          passed              */
     END;                           /*              scope checking  */
   WHEN(HLBRC=CNM_KEYWORD_NA)
     CALL CNMSMSG(HLBPTR,' Not authorized to use KEYWORD '||KEYWORD,
           'MSG','OPER','');
   WHEN(HLBRC=CNM_VALUE_NA)
     CALL CNMSMSG(HLBPTR,' Not authorized to use VALUE   '||KEYVALUE,
           'MSG','OPER','');
   WHEN(HLBRC=CNM_BAD_INVOCATION)
     CALL CNMSMSG(HLBPTR,' Not enough parms specified',
           'MSG','OPER','');
   OTHERWISE
     CALL CNMSMSG(HLBPTR,' RC not recognized...'||HLBRC,
           'MSG','OPER','');
 END;
 HLBRC=CNM_GOOD;                    /* Clear RC                     */
```

Altering Data

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

```
Example output with input of WHO:

Without exit:
  WHO

With exit:
  Command entered was: "WHO" at 12:00:00
```

```
 DSIEX2A: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
 /*********************************************************************/
 /*                                                                   */
         .                     .                   .
         .                     .                   .
         .                     .                   .
 /*   Change Activity:                                                */
 /*      date,author: description of changes                          */
 /*********************************************************************/

 /*********************************************************************/
 /*                                                                   */
 /*  Parameter 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;               /* Builtin 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          */
CNMGETDATA                          /* 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     */
HLBRC=CNM_GOOD;                     /* Clear RC                      */
END DSIEX2A;
```

Storage Access

The following example illustrates how to display the character representation of the contents of the storage that NetView 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. For NetView R3 this will contain the character string NV13.

```
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                  */
/*******************************************************************/
/*                                                                 */
/*  Execution                                                      */
/*                                                                 */
/*******************************************************************/
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
                                      ... a length?                 */
    CNMSENDMSG                     /* No, give error message...     */
           DATA('INVALID NUMBER OF PARAMETERS') /* ...text          */
           MSGTYPE('MSG')          /* ... message                   */
           DESTTYPE('OPER');       /* ... to the operator           */
  WHEN(NUM_BYTES<=0)               /* Did they give a valid length ?*/
    CNMSENDMSG                     /* No, give error message...     */
           DATA('INVALID LENGTH GIVEN') /* ...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               */
               FROM(SOURCE_PTR)        /* ...from the address given*/
               TO(ADDR(INPUT_BFR))     /* ...to the internal buffer*/
               LENG(NUM_BYTES)         /* ...for up to FFF bytes   */
               COPYTYPE('FIXTOFIX');   /* ...data is fixed len vars*/
    IF HLBRC = CNM_GOOD THEN           /* Good RC ?                */
      DO I=1 TO NUM_BYTES BY 64;       /* Display storage          */
        CNMSENDMSG DATA(SUBSTR(INPUT_BFR,I,64))      /* ...64-byte  */
                   MSGTYPE('MSG')      /*  ...in a message         */
                   DESTTYPE('OPER');   /* ...to the operator       */
      END;
    ELSE                               /* Bad RC --                */
      CNMSENDMSG                       /* Send message ...         */
                 DATA('INVALID OR PROTECTED ADDRESS')   /* ...text  */
                 MSGTYPE('MSG')        /* ...in a message          */
                 DESTTYPE('OPER');     /* ...to the operator       */
  END;                                 /* of otherwise             */
END;                                   /* of select                */
```

Data Set Access

The following is an example of opening (using CNMMEMO), reading (using CNMMEMR) and closing (using CNMMEMC) NetView partitioned data sets. This example reads a member of DSIPARM called DSIDMN, and displays it 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='DSIDMN';
/*******************************************************************/
/*                   OPEN THE MEMBER                               */
/*******************************************************************/
CALL CNMMEMO(HLBPTR,               /* Open the data set member ... */
             TOKEN,                /* ... token returned by HLL    */
             DDNAME,               /* ... ddname of PDS            */
             MEMBER);              /* ... member name of PDS       */
IF HLBRC¬=CNM_GOOD THEN
  CALL CNMSMSG(HLBPTR,             /* OPEN failed...               */
               'OPEN FOR DATA SET FAILED RC='||CHAR(HLBRC),
               'MSG',              /* ...single line message       */
               'OPER',             /* ...to the operator           */
               '');                /* ...taskname ignored          */
ELSE
  DO;                              /* Open was successful...       */
    /***************************************************************/
    /*                   READ THE MEMBER                           */
    /***************************************************************/
    CALL CNMMEMR(HLBPTR,           /* Read the first record...     */
             TOKEN,                /* ... provide token from OPEN  */
             MRDATA,               /* ... result goes here         */
             80);                  /* ... read 80 bytes            */
    DO WHILE (HLBRC=CNM_GOOD);     /* Read til EOF                 */
      CALL CNMSMSG(HLBPTR,         /* Write out last record read... */
               SUBSTR(MRDATA,1,72),/* ...write first 72 bytes      */
               'MSG',              /* ...single line message       */
               'OPER',             /* ...to the operator           */
               '');                /* ...taskname ignored          */
      CALL CNMMEMR(HLBPTR,         /* Read the next record...      */
               TOKEN,              /* ... provide token from OPEN  */
               MRDATA,             /* ... result goes here         */
               80);                /* ... read 80 bytes            */
    END;
    /***************************************************************/
    /*               CLOSE THE MEMBER                              */
    /***************************************************************/
    CALL CNMMEMC(HLBPTR,           /* Close the PDS member...      */
             TOKEN);               /* ... using the token from OPEN */
  END;                             /* End of Open was successful... */
```

CNMI

NetView provides the CNMCNMI service routine for use in communicating with devices in the network via the Communications Network Management Interface (CNMI). Any data that is returned may be accessed using the CNMGETD service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

The following example uses the CNMCNMI service routine to send a request product set id data request to a specified PU. Any data returned is sent as a message to the operator.

The syntax of the command is:

```
PNMVTPU puname <OWN|ALL>
```

where:

puname is the name of the PU to be retrieved (required)

OWN implies that vital product data is to be retrieved for the PU only (default)

ALL implies that vital product data is to be retrieved for the PU and its attached ports

```
/*******************************************************************/
/*                                                                 */
/*  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 CNM 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                                        */
/*******************************************************************/

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 in command line   */
             '%*S%S%S',            /* ...skip over the command      */
             COUNT,                /* ...number of args parsed      */
             PUNAMEV,              /* ...puname                     */
             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;
  OTHERWISE                        /* Invalid parm... tell user     */
    DO;
      CALL CNMSMSG(HLBPTR,'INVALID COMMAND SYNTAX', /* wrong...     */
                   'MSG','TASK',ORIGIN ->ORIG_TASK);    /* ...syntax */
      RCODE=8;                     /*  Bad syntax                   */
    END;
END;                               /* Of Select                     */
IF RCODE = 0 THEN                  /* Good so far?                  */
  DO;                              /* Yes, continue                 */
    CALL CNMCNMI(HLBPTR,           /* Send RU over the CNMI...      */
                 'SENDRPLY',       /* ...expect a reply             */
                 FORWARD_RU,       /* ...RU built above             */
                 PUNAME,           /* ...to the PUNAME specified    */
                 180);             /* ...timeout after 3 minutes    */
    IF HLBRC=CNM_GOOD THEN         /* Everything OK?                */
      DO;                          /* Yes, continue                 */
        CALL CNMGETD(HLBPTR,       /* Read in the first RU returned */
                 'GETLINE',        /* ...a single RU                */
                  DATAIN,          /* ...into here                  */
                  1024,            /* ...truncate after 1024-bytes  */
                  GETBLOCK,        /* ...provide a new origblock    */
                  CNMIQ,           /* ...on the CNMI queue          */
                  1);              /* ...the first RU               */
        DO WHILE(HLBRC=CNM_GOOD);  /* End of queue reached?          */
          CALL CNMSMSG(HLBPTR,     /* Send info to the operator...   */
                  DATAIN,          /* ...from here                   */
                  'MSG',           /* ...issue message               */
                  'TASK',          /* ...to the task                 */
                  ORIGIN ->ORIG_TASK); /* ...that originated request*/
          CALL CNMGETD(HLBPTR,     /* Read in the next RU returned   */
                 'GETLINE',        /* ...a single RU                 */
                  DATAIN,          /* ...into here                   */
                  1024,            /* ...truncate after 1024-bytes   */
                  GETBLOCK,        /* ...provide a new origblock     */
                  CNMIQ,           /* ...on the CNMI queue           */
                  1);              /* ...the next RU                 */
        END;                       /* of DO WHILE                    */
      END;                         /* Of everything ok               */
    ELSE
      DO;                          /* CNMI error                     */
        SELECT(HLBRC);
          WHEN(CNM_BAD_INVOCATION)       /* Not invoked under a DST */
            CALL CNMSMSG(HLBPTR,'Must run under a DST',
                  'MSG','TASK',ORIGIN ->ORIG_TASK);

          WHEN(CNM_BAD_TIMEOUT)         /* PU never answered request*/
            CALL CNMSMSG(HLBPTR,'PU never answered',
                  'MSG','TASK',ORIGIN ->ORIG_TASK);

          WHEN(CNM_NEG_RESPONSE)       /* PU gave a negative response*/
            CALL CNMSMSG(HLBPTR,'PU gave a negative response',
                  'MSG','TASK',ORIGIN ->ORIG_TASK);
          OTHERWISE
            CALL CNMSMSG(HLBPTR,'CNMI request failed RC='||
                  CHAR(HLBRC),'MSG','TASK',ORIGIN->ORIG_TASK);
        END;                       /* Of Select                      */
      END;                         /* of CNMI error                  */
  END;                             /* of Good so far                 */
HLBRC=RCODE;                       /* Issue rc                       */
```

VSAM (Keyed File Access)

The following is an example of 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.

This example will create a data base that contains 5 records with the following keys and data:

KEY	DATA
01	A
02	B
03	C
04	D
05	E

```
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(   /* Data                         */
   'A','B','C','D','E');
DCL KEYDATA(5) CHAR(2) VAR INIT(   /* Keys                         */
   '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...       */
              'CNMKEYIO 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;
  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...       */
              'CNMKEYIO 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;
/*********************************************************************/
/*                    READ IN THE 5 RECORDS...                       */
/*********************************************************************/
 DO I = 1 TO HBOUND(OUTDATA,1);   /* For 5 records                */
   KEY=KEYDATA(I);                /* Set key portion of record    */
   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            */
              'NOUPDATE');        /* ... this is not an update    */
   IF HLBRC¬=CNM_GOOD THEN
     CALL CNMSMSG(HLBPTR,         /* Issue error message...       */
              'CNMKEYIO GET REQUEST FAILED, RC='||CHAR(HLBRC),
                                  /* ... text of message          */
              'MSG',              /* ... single line message      */
              'TASK',             /* ... to the task              */
             ORIGIN->ORIG_BLOCK.ORIG_TASK);     /* ...that invoked*/
 END;

 HLBRC=CNM_GOOD;                  /* Issue clean rc               */
```

DST User Exit

The following is an example of coding a Netview HLL user exit routine that primes an empty VSAM data base for a DST. If a VSAM data base has not been primed (has at least one record), subsequent I/O requests will fail.

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

/********************************************************************/
/*                                                                  */
/*  Parameter 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;              /* Builtin 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 data base.  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;
```

User Exit

The following is an example of coding a user 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 will not be set. The PSNDDAT command (see "SEND Side" on page 75) is used to interrogate the variable value.

```
PSETTG: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/*********************************************************************/
/*                                                                   */
/*   Descriptive Name: High Level Language PL/I DSIEX03 Example     */
               .           .              .
               .           .              .
               .           .              .
/*   Change Activity:                                                */
/*      date,author: description of changes                          */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*  Parameter 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;              /* Builtin 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;
```

Wait for Data

WAIT Side

The following is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example is 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 "User Exit" on page 72) and this value is retrieved by the PSNDDAT command(see "SEND Side" on page 75) that is invoked on the target task. The code in this example is the PWATDAT command.

The syntax of the command is:

PWATDAT taskname

The flow of the wait for data function is:

```
                                TARGET
  OST                             OST

 Invokes
 PWATDAT  command
 and specifies the
 target task to
 send the request


 PWATDAT  using
 CNMSMSG sends a  ----->
 request to the
 OST specified

 OST issues a WAIT
 FOR DATA

                                PSNDDAT command is
                                invoked on the
                                OST.  It finds the
                                task global variable
                                set by DSIEX03.


                                CNMSMSG type of
                           <--- DATA is invoked with
                                the value retrieved

 OST wait is
 satisfied --- wake up
 and issue  message
 to the operator
```

```
/********************************************************************/
/*                                                                  */
/*  Other Declarations                                              */
/*                                                                  */
/********************************************************************/
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      */
/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/
CNMSSCAN   DATA(CMDBUF)              /* Scan the input command...    */
           FORMAT('%*S%S')           /* ...skip the command          */
           COUNT(PARMCNT)            /* ...number of parms           */
           P1(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 HLBRC ¬= 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          */
         CNMGETDATA FUNC('GETMSG')  /* Read in the response...      */
               QUEUE(DATAQ)          /* ...on the data queue         */
               DATA(NEWMSG)          /* ...read into NEWMSG variable */
               LENG(256)             /* ...give plenty of room       */
               ORIGIN(GETBLOCK);     /* ...provide a different org blk*/
        /*  REMOVE PROCESS ID FROM THE BUFFER !!!!                  */
        /*  First 8 bytes must be removed                           */
         NEWMSG = SUBSTR(NEWMSG,9);
         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           */
```

SEND Side

The following is part of an example for sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example 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 "User Exit" on page 72). This value is retrieved by the PSNDDAT command that is invoked by the PWATDAT command (see "Wait for Data" on page 73) on the target task. This command processor is executed when the PSNDDAT command is entered.

```
/*********************************************************************/
/*                                                                   */
/*  Other Declarations                                               */
/*                                                                   */
/*********************************************************************/
DCL TIME CHAR(256) VAR;           /* Time the last command was
                                     ...entered                      */
DCL MYOPID CHAR(8) VAR;           /* Operator ID that we are running
                                     ...under                        */
/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/

CNMINFOC                          /* Determine my opid...            */
    ITEM('OPID')                  /* ...variable is opid             */
    DATA(MYOPID)                  /* ...put result here              */
    LENG(8);                      /* ...truncate after 8 bytes       */

IF MYOPID=ORIGIN->ORIG_BLOCK.ORIG_TASK THEN /* Command issued ...
                                     ...directly or target task
                                     ...was same as operators task */
  CNMSENDMSG                      /* Not allowed...                  */
       DATA('Invalid syntax')     /*...text of message               */
       MSGTYPE('MSG')             /* ...message is a single line     */
       DESTTYPE('OPER');          /* ...to a operator                */
ELSE
  DO;
    CNMVARPOOL                    /* Retrieve last time variable     */
       FUNC(GET)                  /* ...read in the value            */
       NAME('LAST_COMMAND_TIME')     /* ...of the variable           */
       POOL('TGLOBAL')            /* ...in the task global pool      */
       DATA(TIME)                 /* ...into time                    */
       LENG(256);                 /* ...truncate at 256              */
  IF (HLBRC=CNM_GOOD) THEN       /* Variable set?                */
    CNMSENDMSG                   /* Yes, continue...             */
     DATA(ORIGIN->ORIG_PROCESS || /* ...must put the process id in*/
         'Last command entered at : "'|| /* ...text of message   */
          TIME||'"')             /* ... more text                */
     MSGTYPE('DATA')             /* ...message is data           */
     DESTTYPE('TASK')            /* ...to a task                 */
     DEST(ORIGIN->ORIG_TASK);    /* ...that invoked us           */
  ELSE
    CNMSENDMSG                   /* No, inform user...           */
      DATA(ORIGIN->ORIG_PROCESS ||    /* ...must put in process id*/
          'Must install DSIEX03 to set TIME variable OR no '||
          'command entered yet on that task')
      MSGTYPE('DATA')             /* ...message is data          */
      DESTTYPE('TASK')            /* ...to a task                */
      DEST(ORIGIN->ORIG_TASK);    /* ...that invoked us          */
END;
```

Chapter 6. Using KnowledgeTool Programs in NetView

This chapter explains how you can use KnowledgeTool™ Version 2 to provide knowledge applications that interact with NetView.

Detailed guidance for developing knowledge applications can be found in the *KnowledgeTool Application Development Guide* (SH20-9262).

Knowledge Applications in the NetView Environment

KnowledgeTool is a PL/I extension and to NetView, a knowledge application is an HLL command processor. Within the knowledge application, NetView functions can be invoked anywhere normal procedural calls can be made — for example, in the right-hand side of rules, in ON ENTRY blocks, in ON IDLE blocks, and in ON CYCLE blocks. Functions that invoke HLL services can be used on the left-hand side of rules.

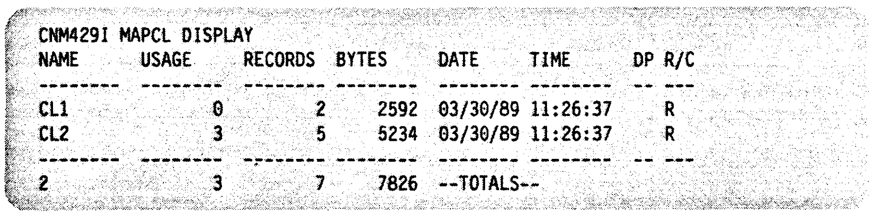
After the HLL program has been written according to KnowledgeTool conventions, it is compiled and link-edited using commands supplied with KnowledgeTool. The KTCOMP command puts the program through several compilation steps that produce an object file ready for linkage. The KTLINK command uses the linkage editor to create the application load module using options specified in the knowledge application profile.

The KTRUN command is not used under NetView. Instead, after you specify the application load module name as the MOD parameter of a CMDMDL statement in the DSICMD member of DSIPARM, you can invoke the knowledge application like any NetView command.

Applications created with KnowledgeTool Version 2 run as command processors under OST, NNT, or Autotasks. Every knowledge application is initialized when it is invoked. If the application uses a loop waiting for an event, for example the arrival of a message, the initialization overhead can be avoided.

Sample Knowledge Application

The following example issues the MAPCL command to monitor usage of loaded command lists. The following figure shows a display resulting from MAPCL.



```
CNM429I MAPCL DISPLAY
NAME      USAGE    RECORDS  BYTES    DATE     TIME     DP R/C
-------- -------- -------- -------- -------- -------- -- ---
CL1             0        2     2592 03/30/89 11:26:37     R
CL2             3        5     5234 03/30/89 11:26:37     R
-------- -------- -------- -------- -------- -------- -- ---
2               3        7     7826 --TOTALS--
```

If a loaded command list has not been used for 10 or 20 minutes, depending on its size, it is dropped. This example is intended to illustrate some capabilities of knowledge applications. Message automation could also perform this simple task, but using a knowledge application would allow you to include more complex deci-

™ KnowledgeTool is a trademark of International Business Machines Corporation sions and actions making full use of the expert system technology available with KnowledgeTool.

The profile for this example can be developed by following the instructions in the NetView section of *KnowledgeTool Application Development Guide* (SH20-9262).

The sample contains the following KnowledgeTool blocks:

ON ENTRY	Run once during the initialization of the rule block.
ON IDLE	Run whenever the conflict set is empty.

```
SAMPKT: PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/*********************************************************************/
/*                                                                   */
/*   Descriptive Name:  Sample KnowledgeTool Program                 */
/*                                                                   */
/*   Function:                                                       */
/*      Monitor usage of loaded command lists and drop               */
/*      those that are unused                                        */
/*         IF SIZE> 2,000 bytes drop if idle more than 10 minutes    */
/*         IF SIZE<= 2,000 bytes drop if idle more than 20 minutes   */
/*                                                                   */
/*   Dependencies:  None                                             */
/*                                                                   */
/*   Restrictions:  None                                             */
/*                                                                   */
/*   Language:  PL/I (KT)                                            */
/*                                                                   */
/*   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.           */
/*      3)  A 40-byte structure which describes the origin of the    */
/*          request that caused execution of this program.           */
/*                                                                   */
/*   Output:  None                                                   */
/*                                                                   */
/*   Return Codes:  returned in Hlbrc                                */
/*      0 = normal exit                                              */
/*                                                                   */
/*   External Module References:  None                               */
/*********************************************************************/
```

```
/*********************************************************************/
/* NetView High-Level Language include files                         */
/*********************************************************************/
%INCLUDE DSIPLI;                   /* Include the HLL macros         */
/*********************************************************************/
/* Parameter declarations                                            */
/*********************************************************************/
DCL HLBPTR   PTR;                  /* Pointer to the HLB             */
DCL CMDBUF   CHAR(*) VARYING;      /* Buffer for the command         */
DCL ORIGBLCK CHAR(40);             /* Area for the Origin Block      */
/*********************************************************************/
/* Class  declarations                                               */
/*********************************************************************/
DCL 1 CLIST   CLASS,
      2 CLNAME CHAR(8) VARYING,    /* Command list name              */
      2 CLBYTE FIXED BIN(31),      /* Command list size in bytes     */
      2 CLCOUNT FIXED BIN(31),     /* Use Count                      */
      2 CLTIME FIXED BIN(31),      /* Time of last element update    */
      2 IDLETIME FIXED BIN(31);    /* Time Command list not in use   */
DCL CL_SEL SELECTOR;               /* CLIST selector                 */
DCL 1 CTIME   CLASS,
      2 CURTIME FIXED BIN(31);     /* Current time in minutes        */
DCL T     SELECTOR;                /* Time  SELECTOR                 */
/*********************************************************************/
/* Other declarations                                                */
/*********************************************************************/
DCL 1 ORIGTEMP LIKE ORIG_BLOCK; /* ORIGBLOCK for read operations     */
DCL (COUNT1,                       /* count for SSCAN service        */
     I,                            /* DO loop control                */
     MAPCOUNT,                     /* Use count for SSCAN service    */
     MAPBYTE,                      /* Command list size in bytes     */
     PREVTIME)                     /* Time of previous MAPCL         */
            FIXED BIN(31);
DCL MSGBUF VARYING CHAR(80);       /* Buffer for incoming messages   */
DCL MAPNAME VARYING CHAR(08);      /* Command list name from MAPCL   */
DCL (SUBSTR,ADDR,TIME,MATCH) BUILTIN;
/*********************************************************************/
/* ON ENTRY Define messages to be trapped                            */
/*********************************************************************/
ON ENTRY BEGIN;
  CALL CNMCMD(HLBPTR,'TRAP MESSAGES CNM429I');
  ALLOCATE CTIME SET (T);
END;
```

```
/*******************************************************************/
/* RULE to cleanup working memory                                  */
/*******************************************************************/
RUL1: WHEN (CL_SEL=>CLIST  & T=>CTIME
       (CL_SEL=>CLTIME ¬=T=>CURTIME)) BEGIN;
  FREE CL_SEL=>CLIST;
ENDRUL1: END;
/*******************************************************************/
/* RULE to drop large Command lists                                */
/*******************************************************************/
RUL2: WHEN (CL_SEL=>CLIST  (CL_SEL=>CLBYTE >  20000,
                           CL_SEL=>IDLETIME>10 )) BEGIN;
  CALL CNMCMD (HLBPTR,'DROPCL '||CL_SEL=>CLNAME);
  FREE CL_SEL=>CLIST;
ENDRUL2: END;
/*******************************************************************/
/* RULE to drop small Command lists                                */
/*******************************************************************/
RUL3: WHEN (CL_SEL=>CLIST  (CL_SEL=>CLBYTE <=  20000,
                           CL_SEL=>IDLETIME>20 )) BEGIN;
  CALL CNMCMD (HLBPTR,'DROPCL '||CL_SEL=>CLNAME);
  FREE CL_SEL=>CLIST;
ENDRUL3: END;
/*******************************************************************/
/* ON IDLE                                                         */
/*******************************************************************/
ON IDLE BEGIN;                      /* wait for incoming message    */
  CALL CNMCMD(HLBPTR,'WAIT 2 MINUTES  FOR MESSAGES');
  IF HLBRC = CNM_TIME_OUT_WAIT THEN CALL CNMCMD(HLBPTR,'MAPCL');
/*******************************************************************/
/* Convert time in CHAR to numeric. Add 24 hours if time wrapped   */
/*******************************************************************/
  T=>CURTIME = SUBSTR(TIME(),3,2)+60*SUBSTR(TIME(),1,2);
  IF (T=>CURTIME < PREVTIME) THEN T=>CURTIME = T=>CURTIME + 24*60;
  PREVTIME = T=>CURTIME;
  HLBRC=0;
  DO I=1 TO 4  WHILE (HLBRC=0);     /* Read past header records     */
  CNMGETDATA FUNC('GETLINE') DATA(MSGBUF) LENG(80)
    ORIGIN(ORIGTEMP) QUEUE(1);
  END;
```

```
/*********************************************************************/
/*    Process to first trailer record                                */
/*********************************************************************/
  DO WHILE ((HLBRC=0)&(SUBSTR(MSGBUF,1,4)¬='----'));
    CNMSSCAN DATA(MSGBUF)         /* Parse MAPCL results             */
      FORMAT('%S%D%*D%D')         /* Skip number of records          */
      COUNT(COUNT1)
      P1(MAPNAME)                 /* Command list  name              */
      P2(MAPCOUNT)                /* Command list use count          */
      P3(MAPBYTE);                /* Command list size               */
    IF MATCH(CL_SEL=>CLIST(CL_SEL=>CLNAME=MAPNAME)) THEN DO;
      IF MAPCOUNT=CL_SEL=>CLCOUNT THEN DO;/* Command list was idle
        CL_SEL=>IDLETIME = CL_SEL=>IDLETIME + T=>CURTIME -
          CL_SEL=>CLTIME;
      END;                        /* END Command list was idle       */
      ELSE DO;                    /* Command list was used           */
        CL_SEL=>IDLETIME = 0;  /* Reset Idle time                    */
        CL_SEL=>CLCOUNT = MAPCOUNT;/* Set new use count              */
      END;
    END;                          /* End match                       */
    ELSE DO;                      /* No match                        */
      ALLOCATE CLIST SET(CL_SEL);/* allocate new Command list        */
      CL_SEL=>CLNAME = MAPNAME;
      CL_SEL=>CLBYTE = MAPBYTE;
      CL_SEL=>CLCOUNT = MAPCOUNT;
      CL_SEL=>IDLETIME = 0;
    END;                          /* End no match                    */
    CL_SEL=>CLTIME = T=>CURTIME;
    CNMGETDATA FUNC('GETLINE')    /* Read next line of response      */
      DATA(MSGBUF) LENG(80)  ORIGIN(ORIGTEMP) QUEUE(1);
  END;                            /* end Do while                    */
  CNMGETDATA FUNC('FLUSHQ')  QUEUE(1); /* Flush trailer records      */
END;                              /* End ON IDLE                     */
END;                              /* End Procedure                   */
```

Developing Knowledge Applications

Information on developing knowledge applications is given in the *KnowledgeTool Application Development Guide.* To use the KnowledgeTool debugger under NetView you must provide the appropriate system extensions.

There are three system extensions (sometimes called knowledge routines) that control input and output when using the KnowledgeTool debugger (in line mode, as required under NetView). These are called STDIN, STDOUT, and STDERR, but they are also known by their internal names as EWCCSIN0, EWCCSOU0, and EWCCMDS0 respectively. The person who manages the NetView environment must change the content of these system extensions, so they direct input, output, and messages to the correct destinations for NetView. The sample system extensions that follow can be compiled by PL/I with a compile SYSLIB for the Netview Maclib. Compiler options of INCLUDE and MACRO should be specified. Unlike NetView applications, extensions should not include DSIEXKT or DSIHSTUB when they are linked by KTLINK.

Example of Standard Input System Extension

This example defines NetView operator terminal as a standard input device, STDIN.

```
*PROCESS MARGINS(2,72);
 /*********************************************************************/
 /*   Descriptive Name: Standard Input System Extension              */
 /*                                                                   */
 /*                                                                   */
 /*   Function:                                                       */
 /*      Defines NetView operator terminal as a standard input        */
 /*      device, STDIN.                                               */
 /*                                                                   */
 /*   Dependencies:  None                                             */
 /*   Restrictions:  None                                             */
 /*                                                                   */
 /*   Language:  PL/I                                                 */
 /*                                                                   */
 /*   Input:  None                                                    */
 /*                                                                   */
 /*   Output:  None                                                   */
 /*                                                                   */
 /*   Return Codes: returned in HLBRC                                 */
 /*      0 = normal exit                                              */
 /*                                                                   */
 /*   External Module References:  None                               */
 /*                                                                   */
 /*********************************************************************/

 %INCLUDE EWCDPLI0;                      /* Frame  definitions       */
 INVOKED_KR(EWCCSIN,IPARM,OPARM);     /* STDIN Knowledge Routine     */
 %INCLUDE DSIPLI;                        /* HLL control blks & macros*/


 DCL IPARM POINTER;                      /* Ptr to buffer pointer    */
 DCL (STG,UNSPEC) BUILTIN;
 DCL OPARM FIXED BIN(31);                /* Return code              */
 DCL IBUF CHAR(512) BASED(IPARM);        /* Input buffer             */
 DCL ISTRING CHAR(511) VARYING;
 DCL 1 IORIGIN LIKE ORIG_BLOCK;          /* Control block for input
                                            services                 */
 DCL NULL_NUM  FIXED BIN(8) INIT(0);
 DCL NULL_CHAR CHAR;                  /* Null string                 */


 DCL HLBPTR  PTR;
 DCL HLBPTR_MAP PTR BASED;
```

```
UNSPEC(NULL_CHAR)=UNSPEC(NULL_NUM);/* Initialize null string       */
INIT_FS(KR);                       /* Initialize enviornment       */
HLBPTR=EWCDPARM_PTR->HLBPTR_MAP   /* Extract HLBPTR for use by     */
   ->HLBPTR_MAP->HLBPTR_MAP;      /*   CNM... commands             */
CALL CNMSMSG(HLBPTR,                /* Display prompt message      */
  'ENTER KNOWLEDGETOOL COMMAND: (GO XXX) ','MSG','OPER','');
CALL CNMCMD(HLBPTR,'WAIT FOR OPINPUT'); /* Wait for operator input */
CALL CNMGETD(HLBPTR,'GETLINE',ISTRING,  /* Get operator input      */
  STG(ISTRING)-2,IORIGIN,2,0);
IF HLBRC = 0 THEN                   /* Test GET return code        */
  DO;                               /* Get Successful              */
    IBUF=ISTRING||NULL_CHAR;       /*Place oper input in buffer    */
    CALL CNMSMSG(HLBPTR,'-->'||ISTRING, /* Echo input              */
     'MSG','OPER','');
    OPARM=0;                        /* Set return code             */
  END;
ELSE
  DO;                               /* Get unsuccessful            */
    IBUF=NULL_CHAR;                /* Set Buffer to null string    */
    OPARM=0;                        /* Set Return code             */
  END;

END EWCCSIN;
```

Example of Standard Output System Extension

This example defines NetView operator terminal as a standard output device, STDOUT.

```
*PROCESS MARGINS(2,72),SOURCE,INSOURCE;
 /*********************************************************************/
 /*   Descriptive Name: Standard Output System Extension             */
 /*                                                                   */
 /*                                                                   */
 /*   Function:                                                       */
 /*      Defines NetView operator terminal as a standard output       */
 /*      device, STDOUT                                               */
 /*                                                                   */
 /*   Dependencies:  None                                             */
 /*   Restrictions:  None                                             */
 /*                                                                   */
 /*   Language:  PL/I                                                 */
 /*                                                                   */
 /*   Input:  None                                                    */
 /*                                                                   */
 /*   Output:  None                                                   */
 /*                                                                   */
 /*   Return Codes: returned in HLBRC                                 */
 /*      0 = normal exit                                              */
 /*                                                                   */
 /*   External Module References: None                                */
 /*                                                                   */
 /*********************************************************************/
 %INCLUDE EWCDPLI0;                        /* Frame Definitions       */
 INVOKED_KR(EWCCSOU,IPARM,OPARM);     /* STDOUT Knowledge Routine    */
 %INCLUDE DSIPLI;                          /* HLL control blks & macros*/

 DCL STR CHAR(256) VARYING;
 DCL IPARM POINTER;                        /* ptr to input buffer     */
 DCL OPARM POINTER;                        /* dummy pointer           */
 DCL IBUF CHAR(256) BASED(IPARM);          /* input buffer            */
 DCL NULL_NUM  FIXED BIN(8) INIT(0);  /* used to init null char       */
 DCL NULL_CHAR CHAR;                  /* null character               */
 DCL (INDEX,MIN,SUBSTR,UNSPEC) BUILTIN;
 DCL HLBPTR  PTR;
 DCL HLBPTR_MAP PTR BASED;

 INIT_FS(KR);                         /* initialize enviornment       */
 UNSPEC(NULL_CHAR)=UNSPEC(NULL_NUM);/* initialize null character      */
 STR=SUBSTR(IBUF,1,MIN(256,                /* extract msg from IBUF   */
   INDEX(IBUF,NULL_CHAR)-1));

 HLBPTR=EWCDPARM_PTR->HLBPTR_MAP    /* extract HLBPTR for use by      */
 ->HLBPTR_MAP->HLBPTR_MAP;          /*    CNMSENDMSG                  */
 CALL CNMSMSG(HLBPTR,STR,'MSG','OPER',' '); /* send msg to operator */

 END EWCCSOU;
```

Example of Standard Error System Extension

This example defines NetView operator terminal as a standard error device, STDERR.

```
*PROCESS MARGINS(2,72);
 /*********************************************************************/
 /*   Descriptive Name: Standard error System Extension               */
 /*                                                                   */
 /*                                                                   */
 /*   Function:                                                       */
 /*      Defines NetView operator terminal as a standard error        */
 /*      device, STDERR                                               */
 /*                                                                   */
 /*   Dependencies:  None                                             */
 /*   Restrictions:  None                                             */
 /*                                                                   */
 /*   Language:  PL/I                                                 */
 /*                                                                   */
 /*   Input:  None                                                    */
 /*                                                                   */
 /*   Output:  None                                                   */
 /*                                                                   */
 /*   Return Codes: returned in HLBRC                                 */
 /*      0 = normal exit                                              */
 /*                                                                   */
 /*   External Module References:  None                               */
 /*                                                                   */
 /*********************************************************************/
 %INCLUDE EWCDPLI0;                       /* frame definitions        */
 INVOKED_KR(EWCCMDS,IPARM,OPARM);     /* STDERR Knowledge routine    */
 %INCLUDE DSIPLI;                         /* HLL control blks & macros*/

 DCL STR CHAR(256) VARYING;
 DCL IPARM POINTER;                       /* ptr to input buffer      */
 DCL OPARM POINTER;                       /* dummy pointer            */
 DCL IBUF CHAR(256) BASED(IPARM);         /* input buffer             */
 DCL NULL_NUM  FIXED BIN(8) INIT(0);  /* used to init null char       */
 DCL NULL_CHAR CHAR;                  /* null character               */
 DCL (INDEX,MIN,SUBSTR,UNSPEC) BUILTIN;
 DCL HLBPTR  PTR;
 DCL HLBPTR_MAP PTR BASED;

 INIT_FS(KR);                         /* initialize enviornment       */
 UNSPEC(NULL_CHAR)=UNSPEC(NULL_NUM);/*initialize null character       */
 STR=SUBSTR(IBUF,1,MIN(256,               /* extract msg from IBUF    */
   INDEX(IBUF,NULL_CHAR)-1));
 HLBPTR=EWCDPARM_PTR->HLBPTR_MAP   /* extract HLBPTR for use by       */
   ->HLBPTR_MAP->HLBPTR_MAP;       /*   CNMSENDMSG                    */
 CALL CNMSMSG(HLBPTR,STR,'MSG','OPER',''); /* send msg to operator    */
 END EWCCMDS;
```

Chapter 7. Compiling, Link-Editing, and Running Your PL/I Program

Once you have a PL/I compiler installed, you can modify the PL/I compile and link-edit JCL for use with NetView. The objective of this chapter is to provide the information necessary to make these modifications.

Several examples of compile and link-edit JCL will be provided in this chapter. These are given as examples only. You are responsible for modifying the compile and link-edit JCL samples that were shipped with the PL/I compiler.

You must have completed the installation steps for HLL as described in the *NetView Installation and Administration Guide* before attempting to execute PL/I programs in the NetView environment.

Compiling

In order to compile PL/I programs using NetView services, it is necessary to modify the compile step in the JCL to reference the NetView macro library(s). You will need to include in the compiled JCL a SYSLIB statement for SYS1.MACLIB. An example of modifications to the compile step JCL is shown below.

```
//COMPILE EXEC PGM=IEL0AA,REGION=1000K,
//       PARM='OBJECT,MACRO,LIST'
                  .
                  .
                  .
//SYSLIB  DD  DSN=SYS1.MACLIB,DISP=SHR
                  .
                  .
                  .
```

Note: When you are compiling PL/I programs, you will receive a warning message IEL0548I. This message should be ignored.

Link-editing

The following rules apply when link-editing PL/I modules:

- All PL/I load modules must be REENTRANT.
- PL/I load modules can reside in 24 or 31 bit storage and can be entered in either addressing mode.
- All PL/I load modules must be link-edited with DSIHSTUB and DSIEXPLI. DSIHSTUB must be the ENTRY point. For KnowledgeTool applications, DSIEXPLI should be replaced by DSIEXKT.

In order to link-edit a PL/I module to run with NetView, you must modify the PL/I link-edit step in the JCL to reference the appropriate NetView Library(s). This will allow you to include DSIHSTUB and DSIEXPLI at link-edit time. Add SYS1.LINKLIB to the list of automatic call libraries already defined by SYSLIB in the PL/I link-edit step of the JCL. An example is shown below:

```
//LKED     EXEC  PGM=IEWL,
//    PARM='XREF,RENT,LET,LIST,AMODE=&AMODE,RMODE=&RMODE',
//    REGION=4096K,COND=(8,LE,COMPILE)
                      .
                      .
                      .
//         DD    DSN=SYS1.PLIBASE,DISP=SHR
//SYSLIB   DD    DSN=SYS1.LINKLIB,DISP=SHR
                      .
                      .
                      .
  INCLUDE        SYSLIB(DSIHSTUB)
  INCLUDE        SYSLIB(DSIEXPLI)
  ORDER          DSIHSTUB
  ENTRY          DSIHSTUB
  MODE           AMODE(31),RMODE(ANY)
  NAME           LMODNAME(R)
```

Note: All HLL modules must be compiled and link-edited with the REENTRANT option. For PL/I, you will have to include the REENTRANT option on the PL/I procedure statement. The resulting object deck(s) must then be link-edited with the RENT option.

Running

A set of run-time libraries will be shipped with the compiler. In order to execute a PL/I program in the NetView environment, you must modify your NetView start up procedure to reference the appropriate PL/I run-time libraries. Refer to the *NetView Installation and Administration Guide* for more information.

HLL command processors require a CMDMDL statement in member DSICMD of the DSIPARM dataset. User exits are loaded at initialization and need to conform to user exit naming conventions. For more information on user exits see Chapter 2 on page 9.

Part 3. Coding Your C Program

Chapter 8. Coding Your C Program - Interfaces and Restrictions

This chapter will provide necessary information for coding HLL command processors and user exits in C. The appropriate interfaces and language dependent restrictions will also be discussed.

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*. *argc* is not used by NetView. *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 one through three point to the EBCDIC representation of the initial parameters (*Hlbptr, Cmdbuf* and *Origblck*) passed to *main* from NetView. The initial parameters must be converted to hex using *sscanf*. The original command line will be passed to the user's program in *Cmdbuf*. Chapter 9 contains a sample template for coding the *main* function and defining and converting the initial parameters. The descriptions of the initial parameters are as follows:

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.

Cmdbuf
A pointer to a varying length character string that contains the command or message that drove this program.

If this program was driven as a user exit (other than DSIEX02A), this string contains the message which drove this exit. If driven as DSIEX02A, *Cmdbuf* will not contain any useful information. The user will have to retrieve the message from the Initial Data Queue (IDATAQ).

Origblck
A pointer to a 40-byte structure which describes the origin of the request that caused execution of this program. *Origblck* is mapped by DSICORIG.

'LL Run-Time Options

HLL run-time options can be specified by declaring and initializing the external variable named HLLOPTS. If the user does not code HLL run-time options, the default HLL run-time options are assumed. The default value for HLLOPTS is zero. The following bits are defined in HLLOPTS:

Bit Position	Field Name	Description
0	HLL_QUEUED_INPUT	Determines if an HLL program will accept QUEUEd input. Refer to the QUEUE command in the *NetView Operation* manual for further detail. 0 = HLL program will NOT accept QUEUEd input 1 = HLL program will accept QUEUEd input
1	HLL_NO_CANCEL	Determines if an HLL program will terminate on CANCEL/RESET. Refer to RESET command in the *NetView Operation* manual for further detail. 0 = Cancellable 1 = Non-cancellable
2-31		RESERVED for internal use. Do not assign any values to these fields.

The following example illustrates how the default HLL run-time options can be overridden in an HLL program written in C. In this case, the user has chosen to make this C program non-cancellable.

```
#pragma variable(HLLOPTS,NORENT)
extern unsigned int HLLOPTS = 0X40000000;
```

The *#pragma variable* preprocessor directive shown above indicates that the variable named HLLOPTS is to be used in a non-reentrant fashion. This does not have any effect on the reentrancy of the HLL program.

C Run-time Options

The run-time options for a C program can be specified using the following preprocessor directive. Refer to the *C/370 User's Guide* for a detailed explanation of each of the options. The run-time options for executing C programs in the NetView environment are shown in the C coding template provided in Chapter 9. C programs must run with the NOSTAE and NOSPIE options when running in the NetView environment. Running with the STAE or SPIE options will cause unpredictable results in cases where error recovery is necessary.

```
#pragma runopts (NOEXECOPS,NOSTAE,NOSPIE,ISASIZE(4K),ISAINC(4K))
```

To achieve optimum performance, it is recommended to run with the REPORT option until accurate ISA and HEAP sizes are determined. Refer to the run-time storage section of the *C/370 User's Guide* for further details.

Parameters Passed to HLL Service Routines

There are four different types of parameters that can be passed to HLL service routines. Each of the parameters described throughout Chapter 12 fall into one of these categories:

Pointer Variables

Integer Variables

Fixed Length Character Strings

Varying Length Character Strings

A discussion of each of these parameter types follows. The intent of this section is to describe how each of these parameter types can be declared, initialized and passed to the HLL service routines. Examples and recommendations for writing HLL programs in C have been provided in this chapter. Note that these examples are not complete. They have been included here to emphasize how the HLL service routine parameters should be declared, initialized and passed. For complete examples of user written HLL programs, see the HLL samples shipped with NetView. (See Appendix D on page 295.)

Pointer Variables

A pointer variable is a 4-byte pointer field containing an address. All HLL service routines require at least one argument of this type, *Hlbptr*. *Hlbptr* is required for all HLL service routine invocations. The value of *Hlbptr* is calculated by NetView and passed to the HLL command processor or user exit. Therefore, it only needs to be defined in C. The user should NEVER assign a value to this variable. This is the only parameter of this type which does not have to be assigned by the user.

Note: The user does not need to specify the *Hlbptr* parameter when coding the HLL service routine invocations in C. *Hlbptr* is inserted for the user before the HLL service routine is actually invoked.

If an HLL service routine is expecting an address in a pointer field, the user is responsible for assigning a value to that pointer field before invoking the HLL service routine. *Hlbptr* is the only exception to this rule. For C, it is advised to use the & (address) operator when passing pointer variables to HLL service routines rather than creating a separate pointer variable for this purpose. This will ensure that the pointer variable has been assigned a value before invoking the HLL service routine.

```
1  #define VARTOVAR   "VARTOVAR"    /* VARTOVAR constant              */
2  Dsihlb *Hlbptr;                  /* HLB pointer MUST BE DEFINED    */
3  Dsivarch *srcptr;                /* Pointer to source buffer       */
4  Dsivarch *dstptr;                /* Pointer to destination buffer */

5  int dstlen;                      /* Length of destination buffer  */

6  Dsivarch srcbuf;                 /* Source buffer                  */
7  Dsivarch dstbuf;                 /* Destination buffer             */

8  srcptr = &srcbuf;                /* Address of source buffer       */
9  dstptr = &dstbuf;                /* Address of destination buffer */

10 dstlen = 255;

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

Figure 3. Using Pointer Variables in C

2

Hlbptr is defined as a pointer variable. The user does not need to assign a value to *Hlbptr* or include it in the *Cnmcpys* invocation. The value of *Hlbptr* is calculated by NetView and inserted in the parameter list for the user during the preprocessor phase of the compilation.

3

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

8

srcptr is assigned the address of the source buffer (*srcbuf*) to be used as a parameter to *Cnmcpys*.

11

srcptr is passed as a parameter to *Cnmcpys*.

Replacing **11** with the following step illustrates the use of the & (address) operator in C. Use of this operator will eliminate the need to define pointer variables and is advisable whenever possible. Note the use of a string constant instead of the VARTOVAR constant.

```
Cnmcpys(&srcbuf,&dstbuf,dstlen,"VARTOVAR"); /* Copy buffer         */
```

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, etc.. Figure 4 illustrates the use of integer variables in the HLL environment.

```
struct  Vchar12 {
    short int size;                   /* Size of buffer              */
    char buffer [13];                 /* Text buffer                 */
            };

1   Dsihlb *Hlbptr                    /* HLB pointer MUST BE DEFINED  */
2   Vchar12 spname;                   /* Subpool name                */
3   char spfunc[9];                   /* Subpool function            */
4   int sptoken;                      /* Subpool token (returned)    */
5   int spleng;                       /* Cell size                   */
6   int sppricnt;                     /* Number of cells in primary  */
7   int spseccnt;                     /* Number of cells in secondary */
8   int spclass;                      /* Class of storage            */

9   spfunc = "ALLOC  ";               /* Function is ALLOCATE        */
10  Cnmvlc(&spname,NOHEXCNV,"POOLNAME"); /* Initialize subpool name  */
11  sptoken  = 0;                     /* Initialize subpool token    */
12  spleng   = 256;                   /* Cell size = 256 bytes       */
13  sppricnt = 3;                     /* Primary count = 3           */
14  spseccnt = 2;                     /* Secondary count = 2         */
15  spclass  = 1;                     /* Class = 31 bit addressable  */

16 Cnmpool(spfunc,&sptoken,&spname,spleng,sppricnt,spseccnt,
                 spclass);            /* Allocate Subpool             */
```

Figure 4. Using Integer Variables in C

4
sptoken is defined as a 4-byte integer (int).

5
spleng is defined as a 4-byte integer (int).

11
sptoken is initialized to zero. A value will be returned in *sptoken* upon successful completion of the *Cnmpool* invocation.

12
spleng is assigned a value of 256 to be used in the call to *Cnmpool*.

16
Cnmpool is invoked using *&sptoken* and *spleng* as parameters. The value of *sptoken* will be 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 parameter fields that return values to the the user's program must specify a pointer to that parameter field when invoking an HLL service routine. Otherwise, the user will 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 the section on varying length character strings.

Fixed Length Character Strings

The majority of the HLL service routines require the user to pass one or more fixed length character strings as arguments. Most of these fixed length character strings, except *adorigin* and *gdorigin*, are eight characters long. These exceptions are discussed below.

C string constants for most of the fixed length character strings have been provided in DSICCONS. DSICCONS is optional and can be tailored to the specific needs of the user. The following steps correlate to the steps in Figure 4 on page 95.

3
spfunc is defined as a 9-byte character array (8 bytes + \0 character).

9
spfunc is assigned a value of "ALLOC   " to be used in the *Cnmpool* invocation.

16
Cnmpool is invoked using the *spfunc* parameter. *spfunc* could have been defined with a preprocessor define statement (see VARTOVAR in Figure 3 step 1), or passed as a string constant.

```
Cnmpool("ALLOC   ",&sptoken,&spname,spleng,sppricnt,spseccnt,
             spclass);          /* Allocate Subpool             */
```

C does not pad character strings with blanks. It is the responsibility of the user to pad character strings with blanks.

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 will ensure 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, the user must define the character array's length to be 1 character greater than the length expected by the HLL service routine. The use of character arrays to represent fixed length character strings should be avoided whenever possible. Any of the alternative methods mentioned above can be used 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.

The only fixed length character fields required for HLL services that are not 8 bytes in length are origin blocks. The mapping structure for an origin block resides in file DSICORIG which is included by DSIC.

There are two types of origin blocks used by the HLL service routines. The first type of origin block (*Origblck*) is a 40-byte structure which must be declared by the user. This is a required initial parameter which was previously described in 'Initial Parameter' section of this chapter. The user is responsible for defining this 40-byte structure but should never need to alter it. Refer to the C coding template in Chapter 9 on page 107 for an example of how to define *Origblck*.

The second type of origin block (*adorigin*, *gdorigin*) is specified by the user. *adorigin* and *gdorigin* must be at least 38 bytes long and must map to the first 38 bytes of the origin block structure (DSICORIG). The user MUST define these origin blocks separately from the origin block which is required as an initial parameter.

The initial parameter origin block (*Origblck*) should NOT be used in place of *adorigin* or *gdorigin*.

Varying Length Character Strings

Several of the HLL service routines require the user 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 to map the internal representation of a varying length character string as shown below.

LL	TEXT

Where:

LL	2-byte integer field containing the length of TEXT
TEXT	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 one byte for the end of string character (\0). Dsivarch resides in DSICVARC and is included by DSIC. The structure consists of two parts:

size	A 2-byte (short) integer field which contains the size of the character array represented by *buffer*. The end of string character (\0) is not included in this size but should delimit the character array.
buffer	A 255-byte null-terminated (delimited by the end of string character (\0)) character array.

The user is responsible for creating structures like Dsivarch to represent varying length character strings to be passed to HLL service routines. The size portion of the buffer can be used to manipulate buffers that contain hex data. The presence of the null character at the end of the buffer enables the user to use the buffer portion of the structure in other C functions that require the end of string character as a delimiter. The user is 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.

NetView has provided two functions to enable the user to easily 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 that should be used when initializing or altering the buffer portion of a varying length character string. If the user chooses to alter the contents of a varying length character string without using *Cnmvlc* or *Cnmnvlc*, they are responsible for updating the size field and ensuring that the buffer is null-terminated. *Cnmnvlc* and *Cnmvlc* are described in this chapter because they are specific to C support only.

The following steps correlate to the steps in Figure 4 on page 95.

2

spname is defined as a varying length character string.

10

spname is assigned the value POOLNAME using the *Cnmvlc* function.

16

Cnmpool is invoked with the parameter *&spname*. When passing a structure as a parameter, the user 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*.

The user could have defined a pointer variable, assigned the address of the structure to that pointer variable and passed the pointer variable to *Cnmpool*. However, this approach introduces the possibility that the user could forget to assign the pointer variable before passing it to the HLL service routine.

Cnmvlc - Convert string to varying length character string

Cnmvlc enables the user 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 formatting is specified, you are responsible for providing a valid argument list. *Cnmvlc* will also convert the input string to hex if desired. This option may be helpful when coding command processors that invoke CNMCNMI and CNMKIO.

Cnmvlc calculates the length of the converted string (not including 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 *buffer* portion of the structure is null-terminated so that it can be used by other C library routines.

A pointer to the converted varying length character string structure will be returned to the caller on successful completion of this routine. If an error condition occurs, *Cnmvlc* will return a null pointer. Examples illustrating the use of *Cnmvlc* can be found in Chapter 9.

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

Where:

vstring
A varying length character string to receive the converted string. A varying length character string structure consists of a 2-byte *size* field (short integer) followed by a null-terminated *buffer* (character array). Refer to the "Varying Length Character Strings" section of this chapter for further detail.

convert
A two byte integer field containing the value 0 or 1, indicating whether or not the input string should be converted to hex. Constants NOHEXCNV and CNVTOHEX have been defined in DSICCONS.

0 (NOHEXCNV)	Do not convert input string to hex.
1 (CNVTOHEX)	Convert input string to hex.

istring
An input string which follows the conventions specified for the *format-string* parameter of the *printf* function in C. *istring* must be null-terminated and may or may not contain format specifications (designated by %). The user must provide an argument list if formatting is desired. Refer to the *printf* library routine in *Common Programming Interface - C Reference* for further detail.

Usage Notes:

If the user specifies conversion to hex, all of the characters in the input string must represent valid hex data. *Cnmvlc* will return a void pointer if it encounters a character which cannot be converted to hex.

A null pointer will also be returned if the user has specified an invalid value for *convert* or if the format specifications cannot be resolved.

Some of the HLL services routines (*Cnmcnmi* and *Cnmkio* in particular) require the user to move hex 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 hex zeros) will be interspersed throughout the hex data stream. NetView has provided *Cnmnvlc* (also discussed in this chapter) to alleviate this problem.

Cnmnvlc - Convert string to varying length character string using length

Cnmnvlc enables the user 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 hex data into varying length character strings and is particularly useful when coding HLL command processors which invoke *Cnmcnmi* or *Cnmkio*.

The function provided by *Cnmnvlc* is very similar to that of *Cnmvlc* except that the user is required to pass a length field. *Cnmnvlc* uses the length field to determine how many characters to copy from the input string. Also, *Cnmnvlc* will not accept format specifications or an argument list. The user can choose to convert an input string to hex if desired. If hex conversion is specified, the length parameter should represent the length of the input string before it is converted.

Once the copy function has completed, *Cnmnvlc* stores the value of the length parameter in the structure. The null terminator is actually copied into the *buffer* portion of the structure even though it is not included in *size*.

A pointer to the converted varying length character string structure will be returned to the caller on successful completion of this routine. If an error condition occurs, *Cnmnvlc* will return a null pointer. Examples illustrating the use of *Cnmnvlc* can be found in Chapter 9.

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

Where:

vstring
A varying length character string to receive the converted string. A varying length character string structure consists of a 2-byte *size* field (short integer) followed by a null-terminated *buffer* (character array). Refer to the "Varying Length Character Strings" section of this chapter for further detail.

convert
A two byte integer field containing the value 0 or 1, indicating whether or not the input string should be converted to hex. Constants NOHEXCNV and CNVTOHEX have been defined in DSICCONS.

0 (NOHEXCNV)	Do not convert input string to hex.
1 (CNVTOHEX)	Convert input string to hex.

length
A four byte integer field specifying the number of bytes to copy from the input string. If hex conversion is required, *length* is the length of the input string before it is converted. *length* must be greater than zero.

istring
An input string. If hex conversion is required, all of the characters in the input string must represent valid hex data.

Usage Notes:

If the user specifies conversion to hex, all of the characters in the input string must represent valid hex data. *Cnmnvlc* will return a void pointer if it encounters a character which cannot be converted to hex.

A null pointer will also be returned if the user has specified an invalid value for *convert* or *length*.

Control Blocks and Include Files

There are a number of control blocks and include files that are required for execution of an HLL program (written in C) in the NetView environment. DSIC is the main file that includes the rest of the files and is necessary to compile HLL programs written in C. Optional include files have been provided to assist the user in coding and maintaining HLL programs. DSIC, DSICCNM and DSICCONS may be tailored to the user's needs.

Note: Tailoring files can lead to better performance in many cases. This is especially helpful in performance sensitive environments such as the user exit environment.

The following list of control blocks and include files reside in Appendix C on page 279.

DSIC	(Required) Must be included by all HLL programs written in C. DSIC includes all of the external HLL control blocks and include files needed to compile and run C programs in the NetView environment. Refer to the C coding template in Chapter 9 for usage.
DSICCALL	(Required) C definitions for HLL service routines.
DSICCNM	(Optional) Defines HLL return code constants for C.
DSICCONS	(Optional) Defines constants that are helpful when coding High-Level Language programs in C.
DSICHLB	(Required) C mapping of internal control block DSIHLB.
DSICORIG	(Required) C mapping of the origin block of the request that caused the execution of the program currently running.
DSICVARC	(Required) C mapping of a varying length character string.

C I/O Considerations

C provides several input and output routines that allow the user to transmit data between main storage and auxiliary storage of a computer. C programs utilizing such file I/O capabilities will 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. The user can specify the file name or a data definition name (*ddname*) when opening a file using *fopen*. If a *ddname* is specified, the user must ensure that the appropriate data set has been allocated before attempting the open. Allocation can be performed under TSO or by using the NetView ALLOCATE command described in *NetView Operation*. NetView also provides a FREE command to deallocate a data set.

If the data set is allocated from TSO, the user must also add a corresponding data definition (DD) statement to the NetView start up 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:

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

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

The following example illustrates the use of file I/O in an HLL program written in C. The check for a NULL pointer has been added to protect against a failure in *fopen*. This check is recommended when opening a file for I/O.

```
FILE     *Outfd;                     /* Define file                   */
          .
          .
          .
/*********************************************************************/
/* Check for error opening file for I/O.  If fopen error occurred,  */
/* issue an error message end 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             */
```

If the user chooses to write to a common output file from two or more C programs, access to the common file must be coordinated by the programs. This can be accomplished using NetView's CNMLK service routine if desired. If access is not coordinated, the user may experience a system ABEND 213.

Special care should be taken when attempting to share open files between two or more HLL programs. Sharing of open files must be coordinated between the sharing programs. 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 will 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 12.

Refer to *Common Programming Interface C Reference* and *C/370 User's Guide* for a more detailed discussion on files and C I/O.

Considerations for HLL Command Processors

It is necessary to code a CMDMDL statement in DSICMD for each HLL command processor that you have written. CMDMDL TYPE will be dependent on the functions that your command processor performs. Keep in mind that some of the HLL services are only useful when executed under a Data Services Task (DST). There is no support for HLL command processors running as immediate commands (TYPE=I). The CMDMDL statement is described in *NetView Administration Reference*.

C Run-time Considerations

Most of the 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 which could normally be handled by coding a *signal* function in your program. However, since C programs must run with the NOSTAE and NOSPIE options when running in the NetView environment, the operating system is unable to generate such interrupts. While debugging a C program in the NetView environment, it is allowable to run with the STAE and SPIE options until the run-time problems have been resolved. Run-time errors that are not detected by the operating system will cause a diagnostic message to be written to *stderr*. See *Common Programming Interface C Reference* for more detail on error handling.

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. This field should be checked after each HLL service routine invocation. It is recommended that this field be utilized when passing return codes between HLL programs.

Note that 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 the user. The completion code can only be checked by evaluating the return code field (*Hlbrc*) in the HLB.

For a complete list of HLL API return codes, see DSICCNM in Appendix C. Refer to Chapter 12 for a list of return codes that apply to each HLL service routine.

In C, normal termination can be achieved by assigning a value to *Hlbrc* and issuing a *return()* statement as shown here.

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

Restrictions for HLL Programs Written in C

The following restrictions apply when using C in a NetView environment:

The following C functions are not supported in the NetView environment:

C function	Recommended HLL alternative
system	Cnmcmd

The following functions cannot be used without re-directing *stdin, stdout,* and *stderr*:

C function	Recommended HLL alternative
getchar	Cnmgetd
getenv	Cnminfi, Cnminfc, Origblck, Cnmvars
gets	Cnmgetd
printf	Cnmsmsg
putchar	Cnmsmsg
puts	Cnmsmsg
scanf	Use Cnmgetd to fetch data and sscanf

	to parse data
perror ferror	Check return code and use Cnmsmsg to put out message

Special Considerations:

1. The C signal function will not work for errors detected by the operating system.
2. *Cnmsmsg* cannot be used to display wide character strings. If you need to process wide character strings, you will have to redirect *stdout* and use a C function. (*printf*)
3. When passing return codes between HLL command procedures it is recommended that you use *Hlbrc* in the HLB control block (see Appendix C). Using EXIT and RETURN with a return code to pass return codes between HLL command procedures does not work; however, RETURN and EXIT can be used to pass return codes between your main C program and any procedures it may call.

Chapter 9. C High-Level Language Services

This chapter is an example-oriented discussion of commands and services provided by NetView in support of C. The complete syntax and usage of each command and service routine can be found in Chapter 12 on page 173.

C Sample Template

The following is a coding template sample to be used when coding HLL programs in C. This template can be used, with your enhancements, to utilize NetView functions and commands. Further examples in this chapter should be used in conjunction with this template. To see a fully functional sample of a NetView C command procedure refer to Appendix D on page 295.

```
/********************************************************************/
/*                                                                  */
/*   (C) COPYRIGHT IBM CORP. 1989                                   */
/*                                                                  */
/*   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 a user 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.          */
/*                                                                  */
/*   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                            */
/*      (A log of the changes made to this module for)                 */
/*      (future reference can be kept.              )                  */
/***********************************************************************/
#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              */

/***********************************************************************/
/* External data definitions                                           */
/***********************************************************************/
Dsihlb   *Hlbptr;                   /* Pointer to the HLB              */
Dsivarch *Cmdbuf;                   /* Pointer to command buffer       */
Dsiorig  *Origblck;                 /* Pointer to Origin block         */

main(int argc, char *argv??(??))
{
  /*********************************************************************/
  /* Internal data definitions                                         */
  /*********************************************************************/

  /*********************************************************************/
  /* Convert parameter pointers from character to hex addresses        */
  /*********************************************************************/
  sscanf(argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

  /*********************************************************************/
  /* Initialization                                                    */
  /*********************************************************************/

  /*********************************************************************/
  /* Execution                                                         */
  /*********************************************************************/
  Hlbptr->Hlbrc = CNM_GOOD;       /* Successful completion             */
}
```

Varying Length Character Strings

In Chapter 8 the use of varying length character strings with HLL command procedures was discussed. *Cnmvlc* and *Cnmnvlc* have been provided for the C user for convenience when dealing with varying length character strings. *Cnmvlc* is especially useful when copying NULL-terminated text into a varying length character string, and building RU's for the CNMI. *Cnmnvlc* is especially useful when dealing with data that has NULLs in it, and data that is not NULL-terminated.

In several of the samples and examples we have defined our varying length character strings as type *Dsivarch*. This is only for convenience. It is actually more efficient to use a varying length character string with a buffer size closer to that of your own data. The following examples show how to use *Cnmvlc*, *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.

The following example copies "Hello World" into a varying length character string for use with *Cnmsmsg*. *msg* is defined as a varying length character string.

```
                                   /* put message in varying length */
                                   /* character string...           */
Cnmvlc(&msg,                       /* ...varying len char strng     */
       NOHEXCNV,                   /* ...do not convert to hex      */
       "Hello World");             /* ...message                    */
                                   /* display 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 that data is being copied into, and therefore can be imbedded directly into calls to HLL service routines. The following example shows a call to *Cnmvlc* imbedded in a call to the *Cnmsmsg* service routine.

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

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

```
                                    /* put RU in varying length      */
                                    /* character string...           */
Cnmvlc(&ru,                         /* ...varying length char strng  */
       CNVTOHEX,                    /* ...convert to hex             */
       "810810000000000E41038D000001110000068090028lF108D7E4D5C1D4C500");
                                                        /* ...the RU */

                                    /* Send RU over the CNMI...      */
Cnmcnmi(SENDRPLY,                   /* ...expect a reply             */
        &ru,                        /* ...RU built above             */
        puname.buffer,              /* ...to the PU name specified   */
        180);                       /* ...timeout after 3 minutes    */
```

The following is an example of using a format string with *Cnmvlc*. The following example copies "Day 1 of five" 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".

```
                                    /* put message in varying length */
                                    /* character string...           */
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          */
                                    /* display message...            */
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.

The following example copies an RU returned by the CNMI 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 message is defined as a varying length character string.

Note: The RU in the following example may contain null data. *Cnmnvlc* copies for the length specified regardless of the presence of NULLS.

```
                                    /* 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              */
                                    /* copy varying length character */
                                    /* string to another varying    */
                                    /* length character string...   */
Cnmnvlc(&msg,                       /* ...dest varying len char strng*/
        NOHEXCNV,                   /* ...do not convert to hex     */
        data.size,                  /* ...numbers of chars to move  */
        data.buffer);               /* ...RU to copy                */
```

Defining Varying Length Character Strings

Sometimes it will be necessary to create varying length character strings other than *Dsivarch*.

The following example copies "Hello World" into a user-defined varying length character string.

```
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*/

                                    /* copy Hello World into user   */
Cnmvlc(&msg,noconvert,"Hello World"); /* defined varying length     */
                                    /* character string             */
Cnmsmsg(&msg,MSG,OPER,NULLCHAR);    /* send message to operator     */
}
```

Sometimes it is necessary to pass a command buffer larger than 256 characters to a C program. In these cases you must change the declaration for *Cmdbuf* provided in the C template. The following example excepts a command buffer larger than 256 characters and displays it on the MVS operator console.

```
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 paramater list from...*/
{                                    /* ...Netview                    */
sscanf(argv??(2??),"%x",&Cmdbuf);    /* Convert Cmdbuf ptr to hex     */
Cnmsmsg(Cmdbuf,MSG,SYSOP,NULLCHAR);  /* display message               */
}
```

Data Queue Management

NetView utilizes several data and message queues to work in conjunction with HLL service routines. Information retrieved from these queues, via the GETDATA function, can be manipulated to enhance your network manageability. The following five queues are defined for data and message management.

TRAPQ	Queue 1	This queue enables the user to access messages placed on it after being trapped as a result of an issuance of the TRAP command for messages.
OPERQ	Queue 2	This queue enables the user to access operator input, entered via the GO or QUEUE command..
DATAQ	Queue 3	This queue enables the user to access DATA type messages placed on it via the send message HLL service routine (*Cnmsmsg*).
IDATAQ	Queue 4	The initial data queue enables the user to access the message that invoked the HLL command by the message automation table or the message which drove DSIEX02A.
CNMIQ	Queue 5	This queue enables the user to access CNMI solicited data which was solicited via an issuance of the HLL CNMI service routine (CNMCNMI).

Examples of how the above queues are used with HLL command procedures follow.

Sending Information

The following is an example of sending messages.

```
/*********************************************************************/
/* Internal data definitions                                         */
/*********************************************************************/

 Dsivarch msgbuf;                 /* var len char strng for messages */

/*********************************************************************/
/*              Send a multiline message to user.                    */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"LINE 1 OF 3");
   Cnmsmsg(&msgbuf,MSG_C,OPER,NULLCHAR);
   Cnmvlc(&msgbuf,0,"Line 2 of 3");
   Cnmsmsg(&msgbuf,MSG_D,OPER,NULLCHAR);
   Cnmvlc(&msgbuf,0,"Line 3 of 3");
   Cnmsmsg(&msgbuf,MSG_F,OPER,NULLCHAR);

/*********************************************************************/
/*              Send a multiline message to a task.                  */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"Line 1 of 3");
   Cnmsmsg(&msgbuf,MSG_C,TASK,"OPER1   ");
   Cnmvlc(&msgbuf,0,"Line 2 of 3");
   Cnmsmsg(&msgbuf,MSG_D,TASK,"OPER1   ");
   Cnmvlc(&msgbuf,0,"LINE 3 OF 3");
   Cnmsmsg(&msgbuf,MSG_F,TASK,"OPER1   ");

/*********************************************************************/
/*          Send a message to the system console (only 1-liners).    */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"HELLO SYSOP");
   Cnmsmsg(&msgbuf,MSG,SYSOP,NULLCHAR);
```

```
/*********************************************************************/
/*          Send a message to the authorized receiver.               */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"HELLO AUTHRCV");
   Cnmsmsg(&msgbuf,MSG,AUTHRCV,NULLCHAR);

/*********************************************************************/
/*          Send a message to the log.                               */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"This should only be in the log");
   Cnmsmsg(&msgbuf,MSG,NETVLOG,NULLCHAR);

/*********************************************************************/
/*          Send to the sequential log                               */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"Test message");
   Cnmsmsg(&msgbuf,MSG,SEQLOG,"SQLOGTSK");

/*********************************************************************/
/*          Send to a group                                          */
/*********************************************************************/
   Cnmvlc(&msgbuf,0,"Hello group");
   Cnmsmsg(&msgbuf,MSG,OPCLASS,"+GROUP1 ");
```

Synchronous Commands

The following is a simple example of an HLL command processor invoking another command. The command could be another HLL command, a VTAM command, or a NetView command.

```
 /********************************************************************/
 /* Internal data definitions                                        */
 /********************************************************************/

 Dsivarch command,              /* varying len char strng for cmds */
          message;              /* varying len char strng for msgs */

 /********************************************************************/
 /*                                                                  */
 /*  Execution                                                       */
 /*                                                                  */
 /********************************************************************/

/* copy vtam command d net,appls into varying length character     */
/* string using cnmvlc                                              */

Cnmvlc(&command,                /* varying length character string */
       0,                       /* do not convert to hex           */
       "D NET,APPLS");          /* command to copy into varying    */
                                /* length character string         */

/* issue the command                                                */

Cnmcmd(&command);

if (Hlbptr->Hlbrc != CNM_GOOD)   /* always check rc from Cnmcmd    */
  {                             /* if bad return code from Cnmcmd  */
                                /* place message in varying length */
  Cnmvlc(&message,              /* character string...             */
         0,                     /* ...do not convert to hex        */
         "Command not scheduled successfully"); /* ...message      */
                                /* issue message...                */
  Cnmsmsg(&message,             /* ...message text                 */
          MSG,                  /* ...type is message              */
          OPER,                 /* ...send to invoking operator    */
          NULLCHAR);            /* ...not used                     */
  }
```

Sending Commands

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

This process can be used to execute commands under data services tasks (DST), other operator station tasks (OST), or the primary POI task (PPT).

```
/*******************************************************************/
/* 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                    */
  }
else
  {                            /* inform user task not active    */
  if (Hlbptr->Hlbrc == CNM_TASK_INACTIVE)
    {
                               /* copy message into varying      */
                               /* length character string...     */
    Cnmvlc(&msgbuf,            /* ...varying length string       */
          0,                   /* ...do not convert to hex       */
          "OPER1 Not Active");  /* ...message                    */

                                /* issue message...              */
    Cnmsmsg(&msgbuf,            /* ...text of message            */
           MSG,                 /* ...this is a message          */
           OPER,                /* ...to the operator            */
           NULLCHAR);           /* ...not used                   */
    }
  else
    {                          /* inform user bad rc             */
                               /* copy message into varying      */
                               /* length character string...     */
    Cnmvlc(&msgbuf,            /* ...varying length string       */
          0,                   /* ...do not convert to hex       */
          "Unexpected rc from Cnmsmsg");      /* ...message      */

                                /* issue the message...          */
    Cnmsmsg(&msgbuf,            /* ...text of message            */
           MSG,                 /* ...this is a message          */
           OPER,                /* ...to the operator            */
           NULLCHAR);           /* ...not used                   */
    }
 }
Hlbptr->Hlbrc = CNM_GOOD;       /* clear rc                      */
```

Waiting and Trapping

The following is an example of how to issue a command, trap the output of the command, and respond differently depending on the output that is encountered. It will activate the given LU, and issue an appropriate message.

The syntax that it checks for is:

CACTLU luname

Where luname is the name of the LU to be activated.

```
/*******************************************************************/
/* Internal data definitions                                       */
/*******************************************************************/
Dsiorig 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                          */
/*******************************************************************/

token = strtok((char *) &(Cmdbuf->buffer)," ");  /* parse command */
token = strtok(NULL," ");     /* buffer for LU name             */
strcpy(nodename,token);

if (strlen(token) > 8)        /* node name invalid              */
  token = NULL;


if (token != NULL)           /* if nodename specified          */
  {                          /* copy command into varying length */
                             /* character string...            */
  Cnmvlc(&command,           /* ...varying length string       */
        0,                   /* ...do not convert to hex       */
        "TRAP AND SUPPRESS ONLY MESSAGES IST*");  /* ...command  */


  Cnmcmd(&command);          /* issue command to trap vtam messages*/
  /*****************************************************************/
  /* build vtam command to activate node                           */
  /*****************************************************************/

                            /* copy command into 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     */

                            /* put command to wait for 5 sec in   */
                            /* varying length character string... */
  Cnmvlc(&command,          /* ...varying length string           */
         0,                 /* ...do not convert to hex           */
         "WAIT 10 SECONDS FOR MESSAGES"); /* ...command           */

  Cnmcmd(&command);         /* issue command to wait for 5 secs   */

                            /* get first trapped message...       */
  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               */

/*****************************************************************/
/*                                                                 */
/* Loop through messages until IST093I is found or until no more */
/*    messages are left                                            */
/*                                                                 */
/*****************************************************************/
                            /* put wait continue command in       */
                            /* varying length character string... */
  Cnmvlc(&command,          /* ...varying length string           */
         0,                 /* ...do not convert to hex           */
         "WAIT CONTINUE");           /* ...command                */

  while((strncmp(origptr.Orig_process,"IST093I",7) != 0) &&
        (Hlbptr->Hlbrc == CNM_GOOD))
    {
    Cnmcmd(&command);       /* issue wait continue                */
                            /* get next trapped message...        */
    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               */

    }
    if (Hlbptr->Hlbrc == CNM_GOOD)    /* 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       */

                                /* 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       */
                                /* ...message                        */
                                /* issue message...                  */
      Cnmsmsg(&msgbuf,          /* ...message to issue               */
              MSG,              /* ...type is message                */
              OPER,             /* ...issue to operator              */
              NULLCHAR);        /* ...unused                         */
    }
  }
  else                          /* nodename not specified            */
    {                           /* inform user he needs more args    */

                                /* copy message into varying length  */
                                /* character string...               */
    Cnmvlc(&msgbuf,             /* ...varying length string          */
           0,                   /* ...do not convert to hex          */
           "ERROR - INVALID NODENAME");      /* ...message           */
                                /* ...message                        */
                                /* issue the message...              */
    Cnmsmsg(&msgbuf,            /* ...message to issue               */
            MSG,                /* ...type is message                */
            OPER,               /* ...issue to operator              */
            NULLCHAR);          /* ...unused                         */
    }
```

Retrieving Information

The following gives an example of how an HLL command processor or user exit routine can retrieve information from NetView. Assembler language command processors and user exit routines need DSECTs to access information about NetView. HLL command processors and user exit routines can access some of this information as shown below. Many variables are available. Please refer to the command and service routine reference on *Cnminfc* and *Cnminfi* for an exhaustive list of the values supported ("CNMINFC (CNMINFOC) – Query NetView Character Information" on page 209 and "CNMINFI (CNMINFOI) – Query NetView Integer Information" on page 211).

```
/*********************************************************************/
/* 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",             /* ...specify the variable        */
       &cdata,                  /* ...result goes here            */
       18);                     /* ...at most eight bytes         */

 cdata.buffer??(cdata.size??) = '\0'; /* terminate data with null */
```

```
/******************************************************************/
/* build message to display results                               */
/******************************************************************/

                              /* put message in varying length   */
                              /* character string...             */
Cnmvlc(&msgbuf,               /* varying length string...        */
       0,                     /* do not convert to hex...        */
       "DATE/TIME IS: %s",cdata.buffer);  /* ...message          */

                              /* display results...              */
Cnmsmsg(&msgbuf,              /* ...text of message              */
        MSG,                  /* ...is a massage                 */
        OPER,                 /* ...to invoking operator         */
        NULLCHAR);            /* ...not used                     */

                              /* retrieve the number of colors   */
                              /* that the terminal supports...   */
Cnminfi("COLORS  ",           /* ...specify the variable         */
        &idata);              /* ...result goes here             */

/******************************************************************/
/* build message to display results                               */
/******************************************************************/

                              /* put message in varying length   */
                              /* character string...             */
Cnmvlc(&msgbuf,               /* ...varying length string        */
       0,                     /* ...do not convert to hex        */
       "NUMBER OF COLORS SUPPORTED ARE: %d",idata); /* ...message*/

                              /* display results...              */
Cnmsmsg(&msgbuf,              /* ...message text                 */
        MSG,                  /* ...is a message                 */
        OPER,                 /* ...to the invoking operator     */
        NULLCHAR);            /* ...not used                     */
```

Command List Variable Access

The following example illustrates the capability to update common global variables. This example simply declares, initializes, and alters a variable named "GVARIABLE".

Task globals are updated and read the same way. The only difference is the pool name that is specified.

```
/*******************************************************************/
/* Internal data definitions                                       */
/*******************************************************************/
Dsivarch datain,              /* store data returned by Cnmvars  */
         variable;            /* store variable name for Cnmvars */
int length = 24,          /* max len of data returned by Cnmvars */
    x;                        /* used to increment value returned*/
                              /* ...by cnmvars                   */

/*******************************************************************/
/*                                                                 */
/*  Execution                                                      */
/*                                                                 */
/*******************************************************************/
/*******************************************************************/
/* Initialize the variable                                         */
/*******************************************************************/
                              /* copy initial value into varying */
                              /* length character string...      */
Cnmvlc(&datain,               /* ...varying length string        */
       0,                     /* ...do not convert to hex        */
       "Initial value");      /* ...initial value                */

                              /* copy variable name into varying */
                              /* length character string...      */
Cnmvlc(&variable,             /* ...varying length string        */
       0,                     /* ...do not convert to hex        */
       "GVARIABLE");          /* ...variable name                */

                              /* put an initial value in variable*/
Cnmvars(PUT,                  /* ...function is write            */
        &datain,              /* ...data is here                 */
        length,               /* ...length not used              */
        &variable,            /* ...variable name                */
        "CGLOBAL ");          /* ...variable pool is cglobal     */
```

```
/*******************************************************************/
/* Find out the value of the variable                              */
/*******************************************************************/
                               /* read the global variable...     */
Cnmvars(GET,                   /* ...function is read             */
        &datain,               /* ...data goes here               */
        length,                /* ...truncate after 24 bytes      */
        &variable,             /* ...variable name                */
        CGLOBAL);              /* ...variable pool is cglobal     */

                               /* show operator value...          */
Cnmsmsg(&datain,               /* ...data obtained from varpool   */
        MSG,                   /* ...type is message              */
        OPER,                  /* ...send message to operator     */
        NULLCHAR);             /* ...not used                     */

                               /* put new in varying length       */
Cnmvlc(&datain,                /* character string...             */
       0,                      /* ...do not convert to hex        */
       "New value");           /* ...new value                    */

/*******************************************************************/
/* set the global variable                                         */
/*******************************************************************/
                               /* update the global variable...   */
Cnmvars(PUT,                   /* ...function is write            */
        &datain,               /* ...data is here                 */
        length,                /* ...datalen is not used          */
        &variable,             /* ...variable name                */
        CGLOBAL);              /* ...variable pool is CGLOBAL     */

/*******************************************************************/
/* verify variable has been changed                                */
/*******************************************************************/
                               /* read the global variable ...    */
Cnmvars(GET,                   /* ...function is read             */
        &datain,               /* ...data goes here               */
        length,                /* ...truncate after 24 bytes      */
        &variable,             /* ...variable name                */
        CGLOBAL);              /* ...variable pool is CGLOBAL     */
                               /* show value to operator ...      */
Cnmsmsg(&datain,               /* ...data to be displayed         */
        MSG,                   /* ...type is message              */
        OPER,                  /* ...send message to operator     */
        NULLCHAR);             /* ...not used                     */
```

Using Locks

The previous example illustrated the capability to update common global variables, but it did not protect the updating of the variable named "GVARIABLE" by using a lock. This example has been modified to obtain a lock before attempting the update.

The lock name can be the same as the global variable, or it can be different. The need for protecting the updating needs to be assessed on a case-by-case basis.

If you decide that it is important to synchronize the updating of a variable, you can use the lock method shown below or you may wish to run all the updates on a given task. Since only one process can occur on a task at a time, the updates will be serialized. Note that this could be any task, including the PPT.

```
/********************************************************************/
/* Internal data definitions                                        */
/********************************************************************/
Dsivarch datain,               /* store data returned by cnmvars  */
         variable,             /* store variable name for Cnmvars */
         msg;                  /* var len char strng for messages */
int length = 24,               /* max len of data from Cnmvars    */
    x;                         /* used to increment value returned*/
                               /* ...by Cnmvars                   */

/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/
/********************************************************************/
/* Initialize the variable                                          */
/********************************************************************/
                               /* copy initial value into varying */
                               /* length character string...      */
Cnmvlc(&datain,                /* ...varying length string        */
       0,                      /* ...do not convert to hex        */
       "Initial value");       /* ...initial value                */

                               /* copy variable name into varying */
                               /* length character string...      */
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);

                              /* put an initial value in variable*/
Cnmvars(PUT,                  /* ...function is write           */
        &datain,              /* ...data is here                */
        length,               /* ...length not used             */
        &variable,            /* ...variable name               */
        "CGLOBAL ");          /* ...variable pool is cglobal    */

/*****************************************************************/
/* Find out the value of the variable                           */
/*****************************************************************/
                              /* read the global variable...    */
Cnmvars(GET,                  /* ...function is read            */
        &datain,              /* ...data goes here              */
        length,               /* ...truncate after 24 bytes     */
        &variable,            /* ...variable name               */
        CGLOBAL);             /* ...variable pool is cglobal    */


                              /* show operator value...         */
Cnmsmsg(&datain,              /* ...data obtained from varpool  */
        MSG,                  /* ...type is message             */
        OPER,                 /* ...send message to operator    */
        NULLCHAR);            /* ...not used                    */

                              /* put new value in varying length */
Cnmvlc(&datain,               /* character string...            */
       0,                     /* ...do not convert to hex       */
       "New Value");          /* ...new value                   */

/*****************************************************************/
/* set the global variable                                      */
/*****************************************************************/
                              /* update the global variable...  */
Cnmvars(PUT,                  /* ...function is write           */
        &datain,              /* ...data is here                */
        length,               /* ...datalen is not used         */
        &variable,            /* ...variable name               */
        CGLOBAL);             /* ...variable pool is CGLOBAL    */
```

```
/*******************************************************************/
/* verify variable has been changed                                */
/*******************************************************************/
                              /* read the global variable ...    */
Cnmvars(GET,                  /* ...function is read             */
        &datain,              /* ...data goes here               */
        length,               /* ...truncate after 24 bytes      */
        &variable,            /* ...variable name                */
        CGLOBAL);             /* ...variable pool is CGLOBAL     */
                              /* show value to operator ...      */
Cnmsmsg(&datain,              /* ...data to be displayed         */
        MSG,                  /* ...type is message              */
        OPER,                 /* ...send message to operator     */
        NULLCHAR);            /* ...not used                     */

/*******************************************************************/
/* Release the lock to let other tasks update GVARIABLE            */
/*******************************************************************/
Cnmlk(UNLOCK,                 /* obtain the lock...              */
      &variable,              /* ...function is obtain lock      */
      "       ",              /* ...not used                     */
      "       ");             /* ...not used                     */

if (Hlbptr->Hlbrc == 0)
   Cnmsmsg(Cnmvlc(&msg,0,"got rid of lock"),MSG,OPER,NULLCHAR);
```

Operator Input

The following is an example of 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, input is retrieved by the application using the CNMGETD service routine, and the operator can respond using the GO command.

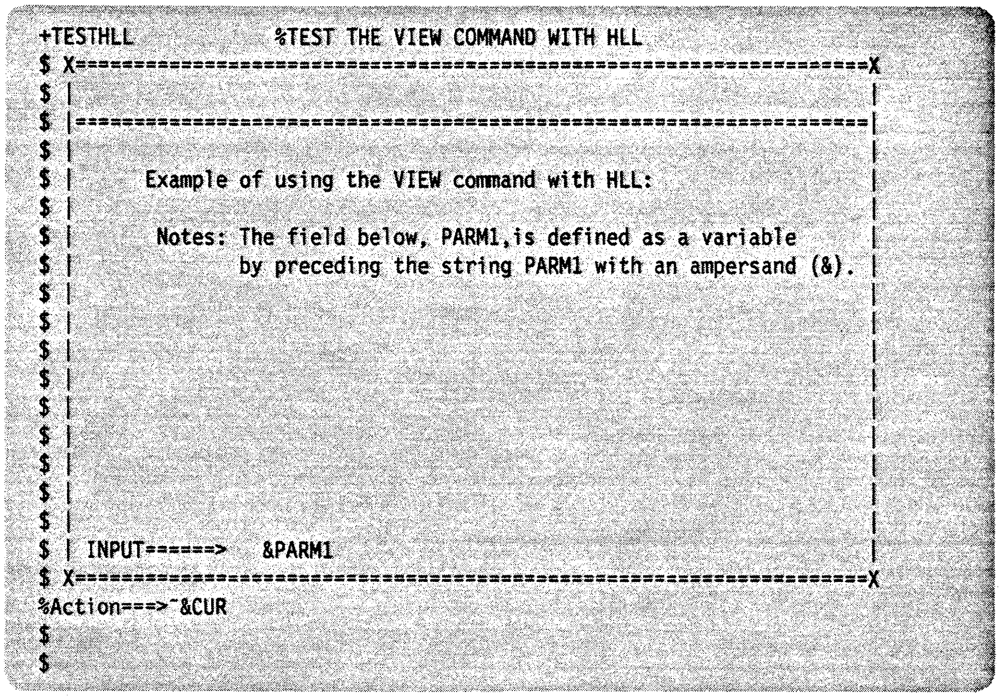
```
/*******************************************************************/
/* Internal data definitions                                       */
/*******************************************************************/
Dsivarch msgbuf,             /* var len char strng for messages */
         command,            /* var len char strng for cmds     */
         inbuf;              /* store data returned by Cnmgetd  */

/*******************************************************************/
/*                                                                 */
/*  Execution                                                      */
/*                                                                 */
/*******************************************************************/
                                   /* copy message into varying */
Cnmvlc(&msgbuf,                    /* length message string...  */
       0,                          /* ... do not convert to hex */
       "Enter operator input data"); /* ... message             */
                                   /* send a message...         */
Cnmsmsg(&msgbuf,                   /* ...text of message        */
        MSG,                       /* ...single line message    */
        OPER,                      /* ...to the invoking oper   */
        NULLCHAR);                 /* ...not used               */
                                   /* copy command to varying   */
Cnmvlc(&command,                   /* length character string...*/
       0,                          /* ...do not convert to hex  */
       "WAIT 30 SECONDS FOR OPINPUT"); /* ...command            */
                                   /* issue command to wait for */
Cnmcmd(&command);                  /* 30 seconds...             */
```

```
                                    /* get first trapped msg...  */
if (Hlbptr->Hlbrc == CNM_OPINPUT_ON_WAIT)
  {                                 /* Operator input supplied...*/
  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    */
                                    /* send a message...         */
  Cnmsmsg(&msgbuf,                  /* ...text of messgage       */
          MSG,                      /* ...single line message    */
          OPER,                     /* ...to the invoking oper   */
          NULLCHAR);                /* ...not used               */
  }
else
  {                                 /* No operator input supplied*/
                                    /* copy message into varying */
  Cnmvlc(&msgbuf,                   /* length character string...*/
         0,                         /* ...do not convert to hex  */
         "NO OPERATOR INPUT SUPPLIED"); /* ... message           */
                                    /* send a message...         */
  Cnmsmsg(&msgbuf,                  /* ...text of messgage       */
          MSG,                      /* ...single line message    */
          OPER,                     /* ...to the invoking oper   */
          NULLCHAR);                /* ...not used               */
  }
```

VIEW Command Processor

The following is an example of using the full-screen VIEW command processor. First it creates the local variable called PARM1, and the variable is initialized. The VIEW command processor is invoked, displaying a full-screen panel. For more information on VIEW see the *NetView Customization Guide*.

The panel that is invoked by the following example appears below:



```
+TESTHLL           %TEST THE VIEW COMMAND WITH HLL
$ X==============================================================X
$ |                                                              |
$ |==============================================================|
$ |                                                              |
$ |      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                                        |
$ X==============================================================X
%Action===>¬&CUR
$
$
```

```
/********************************************************************/
/* 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 parml                */
        LOCAL);              /* ...the pool is local            */
                             /* copy initialization data to     */
Cnmvlc(&data,                /* variable length character strng..*/
       0,                    /* ...do not convert to hex        */
       "the contents of parml go here");  /* ...the data        */

Cnmvars(PUT,                 /* initialize parml...             */
        &data,               /* ...result goes here             */
        48,                  /* ...max length is 48 bytes       */
        &variable,           /* ...name of variable is parml    */
        LOCAL);              /* ...the pool is local            */

/********************************************************************/
/* Issue the view command.  Give the task name as a unique name    */
/* to go on the View stack.                                         */
/********************************************************************/
                             /* copy command to variable length  */
Cnmvlc(&command,             /* character string...             */
       0,                    /* ...do not convert to hex        */
       "VIEW %.8s TESTHLL NOMSG INPUT",Origblck->Orig_task);/* cmd*/

Cnmcmd(&command);            /* issue the command               */
```

Message Processing

The following example lists the message attributes of a message. The invocation must be as a result of an entry in the message automation table, which is documented in the *NetView Administration Reference.* This example will function correctly for both single line messages and multiple line messages.

```
/*******************************************************************/
/* Internal data definitions                                       */
/*******************************************************************/
Dsiorig getblock;              /* orig block for Cnmgetd         */
Dsivarch inbuf,                /* returned data from Cnmgetd     */
         datain,               /* returned data from Cnmgeta     */
         message,              /* var len char strng for messages */
         temp;                 /* temp var len char strng used... */
                               /* ...to build messages           */
int i;                         /* counter                        */

char *attr??(12??) = {         /* array of message attributes... */
"AREAID  ",                    /* ...to be obtained by Cnmgeta   */
"DESC    ",
"JOBNAME ",
"JOBNUM  ",
"MCSFLAG ",
"MSGTYP  ",
"REPLYID ",
"ROUTCDE",
"SESSID  ",
"SMSGID  ",
"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           */
```

```
while((Hlbptr->Hlbrc==CNM_GOOD) || (Hlbptr->Hlbrc==CNM_DATA_TRUNC))
  {                             /* do while no probs...          */
                                /* ...ignoring truncation        */
  for(i=0;i<=11;i++)            /* for 12 possible attributes... */
    {                           /* get the ith attribute...      */
    Cnmgeta(attr??(i??),        /* ...ith member of the array    */
            &datain,            /* ...result goes here           */
            12,                 /* ...at most 12 bytes           */
            IDATAQ);            /* ...on initial data queue      */

                                /* put result of above cnmgeta in */
    Cnmnvlc(&temp,              /* varying length char string... */
            0,                  /* ...do not convert to hex      */
            datain.size,        /* ...size of result             */
            datain.buffer);     /* ...result to be copied        */
                                /* build msg to display results in */
    Cnmvlc(&message,            /* varying length char string... */
           0,                   /* ...do not convert to hex      */
           "%s = %s",attr??(i??),temp.buffer); /* ...message     */
                                /* print that message...         */
    Cnmsmsg(&message,           /* ...message text               */
            MSG,                /* ...single line message        */
            OPER,               /* ...to invoking operator       */
            NULLCHAR);          /* ...not used                   */
    }
  Cnmvlc(&message,0,"LINETYPE = %c",getblock.Orig_line_type);
  Cnmsmsg(&message,MSG,OPER,NULLCHAR);
  Cnmvlc(&message,0,"HDRMTYPE = %c",getblock.Orig_msg_type);
  Cnmsmsg(&message,MSG,OPER,NULLCHAR);

  Cnmvlc(&message,0,"MSGID = %.8s",getblock.Orig_process);
                                         /* orig fields not NULL..*/
                                         /* ...terminated        */
  Cnmsmsg(&message,MSG,OPER,NULLCHAR);

  inbuf.buffer??(inbuf.size??) = '\0'; /* append NULL to end of  */
                                /* ...data retrieved by Cnmgetd  */

  Cnmvlc(&message,0,"MSGSTR = %s",inbuf.buffer);
  Cnmsmsg(&message,MSG,OPER,NULLCHAR);

                                /* get first trapped message...  */
  Cnmgetd(GETLINE,              /* ...function is get a message  */
          &inbuf,               /* ...result goes here           */
          255,                  /* ...max input length           */
          &getblock,            /* ...must provide a work area   */
          IDATAQ,               /* ...message is from automation */
          1);                   /* ...get the next line          */
}
Hlbptr->Hlbrc == CNM_GOOD;
```

Scope Checking

The following is an example of the scope checking capabilities provided by NetView. In this example, the user is required to set up the following elements for the command (shown below):

1. operator id
2. operator classes that can access the command
3. operator profile

The command gives the return code that the scope check service routine returned to the operator.

The syntax that this command checks for is:

```
CSCOPCK PARMx(VALx)
```

The following is the setup for the scope check example.

In DSIPARM(DSICMD):

- Define the operator classes that can access the command, its keywords, and its keyword values.
- The example below says that the command CSCOPCK can be executed by operators in scope class 1 and 2. Scope class 1 can issue any keyword or keyword value, but scope class 2 cannot use the value of VAL1 with keyword PARM2, and scope class 2 cannot issue PARM3 at all.

```
CSCOPCK   CMDMDL    MOD=CSCOPCK,RES=N,TYPE=RD
          CMDCLASS  1,2
PARM2     KEYCLASS  1,2
VAL1      VALCLASS  1
PARM3     KEYCLASS  1
VAL1      VALCLASS  1
```

In DSIPARM(DSIOPF):

- Define the operator ids and the profiles that the operator ids can use.

```
JOE     OPERATOR  PASSWORD=USER
        PROFILEN  DSIPROF3
```

In DSIPRF(profilename):

- Define the operator class value that will correspond to the profile that the operator logs on with.

```
DSIPROF3   PROFILE
           OPCLASS 3
           END
```

```
/********************************************************************/
/* Internal data definitions                                        */
/********************************************************************/
Dsivarch datain,                /* store data returned by cnmvars  */
         variable,              /* store variable name for Cnmvars */
         msg;                   /* var len char strng for messages */
int length = 24,                /* max len of data from Cnmvars    */
    x;                          /* used to increment value returned*/
                                /* ...by Cnmvars                   */

/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/
/********************************************************************/
/* Initialize the variable                                          */
/********************************************************************/
                                /* copy initial value into varying */
                                /* length character string...      */
Cnmvlc(&datain,                 /* ...varying length string        */
      0,                        /* ...do not convert to hex        */
       "Initial value");        /* ...initial value                */

                                /* copy variable name into varying */
                                /* length character string...      */
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);

                                /* put an initial value in variable*/
Cnmvars(PUT,                    /* ...function is write            */
       &datain,                 /* ...data is here                 */
       length,                  /* ...length not used              */
       &variable,               /* ...variable name                */
       "CGLOBAL ");             /* ...variable pool is cglobal     */
```

```
/*******************************************************************/
/* Find out the value of the variable                              */
/*******************************************************************/
                              /* read the global variable...     */
Cnmvars(GET,                  /* ...function is read             */
        &datain,              /* ...data goes here               */
        length,               /* ...truncate after 24 bytes      */
        &variable,            /* ...variable name                */
        CGLOBAL);             /* ...variable pool is cglobal     */

                              /* show operator value...          */
Cnmsmsg(&datain,              /* ...data obtained from varpool   */
        MSG,                  /* ...type is message              */
        OPER,                 /* ...send message to operator     */
        NULLCHAR);            /* ...not used                     */

                              /* put new value in varying length */
Cnmvlc(&datain,               /* character string...             */
       0,                     /* ...do not convert to hex        */
       "New Value");          /* ...new value                    */

/*******************************************************************/
/* set the global variable                                         */
/*******************************************************************/
                              /* update the global variable...   */
Cnmvars(PUT,                  /* ...function is write            */
        &datain,              /* ...data is here                 */
        length,               /* ...datalen is not used          */
        &variable,            /* ...variable name                */
        CGLOBAL);             /* ...variable pool is CGLOBAL     */

/*******************************************************************/
/* verify variable has been changed                                */
/*******************************************************************/
                              /* read the global variable ...    */
Cnmvars(GET,                  /* ...function is read             */
        &datain,              /* ...data goes here               */
        length,               /* ...truncate after 24 bytes      */
        &variable,            /* ...variable name                */
        CGLOBAL);             /* ...variable pool is CGLOBAL     */
                              /* show value to operator ...      */
Cnmsmsg(&datain,              /* ...data to be displayed         */
        MSG,                  /* ...type is message              */
        OPER,                 /* ...send message to operator     */
        NULLCHAR);            /* ...not used                     */

/*******************************************************************/
/* Release the lock to let other tasks update GVARIABLE            */
/*******************************************************************/
Cnmlk(UNLOCK,                 /* obtain the lock...              */
      &variable,              /* ...function is obtain lock      */
      "       ",              /* ...not used                     */
      "       ");             /* ...not used                     */

if (Hlbptr->Hlbrc == 0)
   Cnmsmsg(Cnmvlc(&msg,0,"got rid of lock"),MSG,OPER,NULLCHAR);
```

Altering Data

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

```
Example output with input of WHO:

Without exit:
  WHO

With exit:
  Command entered was: "WHO" at 12:00:00
```

```
/*********************************************************************/
/*                                                                   */
/*   Descriptive Name: High Level Language C Dsiex02a 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             */
Dsivarch *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       */
  Dsivarch  datain;              /* Old command text                */
  Dsivarch  time;                /* Area for time                   */
  Dsivarch  msgbuf;              /* message buffer                  */


  /*******************************************************************/
  /* Convert parameter pointers from character to hex addresses     */
  /*******************************************************************/
  sscanf(argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

  /*******************************************************************/
  /* Initialization                                                  */
  /*******************************************************************/

  /*******************************************************************/
  /*                                                                 */
  /*  Execution                                                      */
  /*                                                                 */
  /*******************************************************************/
                                 /* Retrieve the time...            */
  Cnminfc ("TIME    ",           /* ...variable is time of day      */
          &time,                 /* ...the result goes here         */
          255);                  /* ...max length of 255            */

                                 /* Peek the msg before altering    */
  Cnmgetd(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*/
                                 /* Echo'ed message?                */
  if (getblock.Orig_msg_type == '*') {
    Cnmvlc(&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                        */
}
```

Storage Access

The following example illustrates how to display the character representation of the contents of the storage that NetView can access. For example, after locating the address of the main vector table using DISPMOD DSIMNTEX, you can display the first four bytes of the DSIMVT control block, which for NetView R3 this will contain the character string NV13.

```
/*********************************************************************/
/* 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;
char inputbfr??(4097??) = " ";   /* Buffer where data is copied  */
char *srceptr;                   /* Address to copy from         */
int i,x;                         /* Work counter                 */
Dsivarch msgbuf;                 /* Buffer for Cnmsmsg           */

/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/
inbufr_p = inputbfr;
numbytes = 0;
numparms = sscanf((char *) &(Cmdbuf->buffer), /* parse cmd buffer */
           "%*s%x%x",             /* 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 parameters");
   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        */
 }

 if (xaddr != NULL)             /* If valid address then...     */
  {                             /* Copy storage...              */
    Cnmcpys(&xaddr,             /* ...from the address given    */
            &inbufr_p,          /* ...to the internal buffer    */
            numbytes,           /* ...for up to FFF bytes       */
            FIXTOFIX);          /* ...data is fixed length type */

    if (Hlbptr->Hlbrc == CNM_GOOD)    /* Good rc?               */
     {
       for(i=1; i <= (x=ceil(numbytes/64.0)*64); i = i + 64)
        {
        if (((numbytes - i) + 1) >= 64)
          {
          memmove(msgbuf.buffer,inputbfr+(i-1),64);
          msgbuf.size = 64;
          }
        else
          {
          memmove(msgbuf.buffer,inputbfr+(i-1),numbytes % 64);
          msgbuf.size = numbytes % 64;
          }
          Cnmsmsg(&msgbuf,      /* Display 64 bytes of storage...*/
                  MSG,          /* ...message                   */
                  OPER,         /* ...to the operator           */
                  NULLCHAR);    /* ...not used                  */
        }
     }
    else                        /* Else bad return code         */
     {
          Cnmvlc(&msgbuf,0,"Invalid or protected address");
          Cnmsmsg(&msgbuf,      /* Send error message...        */
                  MSG,          /* ...message                   */
                  OPER,         /* ...to the operator           */
                  NULLCHAR);    /* ...not used                  */
     }
  }
```

Data Set Access

The following is an example of opening (using CNMMEMO), reading (using CNMMEMR) and closing (using CNMMEMC) NetView partitioned data sets. This example reads a member of DSIPARM called DSIDMN, and displays it to the operator.

```
/*******************************************************************/
/* Internal data definitions                                       */
/*******************************************************************/
int  token;                       /* Token used to match open to   */
                                  /* ...read and close             */
Dsivarch msgbuf;                  /* Line that is read             */


/*******************************************************************/
/*                 Open the member                                 */
/*******************************************************************/
                             /* Open the data set member...        */
Cnmmemo(&token,              /* ...token returned by Cnmmemo       */
        "DSIPARM ",          /* ...ddname of PDS                   */
        "DSIDMN  ");         /* ...member name of PDS              */

if (Hlbptr->Hlbrc != CNM_GOOD)
 {
   Cnmvlc(&msgbuf,           /* Put error message in msgbuf...     */
          0,                 /* ...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                                */
/*******************************************************************/
                           /* Read the first record...             */
  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        */
                           /* Write out last record read...        */
        Cnmsmsg(&msgbuf,   /* ...write first 72 bytes              */
                MSG,       /* ...single line message               */
                OPER,      /* ...to the operator                   */
                NULLCHAR); /* ...taskname ignored                  */
                           /* Read the next record...              */
        Cnmmemr(token,     /* ...provide token from OPEN           */
                &msgbuf,   /* ...result goes here                  */
                80);       /* ...read 80 bytes                     */
   }

/*******************************************************************/
/*                  Close the member                               */
/*******************************************************************/
                           /* Close the PDS member...              */
   Cnmmemc(token);         /* ...provide token from OPEN           */
 }
```

CNMI

NetView provides the *Cnmcnmi* service routine for use in communicating with devices in the network via the Communications Network Management Interface (*Cnmi*). Any data that is returned may be accessed using the *Cnmgetd* service routine to retrieve records from the *Cnmi* solicited data queue (CNMIQ).

The following example uses the *Cnmcnmi* service routine to send a request-product-set-id data request to a specified PU. Any data returned is sent as a message to the operator.

The syntax of the command is:

CCNMI puname <OWN|ALL>

where:

puname is the name of the PU to be retrieved (required)

OWN implies that vital product data is to be retrieved for the PU only (default)

ALL implies that vital product data is to be retrieved for the PU and its attached ports

```
/*********************************************************************/
/* 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         */
  Dsivarch puname;                  /* puname varying length         */
  Dsivarch msgbuf;                  /* Message buffer                */
  Dsiorig getblock;                 /* Area for the work orig block  */
  Bigvlc datain;                    /* Buffer for the RU             */
  char ownorall??(4??);             /* Own or all placeholder        */
  Dsivarch fwdru;                   /* Forward RU                    */
  Dsivarch ru;                      /* RU data                       */
  Dsivarch own;                     /* 81 if own specified           */
  Dsivarch all;                     /* 83 if all specified           */
  Dsivarch puhdr;                   /* puname header                 */
  Dsivarch endofru;                 /* end of RU                     */
  char *ptr;                        /* ptr used to build fwdru       */

/*********************************************************************/
/*                                                                   */
/*  Vital Product Data RU definitions                                */
/*                                                                   */
/*  From the VTAM Progamming 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 CNM 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                                        */
/*********************************************************************/
```

```
/*******************************************************************/
/* Initialization                                                  */
/*******************************************************************/
Cnmvlc(&ru,1,"810810000000000E41038D00000111000006809002");
Cnmvlc(&own,1,"81");
Cnmvlc(&all,1,"83");
Cnmvlc(&puhdr,1,"F108");
Cnmvlc(&endofru,1,"00");
rcode = 0;

/*******************************************************************/
/*                                                                 */
/*  Execution                                                      */
/*                                                                 */
/*******************************************************************/
ptr = (char *) &fwdru.buffer;
count = sscanf((char *) &(Cmdbuf->buffer),"%*s%s%s",
               puname.buffer,ownorall);

puname.size = strlen(puname.buffer);
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+puname.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+puname.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,Origblck->Orig_task);
    rcode = 8;
    }
```

```
if (rcode == 0) {                /* Good so far?                 */
                                 /* Send RU over the CNMI...     */
   Cnmcnmi(SENDRPLY,             /* ...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               */
              Origblck->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              */
       }
    }
   else                          /* CNMI error                   */
```

```
      {
                                    /* Not invoked under a DST       */
        if (Hlbptr->Hlbrc == CNM_BAD_INVOCATION)
           Cnmvlc(&msgbuf,          /* Buffer for message text       */
                  0,                /* Do not convert to hex         */
                  "Must run under a DST");  /* Error message         */
         else
                                    /* PU never answered request     */
           if (Hlbptr->Hlbrc == CNM_TIME_OUT)
              Cnmvlc(&msgbuf,       /* Buffer for message text       */
                     0,             /* Do not convert to hex         */
                     "PU never answered");  /* Error message         */
            else
                                    /* PU gave a negative response   */
              if (Hlbptr->Hlbrc == CNM_NEG_RESPONSE)
                 Cnmvlc(&msgbuf,    /* Buffer for message text       */
                        0,          /* Do not convert to hex         */
                        "PU gave negative response");
               else
                                    /* Cnmi failure                  */
                 Cnmvlc(&msgbuf,    /* Buffer for message text       */
                        0,          /* Do not convert to hex         */
                        "CNMI request failed rc = %d",
                        Hlbptr->Hlbrc); /* Rc from CNMI routine      */

        Cnmsmsg(&msgbuf,            /* Send error message to user... */
                MSG,                /* ...single line message        */
                TASK,               /* ...dest. type task            */
                Origblck->Orig_task);     /* ...dest. id origin task */

      }                             /* End of CNMI error             */
   }                                /* End of Good so far            */
 Hlbptr->Hlbrc = rcode;             /* Issue rc                      */
```

VSAM (Keyed File Access)

The following is an example of coding a Netview HLL command processor that allows I/O to a VSAM file via the *Cnmkio* service routine.

It must execute on a DST. To run this command on a DST, either use the *Cnmsmsg* service routine (with a type of COMMAND) or use the EXCMD command.

This example will create a data base that contains 5 records with the following keys and data:

KEY	DATA
01	A
02	B
03	C
04	D
05	E

```
/*******************************************************************/
/* Internal data definitions                                       */
/*******************************************************************/
Dsivarch rec;                  /* store output data for Cnmkio  */
Dsivarch inrec;                /* store data returned by Cnmkio */
Dsivarch key;                  /* store key for Cnmkio          */
Dsivarch 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.                                                     */
/*                                                                 */
/*******************************************************************/
keyptr = keydata;
outptr = outdata;
for (i = 0; i <= 4; i++)       /* For 5 records                 */
 {
   memmove(key.buffer,keyptr,2);  /* Set key portion of record   */
   key.size = 2;                  /* Set size of key buffer      */
   keyptr = keyptr+2;             /* Get next key in table       */
   memmove(rec.buffer,key.buffer,2); /* Record must have key 1st */
   memmove(rec.buffer+2,outptr,1);/* attach data to key          */
   rec.size = 3;                  /* Set size of record          */
   outptr = outptr+1;             /* Set pointer to record       */

                               /* Call KEYIO...                 */
   Cnmkio(GET_EQ,              /* ...requesting a get           */
          &inrec,              /* ...data goes in here          */
          10,                  /* ...10 bytes max input         */
          &key,                /* ...key is in key              */
          UPDATE);             /* ...this is an update          */
```

```
    if (Hlbptr->Hlbrc == CNM_NOT_FOUND)
                                /* Call KEYIO...              */
      Cnmkio(PUT,               /* ...requesting a put        */
             &rec,              /* ...data goes in here       */
             0,                 /* ...not used                */
             &key,              /* ...key is in key           */
             DIRECT);           /* ...this is an update       */
     else
                                /* Call KEYIO...              */
      Cnmkio(PUT,               /* ...requesting a put        */
             &rec,              /* ...data goes in here       */
             0,                 /* ...not used                */
             &key,              /* ...key is in key           */
             UPDATE);           /* ...this is an update       */

    if (Hlbptr->Hlbrc != CNM_GOOD)
      {                         /* if put failed              */
                                /* put message in varying length */
     Cnmvlc(&msg,               /* character string...        */
            0,                  /* do not convert to hex      */
            "Cnmkeyio PUT request failed with RC: %d",Hlbptr->Hlbrc);
                                /* put out error message...   */
     Cnmsmsg(&msg,              /* ...message text            */
            MSG,                /* ...type is message         */
            OPER,               /* ...send to issuing operator */
            NULLCHAR);          /* ...not used                */
      }
 }

/******************************************************************/
/*                    Read in the 5 records...                    */
/******************************************************************/
keyptr = keydata;
for (i = 0; i <= 4; i++)        /* For 5 records              */
 {
    memmove(key.buffer,keyptr,2); /* Set key portion of record  */
    key.size = 2;
    keyptr = keyptr+2;
                                /* Call KEYIO...              */
    Cnmkio(GET_EQ,              /* ...requesting a put        */
           &inrec,              /* ...data goes in here       */
           10,                  /* ...not used                */
           &key,                /* ...key is in key           */
           NOUPDATE);           /* ...this is an not update   */

     inrec.buffer??(inrec.size??) = '\0';  /* add NULL to end of */
                                           /* ...data            */
     Cnmvlc(&msg,0,"Key: %.2s Record: %s",key.buffer,inrec.buffer);
     Cnmsmsg(&msg,MSG,SYSOP,NULLCHAR);/* display key and record  */
 }
Hlbptr->Hlbrc = CNM_GOOD;       /* Issue clean rc             */
```

DST User Exit

The following is an example of coding a Netview HLL user exit routine that primes an empty VSAM data base for a DST. If a VSAM data base has not been primed (has at least one record), subsequent I/O requests will fail.

```
/*********************************************************************/
/*                                                                   */
/*   Descriptive Name: High Level Language C DSIEX02A Example        */
         .                    .                  .
         .                    .                  .
         .                    .                  .
/*   Change Activity:                                                */
/*      date,author: description of changes                         */
/*********************************************************************/
#pragma runopts (NOEXECOPS,NOSTAE,NOSPIE,ISASIZE(4K),ISAINC(4K))

/*********************************************************************/
/* Standard include files
/*********************************************************************/
#include <string.h>         /* String functions                      */
#include <stdlib.h>         /* String functions                      */
#include <stdarg.h>         /* String functions                      */
```

```
/*********************************************************************/
/* NetView high level language include files                         */
/*********************************************************************/
#include "dsic.h"                  /* Include HLL macros             */

/*********************************************************************/
/* External data definitions                                         */
/*********************************************************************/
Dsihlb   *Hlbptr;                  /* Pointer to the HLB             */
Dsivarch *Cmdbuf;                  /* Pointer to command buffer      */
Dsiorig  *Origblck;                /* Pointer to Origin block        */

main(int argc, char *argv??(??))
{
  /*******************************************************************/
  /* Internal data definitions                                       */
  /*******************************************************************/

  /*******************************************************************/
  /* Convert parameter pointers from character to hex addresses      */
  /*******************************************************************/
  sscanf(argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

  /*******************************************************************/
  /* 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            */
}
```

User Exit

The following is an example of coding a user 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 will not be set. The CSNDDAT command (see "SEND Side" on page 160) is used to interrogate the variable value.

```
/*********************************************************************/
/*                                                                   */
/*  Descriptive Name: High Level Language PL/I DSIEX03 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               */
Dsivarch *Cmdbuf;                /* Pointer to command buffer        */
Dsiorig  *Origblck;              /* Pointer to Origin block          */

Dsivarch time;                   /* Time last command entered        */
Dsivarch cvname;                 /* Name of variable for Cnmvars     */

main(int argc, char *argv??(??))
{
  /*******************************************************************/
  /* Internal data definitions                                       */
  /*******************************************************************/

  /*******************************************************************/
  /* Convert parameter pointers from character to hex addresses      */
  /*******************************************************************/
  sscanf(argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

  /*******************************************************************/
  /* Initialization                                                  */
  /*******************************************************************/

  /*******************************************************************/
  /*                                                                 */
  /*  Execution                                                      */
  /*                                                                 */
  /*******************************************************************/
                                 /* Command other than CSNDDAT?      */
  if (strstr(Cmdbuf->buffer,"CSNDDAT") == NULL)
   {                             /* Yes...                           */
     Cnminfc("TIME    ",         /* ...what time is it?              */
             &time,              /* ...answer goes here              */
             255);               /* ...length of time                */

     Cnmvlc(&cvname,             /* Set name of variable...          */
            0,                   /* ...do not convert to hex         */
            "LAST_COMMAND_TIME");      /* ...name of variable        */

     Cnmvars(PUT,                /* Put answer in task global...     */
             &time,              /* ...information in TIME           */
             0,                  /* ...length of time                */
             &cvname,            /* ...by the name of                */
             TGLOBAL);           /* ...task global pool              */
   }
Hlbptr->Hlbrc = USERASIS;        /* clear rc                         */
}
```

Wait for Data

WAIT Side

The following is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example is 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 "User Exit" on page 155) and this value is retrieved by the CSNDDAT command(see "SEND Side" on page 160) that is invoked on the target task. The code in this example is the CWATDAT command.

The syntax of the command is:

```
CWATDAT taskname
```

The flow of the wait for data function is:

```
                              TARGET
 OST                            OST

Invokes
CWATDAT  command
and specifies the
target task to
send the request


CWATDAT  using
CNMSMSG sends a  ----->
request to the
OST specified

OST issues a WAIT
FOR DATA

                                CSNDDAT command is
                                invoked on the
                                OST.  It finds the
                                task global variable
                                set by DSIEX03.


                                CNMSMSG type of
                             <--- DATA is invoked with
                                the value retrieved

OST wait is
satisfied --- wake up
and issue  message
to the operator
```

```
/********************************************************************/
/* Internal data definitions                                        */
/********************************************************************/
Dsiorig getblock;                /* Area for the Orig Block         */
Dsivarch time;                   /* Time last command entered       */
char targtask??(9??);            /* Task of inquiry                 */
int len;                         /* length of targtask              */
Dsivarch msgbuf;                 /* Message buffer                  */
char *token;                     /* used to parse command           */

/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/

token = strtok((char *) &(Cmdbuf->buffer)," ");  /* parse command */
token = strtok(NULL," ");    /* buffer for target task name       */

if (strlen(token) > 8)       /* node name invalid                 */
  token = NULL;

strcpy(targtask,token);

len = strlen(targtask);
strncat(targtask,"        ",8 - len);


if (token != NULL)               /* Was target task entered?       */
 {                               /* Syntax ok...                   */
 if (strncmp(targtask,(char *) &(Origblck->Orig_task),
     strlen(targtask)) == 0)
   {              /* is operator who issued CWATDAT being queried?*/
                                 /* put error message in a varying*/
   Cnmvlc(&msgbuf,               /* length character string...    */
          0,                     /* ...do not convert to hex      */
          "Target task cannot be task invoking CWATDAT");

                                 /* display the message...        */
   Cnmsmsg(&msgbuf,              /* ...the message text           */
           MSG,                  /* ...type is message            */
           OPER,                 /* ...send to invoking operator  */
           NULLCHAR);            /* ...not used                   */
   }
  else
   {
   Cnmvlc(&msgbuf,               /* Put command in msgbuf         */
          0,                     /* ...do not convert to hex      */
          "CSNDDAT");            /* ...command                    */

   Cnmsmsg(&msgbuf,              /* Invoke CSNDDAT command        */
           REQUEST,              /* ...type is request            */
           TASK,                 /* ...on a task                  */
           targtask);            /* ...specified by input command */
   Cnmvlc(&msgbuf,              /* Put WAIT command in msgbuf    */
          0,                    /* ...do not convert to hex      */
          "WAIT 120 SECONDS FOR DATA");

   Cnmcmd(&msgbuf);             /* Invoke WAIT command           */

   if (Hlbptr->Hlbrc != CNM_DATA_ON_WAIT) /* Wait successful?*/
    {                           /* No...                         */
     Cnmvlc(&msgbuf,            /* Put error message in msgbuf   */
            0,                  /* ...do not convert to hex      */
            "Wait for data abnormally ended");

     Cnmsmsg(&msgbuf,           /* Send error message...         */
             MSG,               /* ...type is message            */
             OPER,              /* ...to the operator            */
             NULLCHAR);         /* ...not used                   */
    }
   else                         /* Wait was successful           */
    {                           /* Process the results           */
     Cnmgetd(GETMSG,            /* Read in the response          */
             &time,             /* ...read into time variable    */
             256,               /* ...give plenty of room        */
             &getblock,         /* ...provide own origin block   */
             DATAQ,             /* ...on the data queue (3)      */
             0);                /* ...index not used             */

/*****************************************************************/
/*  Remove process and task id from the buffer !!!!              */
/*  First 8 bytes and the last 8 bytes                           */
/*****************************************************************/
     strncpy(msgbuf.buffer,time.buffer+8,time.size-8);
     msgbuf.size = time.size - 8;

     Cnmsmsg(&msgbuf,           /* Inform user...                */
             MSG,               /* ...type is message            */
             OPER,              /* ...to the operator            */
             NULLCHAR);         /* ...not used                   */
    }
   }
 }
else                            /* Target task not entered       */
 {
   Cnmvlc(&msgbuf,              /* Put error message in msgbuf   */
          0,                    /* ...do not convert to hex      */
          "Invalid Target Task");

     Cnmsmsg(&msgbuf,           /* Inform user of syntax error   */
             MSG,               /* ...type is message            */
             OPER,              /* ...to the operator            */
             NULLCHAR);         /* ...not used                   */
 }
```

SEND Side

The following is part of an example for sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example 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 "User Exit" on page 155) This value is retrieved by the CSNDDAT command that is invoked by the CWATDAT command(see "Wait for Data" on page 157) on the target task. This command processor is executed when the CSNDDAT command is entered.

```
/********************************************************************/
/* Internal data definitions                                        */
/********************************************************************/
Dsivarch time,                    /* Date the last command was     */
                                  /* ...was entered                */
        msgbuf,                   /* Buffer for Cnmsmsg            */
        cvname,                   /* Buffer for name of variable   */
        myopid;                   /* operator id we are running    */
                                  /* ...under                      */

/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/
                                  /* determine my opid             */
Cnminfc("OPID    ",               /* ...variable is opid           */
       &myopid,                   /* ...put result here            */
       8);                        /* ...truncate after 8 bytes     */

if (strncmp((char *) &(myopid.buffer),(char *) &(Origblck->Orig_task),
    myopid.size) == 0)
  {                               /* CSNDDAT invoked directly?     */
                                  /* put error message in a varying*/
  Cnmvlc(&msgbuf,                 /* length character string...    */
        0,                        /* ...do not convert to hex      */
        "Cannot issue CSNDDAT directly"); /* CSNDDAT command...    */
                                  /* ...issued directly            */

                                  /* display the message...        */
  Cnmsmsg(&msgbuf,                /* ...the message text           */
         MSG,                     /* ...type is message            */
         OPER,                    /* ...send to invoking operator  */
         NULLCHAR);               /* ...not used                   */
  }
else
  {
  Cnmvlc(&cvname,                 /* Set variable name             */
        0,                        /* ...do not convert to hex      */
        "LAST_COMMAND_TIME");    /* ...variable name              */

                                  /* Retrieve last time variable   */
  Cnmvars(GET,                    /* ...read in the value          */
         &time,                   /* ...into time                  */
         256,                     /* ...truncate at 256            */
         &cvname,                 /* ...of the variable            */
         TGLOBAL);                /* ...in the task global pool    */
if (Hlbptr->Hlbrc == CNM_GOOD) /* Variable set?               */
 {                             /* Yes, continue...             */
   Cnmvlc(&msgbuf,             /* Put data in msgbuf...        */
          0,                   /* ...do not convert to hex     */
          "%.8sLast command entered at: %.8s",/* ...must precede*/
          Origblck->Orig_process, /* data with origin process id*/
          time.buffer);          /* ...put in time             */

                               /* Send data back to requestor   */
   Cnmsmsg(&msgbuf,            /* ...text of message           */
           DATA,               /* ...message is data           */
           TASK,               /* ...to a task                 */
           Origblck->Orig_task);    /* ...that invoked this     */
 }
else                           /* No inform user...            */
 {
   Cnmvlc(&msgbuf,             /* Put error message in msgbuf  */
          0,                   /* ...do not convert to hex     */
          "%.8sMust install DSIEX03 to set time variable",
           Origblck->Orig_process);

                               /* Send message to requestor    */
   Cnmsmsg(&msgbuf,            /* ...text of error message     */
           DATA,               /* ...message is data           */
           TASK,               /* ...to a task                 */
           Origblck->Orig_task);    /* ...that invoked this     */
 }
}
```

Chapter 10. Compiling, Link-Editing, and Running Your C Program

Once you have a C compiler installed, you can modify the C compile and link-edit JCL to use with NetView. The objective of this chapter is to provide the information necessary to make these modifications.

Several examples of compile and link-edit JCL are provided in this chapter. These are given as examples only. You are responsible for modifying the compile and link-edit JCL samples that were shipped with the C compiler.

You must have completed the installation steps for HLL as described in the *NetView Installation and Administration Guide* before attempting to execute C programs in the NetView environment.

Compiling

In order to compile C programs using NetView services, it is necessary to modify the compile step in the JCL to reference the NetView macro library(s). You will need to include in the compile JCL a SYSLIB statement for SYS1.MACLIB. An example of modifications to compile step JCL is shown below:

```
//COMPILE EXEC PGM=EDCCOMP,PARM=('RENT'),REGION=&CREGSIZ
                               .
                               .
                               .
//SYSLIB  DD   DSN=SYS1..MACLIB,DISP=SHR
                               .
                               .
                               .
```

Link-editing

The following rules apply when link-editing C modules:

- All C load modules must be REENTRANT.
- C load modules can reside in 24 or 31 bit storage and can be entered in either addressing mode.
- All C load modules must be link-edited with DSIHSTUB and DSIEXC. DSIHSTUB must be the ENTRY point.

In order to link-edit a C module to run with NetView, you must modify the link-edit step in the JCL to reference the appropriate NetView Library(s). This will allow you to include DSIHSTUB and DSIEXC at link-edit time. Add SYS1.NVULIB and SYS1.LINKLIB to the list of automatic call libraries already defined by SYSLIB in the C link-edit step of the JCL. An example is shown.

```
//LKED     EXEC  PGM IEWL,
//    PARM='XREF,RENT,LET,LIST,AMODE=&AMODE,RMODE=&RMODE',
//    REGION=4096K,COND=(8,LE,COMPILE)
                              .
                              .
                              .
//SYSLIB   DD     DSN=SYS1.NVULIB,DISP=SHR
//         DD     DSN=SYS1.LINKLIB,DISP=SHR
                              .
                              .
                              .
  INCLUDE         SYSLIB(DSIHSTUB)
  INCLUDE         SYSLIB(DSIEXC)
  ORDER           DSIHSTUB
  ENTRY           DSIHSTUB
  MODE            AMODE(31),RMODE(ANY)
  NAME            LMODNAME(R)
```

Note: All HLL modules must be compiled and link-edited with the RENT option. For IBM C/370, you will have to execute the PRE-LINKEDIT step and code the RENT compiler option. The resulting object deck(s) must then be link-edited with the RENT option.

Running

A set of run-time libraries will be shipped with the compiler. In order to execute a C program in the NetView environment, you must modify your NetView start up procedure to reference the appropriate C run-time libraries. Refer to the *NetView Installation and Administration Guide* for more information.

HLL command processors require a CMDMDL statement in member DSICMD of the DSIPARM data set. User exits are loaded at initialization and need to conform to user exit naming conventions. For more information on user exits see Chapter 2 on page 9.

Part 4. HLL Debugging and Service Routine Reference

Chapter 11. Testing And Debugging

It is assumed that your HLL module has been compiled and link-edited successfully upon entry into this chapter. Upon completion of this chapter, the user should be familiar with the NetView Remote Interactive Debugger (RID) and how it can be used to debug HLL programs.

Remote Interactive Debugger (RID)

NetView's Remote Interactive Debugger (RID), gives you the ability to checkpoint entry and exit parameters to HLL service routines and display storage at various predetermined debug points in the code.

Using RID to Monitor a Task

NetView's interactive debug facility gives you the ability to monitor and debug HLL modules during execution. It is necessary to determine a monitoring task and a target task to debug HLL modules effectively. The monitoring task must be an OST. It is from this task that you will issue the RID commands which control the execution of the HLL module running under the target task. The target task may be an OST, PPT, 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 will continue 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 will display parameters upon entry to (HAPIENTR) and exit from (HAPIEXIT) HLL service routines.

RID Command

The following syntax describes the RID command as it is issued from the monitoring or debugging task.

```
RID TASK=opid
   [,STEP|RUN|CONTINUE|END]
   [,MODNAME=*|name]
   [,OPTION=*|HAPIENTR|HAPIEXIT]
```

Where:

STEP
When the STEP option is specified, the target task will be stopped whenever control is given to a debug point that matches the criteria specified 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.

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 point(s). The RUN option will resume execution of a task stopped in STEP mode.

CONTINUE

The CONTINUE option is used to resume execution of a task that was stopped by the STEP option of RID. New debug point match criteria may be specified in conjunction with the CONTINUE option. The CONTINUE keyword is provided for readability only. Execution can be resumed by reissuing the RID command with its original operands.

END

The END option will cause RID debugging of a task to cease and allow 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

Name of the module being monitored by RID. If * is specified, RID will monitor all HLL programs running under the target task. * is the default.

OPTION

Specifies the type of debug point. * is the default.

*	All debug points will be displayed.
HAPIENTR	Entry to HLL API service routine
HAPIEXIT	Exit from HLL API service routine

Usage Notes:

A NetView task can only be monitored (using RID) by one NetView operator at a time.

It is not recommended to use RID to monitor or debug HLL command processors running under the PPT. Running RID against the PPT causes the PPT to get suspended which could cause undesirable results if timer sensitive functions (such as AT or EVERY) are being performed.

The user is required to code a list of parameters for each HLL service routine invocation. When activated, RID will display these parameters and their values on entry to and exit from each HLL service routine. The maximum number of data fields that RID is capable of displaying is ten. This is important when using RID to monitor calls to and from CNMSCAN. The user can specify a maximum of 14 arguments on a call to CNMSCAN. This means that the user may not be able to view the complete parameter list when monitoring CNMSCAN invocations.

The default value (*) for MODNAME and OPTION will be used until a value is specified. Once a value has been specified, it will be used on all successive RID invocations unless explicitly overridden. MODNAME and OPTION will be reset to the default values once RID is invoked with the END option.

Return Codes:

CNM_GOOD	0	Everything OK
CNM_NOT_FOUND	20	Task not found
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_OPTION	128	Invalid option
CNM_BAD_TASKNAME	164	Task name too long
CNM_BAD_MODNAME	168	Module name too long
CNM_BAD_COMBO	176	Invalid combination of options
CNM_TVB_INUSE	180	TVB in use
CNM_RID_INUSE	184	RID in use
CNM_RID_SELF	188	Debug task and target task can not be the same.
CNM_BAD_PUSH + X	4000 + X	Non-zero return code X from DSIPUSH macro. (See *NetView Customization: Using Assembler* for more information on the DSIPUSH macro).
CNM_BAD_SNTXS + X	17000 + X	Non-zero return code X. See values for X below.

Values for X:

	4	Invalid syntax.

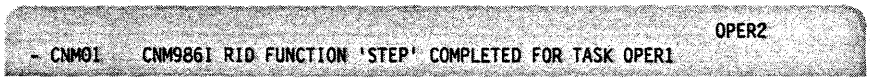
Remote Interactive Debugger RID Scenarios

The following scenario describes how RID can be used to monitor or debug an HLL module. For the purpose of this example, we have assumed that the user has chosen to debug an HLL program that has already been compiled and link-edited into NetView. The process for debugging HLL modules written in C and PL/I are similar except for some slight differences in the RID panels that are displayed. For this example, YOURPGM is the name of the HLL command processor that will be monitored while executing under target task OPER1. OPER2 has been chosen as the monitoring or debugging task.

Invoke RID by issuing the following command from OPER2:

```
RID TASK=OPER1
```

The following shows the system response. RID defaults to step mode operation. Execution of YOURPGM will be halted at each debug point.

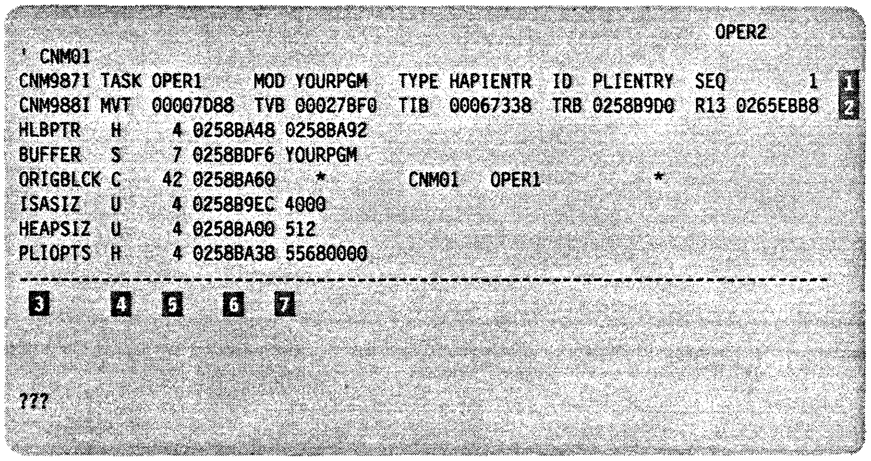


```
                                                                  OPER2
 - CNM01    CNM986I RID FUNCTION 'STEP' COMPLETED FOR TASK OPER1
```

From OPER1, type the name of the HLL program which you have chosen to debug.

```
YOURPGM
```

The following screen is displayed on OPER2's console. This is the entry screen (HAPIENTR) for your PL/I program (ID=PLIENTRY). The ID for the entry screen for a C program is CENTRY.



```
                                                                  OPER2
' CNM01
CNM987I TASK OPER1     MOD YOURPGM   TYPE HAPIENTR  ID  PLIENTRY  SEQ        1  1
CNM988I MVT  00007D88  TVB 00027BF0  TIB  00067338  TRB 0258B9D0  R13 0265EBB8  2
HLBPTR   H     4 0258BA48 0258BA92
BUFFER   S     7 0258BDF6 YOURPGM
ORIGBLCK C    42 0258BA60    *          CNM01   OPER1          *
ISASIZ   U     4 0258B9EC 4000
HEAPSIZ  U     4 0258BA00 512
PLIOPTS  H     4 0258BA38 55680000
----------------------------------------------------------------------------
 3        4   5    6    7



???
```

Figure 5. PL/I Entry Screen for YOURPGM

The numbered boxes in Figure 5 are described as follows:

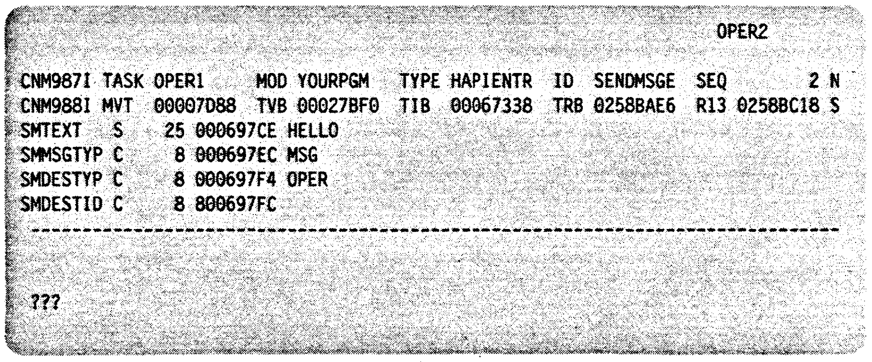
1	Message CNM987I displays the following information: **TASK** Name of the target task being monitored by RID **MOD** Name of the module being monitored by RID **TYPE** Type of debug point currently being displayed **ID** Unique identifier for the debug point being displayed. **SEQ** The sequence number of this RID panel.
2	Message CNM988I displays the addresses of the MVT, TVB, TIB, TRB, and the contents of Register 13 which points to your SAVEAREA. MVT, TVB and TIB are described in the control block reference section of the *NetView Customization: Assembler* manual. TRB is the Transaction Block and is used only by IBM service.

3	HLBPTR, BUFFER, ORIGBLCK are the initial parameters passed to your command processor from NetView. 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: H-Hex, D-Dump, C-Character, S-String, U-Unsigned, I-Integer A-Address.
5	Lengths of the variables in 3 that are expected by the HLL service routines.
6	Addresses in storage where the values for the variables in 3 are stored.
7	Values associated with each of the variables in 3.

Continue to the next debug point by entering the following RID command from OPER2:

```
RID  TASK=OPER1,CONTINUE
```

Since OPTION was not specified, RID will display panels upon entrance to (HAPIENTR) and exit from (HAPIEXIT) all HLL service routines invoked from /* the HLL program being debugged. In Figure 6, RID is displaying the parameters on entry to (HAPIENTR) the CNMSMSG service routine. The parameters for each HLL service routine are explained in Chapter 12.



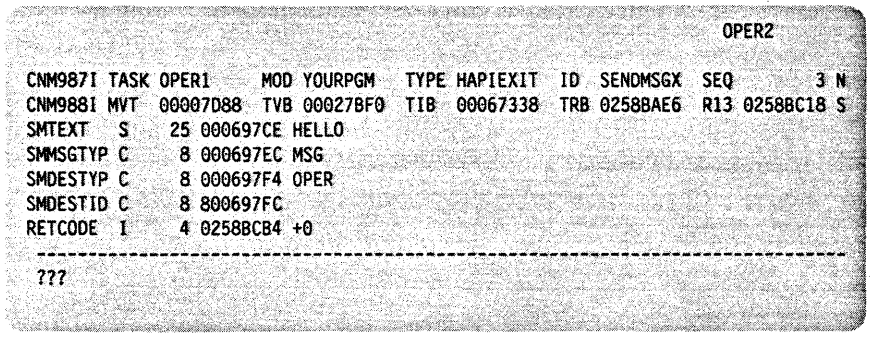
```
                                                                      OPER2

CNM987I TASK OPER1    MOD YOURPGM   TYPE HAPIENTR  ID  SENDMSGE  SEQ        2 N
CNM988I MVT  00007D88  TVB 00027BF0  TIB  00067338  TRB 0258BAE6  R13 0258BC18 S
SMTEXT   S    25 000697CE HELLO
SMMSGTYP C     8 000697EC MSG
SMDESTYP C     8 000697F4 OPER
SMDESTID C     8 800697FC
 -----------------------------------------------------------------------------


 ???
```

Figure 6. Entry Screen for CNMSMSG

Continue to the next debug point by entering the following RID command from OPER2:

```
RID  TASK=OPER1
```

Notice that it is not necessary to issue the RID command with the CONTINUE operand when you want to resume execution of a task. CONTINUE is used for readability. Figure 7 on page 172 displays the parameters on exit from the CNMSMSG service routine. Note that RETCODE has been added. RETCODE is the value of HLBRC (*Hlbrc* for C programs) on exit from an HLL service routine.



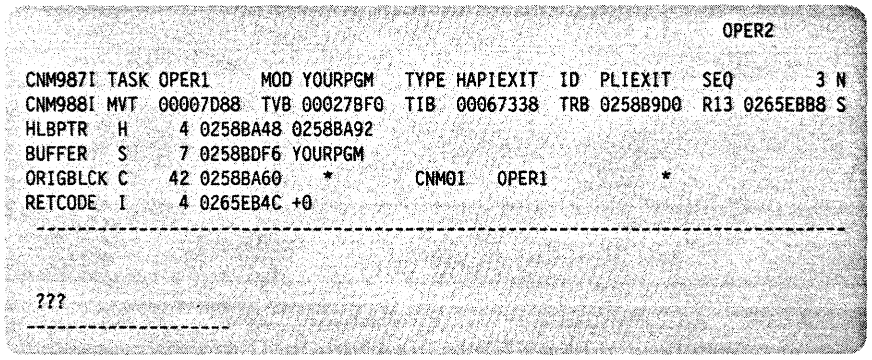
```
                                                                   OPER2

CNM987I TASK OPER1     MOD YOURPGM   TYPE HAPIEXIT  ID  SENDMSGX  SEQ        3 N
CNM988I MVT  00007D88  TVB 00027BF0  TIB  00067338  TRB 0258BAE6  R13 0258BC18 S
SMTEXT   S   25 000697CE HELLO
SMMSGTYP C    8 000697EC MSG
SMDESTYP C    8 000697F4 OPER
SMDESTID C    8 800697FC
RETCODE  I    4 0258BCB4 +0
--------------------------------------------------------------------------------
 ???
```

Figure 7. Exit Screen for CNMSMSG

Continue to the next debug point by entering the following RID command from OPER2:

```
RID  TASK=OPER1
```

The final RID panel displayed in Figure 8 is a PL/I exit panel that corresponds to the PLIENTRY panel in Figure 6 on page 171. Notice that TYPE= HAPIEXIT and ID=PLIEXIT. If this were a C program, ID would be CEXIT. Notice that RETCODE has been added but ISASIZ, HEAPSIZ and PLIOPTS are no longer displayed.



```
                                                                   OPER2

CNM987I TASK OPER1     MOD YOURPGM   TYPE HAPIEXIT  ID  PLIEXIT   SEQ        3 N
CNM988I MVT  00007D88  TVB 00027BF0  TIB  00067338  TRB 0258B9D0  R13 0265EBB8 S
HLBPTR   H    4 0258BA48 0258BA92
BUFFER   S    7 0258BDF6 YOURPGM
ORIGBLCK C   42 0258BA60    *       CNM01    OPER1          *
RETCODE  I    4 0265EB4C +0
--------------------------------------------------------------------------------


 ???
--------------------
```

Figure 8. PL/I Exit Screen for YOURPGM

Chapter 12. Command and Service Reference

This chapter assumes that you have an understanding of the information discussed in the previous chapters. Upon completion of this chapter, you should have an understanding of each of the HLL commands and service routines and their associated parameters.

It primarily contains general-use programming interfaces, which allow the customer to write programs that use the services of NetView. However, this chapter also provides the following types of information, which are explicitly identified where they occur:

Installation exits and other *product-sensitive interfaces* are provided to allow the customer installation to perform tasks such as product tailoring, monitoring, modification, or diagnosis. They are dependent on the detailed design or implementation of the product. Such interfaces should be used only for those specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Notational Conventions

The following notational conventions apply to the commands and service routines described in this chapter.

lowercase item
: Lowercase bold-faced letters represent parameters for which you must supply the value, address, or name, rather than the literal information.

UPPERCASE item
: Uppercase bold-faced letters represent the valid literal values for a specified parameter.

underscored item
: An underscored item represents the default value of a particular parameter. If you specify no parameter, NetView uses the default value.

Braces { }
: Small braces enclose the different options for a parameter. Large braces enclose mutually exclusive parameters; you must select one, and only one, of these parameters. Do not include the braces when coding the information.

Brackets [ ]
: Brackets enclose an optional parameter. Optional parameters can be included or omitted independently of other parameters. Do not include the brackets when coding the information.

OR-sign |
: The OR-sign separates the options for a required (brace-enclosed) parameter or for an optional (bracket-enclosed) parameter. For a required parameter, one of the options must be coded. For an optional parameter, none of the options have to be coded. Do not type the OR-sign when coding the information.

Composite Return Codes

Most of the NetView commands, command lists, and High-Level Language (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 known (constant) 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 following:

1. the return code represented in terms of constants and unknown values (if applicable)
2. the return code represented in terms of resolved constants and unknown values (if applicable)
3. a description of why the return code was issued.

Several of the descriptions will refer the user to a NetView macro. Each of these macros is referenced in *NetView Customization: Using Assembler*.

The unknown values of a composite return code can be resolved in the following manner.

1. Start with the following equation:

   ```
   HLBRC = 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) and DSICCNM (see Appendix C). In the case where there is more than one unknown value to resolve (X and Y), the remaining calculated value will be split into a major and minor 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:

```
CNM_BAD_PUSH + X
```

After resolving the known values, we see that the unknown value X, is equal to 4. The user should refer to the return code section of CNMNAMS, which would indicate that the return code was actually generated by the DSIPUSH macro. See this macro in *NetView Customization: Using Assembler* for the description of the return code.

```
           HLBRC = CNM_BAD_PUSH + X
            4004 = 4000 + X
     4004 - 4000 = X
               4 = X
```

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 - CNM_BAD_EXCMS
```

Resolve the known values:

```
                HLBRC = X - CNM_BAD_EXCMS
                -3108 = X - 3000
         -3108 + 3000 = X
                 -108 = X
```

A value of -108 for X implies that X = SWBEXCNF - Y. (See the return code section of CNMCMD.) After resolving the known values, the user will see that the unknown value Y, is 8. The user should again refer to the return code section of CNMCMD which would indicate that the return code was actually generated by the DSICES macro. See this macro in *NetView Customization: Using Assembler* for the description of the return code.

```
                 -108 = SWBEXCNF - Y
                 -108 = -100 - Y
           -108 + 100 = -Y
          -(-108+100) = Y
                    8 = Y
```

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:

```
                CNM_BAD_ZCSMS + (X * 100) + Y
```

After resolving the known values, the user will see that the unknown values X and Y are 16 and 0 respectively. The user should refer to the return code section of CNMCNMI which would indicate that the return code was actually generated by the DSIZCSMS macro. See this macro in *NetView Customization: Using Assembler* for the description of the major and minor return codes.

```
                HLBRC = CNM_BAD_ZCSMS + (X * 100 ) + Y
                21600 = 20000 + (X * 100) + Y
        21600 - 20000 = (X * 100 ) + Y
                 1600 = (X * 100) + Y

             =>  1600 / 100
                 MAJOR_RC is the quotient  => 16
                 MINOR_RC is the remainder => 0
```

This can also be seen visually:

```
       16 00 => 16 = MAJOR_RC
                00 = MINOR_RC
```

Example (4)

Upon completion of a call to CNMCNMI service routine, HLBRC=20408. A return code value in the 20000 range implies that the composite return code equation is:

```
CNM_BAD_ZCSMS + (X * 100) + Y
```

After resolving the known values, the user will see that the unknown values X and Y are 4 and 8 respectively. The user should refer to the return code section of CNMCNMI which would indicate that the return code was actually generated by the DSIZCSMS macro. See this macro in *NetView Customization: Using Assembler* for the description of the major and minor return codes.

```
        HLBRC = CNM_BAD_ZCSMS + (X * 100 ) + Y
        20408 = 20000 + (X * 100) + Y
20408 - 20000 = (X * 100 ) + Y
          408 = (X * 100) + Y

     =>  408 / 100
         MAJOR_RC is the quotient  => 4
         MINOR_RC is the remainder => 8
```

This can also be seen visually:

```
4 08 =>  4 = MAJOR_RC
         8 = MINOR_RC
```

Example (5)

Upon completion of a call to CNMKIO, HLBRC=28692. Only two composite return codes are issued from CNMKIO. See the return code section of CNMKIO. Since the return code value is <u>NOT</u> in the 2000 range (this would indicate a DST failure), the value of 28692 implies that the composite return code equation is:

```
(CNM_BAD_ZVSMS + X) * 256 + Y
```

This equation is difficult to resolve without knowing either X or Y. Use the following equations to determine the major and minor return codes:

```
MAJOR_RC = (HLBRC / 256) - 100
         = (28692 / 256) - 100   (keep only the quotient)
         = 112 - 100
         = 12

MINOR_RC = HLBRC - ((CNM_BAD_ZVSMS + MAJOR_RC) * 256)
         = 28692 - ((100 + 12) * 256)
         = 28692 - (112 * 256)
         = 28692 - 28672
         = 20
```

You should refer to the return code section of CNMKIO which would indicate that the return code was actually generated by the DSIZVSMS macro. See this macro in *NetView Customization: Using Assembler* for the description of the major and minor return codes.

Command Reference

The following commands are useful when executing HLL command processors. The GO, QUEUE, and RESET commands are operator commands and may be issued from the operator console or from an HLL command processor via CNMCMD. The TRAP and WAIT commands must be issued from within an HLL command processor via CNMCMD. Refer to *NetView Customization: Writing Command Lists* for information about using TRAP and WAIT in a command list.

GO Command

The GO command allows you to resume running a command procedure that is in a PAUSE or WAIT state. You can also use the GO command to pass values to a command procedure that is in a PAUSE state.

HLL Usage Notes

The GO command can be entered from a terminal to satisfy a WAIT or PAUSE. A return code of CNM_GO_ON_WAIT will be generated when GO is entered in the following situations:

1. The event specified on the WAIT command is MESSAGES (timer may or may not be set)
2. The event specified on the WAIT command is DATA (timer may or may not be set)
3. The timer is set for the WAIT command with no specified events.

A return code of CNM_OPINPUT_ON_WAIT will be generated if one of the events specified on the WAIT command is OPINPUT. GO may be entered alone or operator input may follow the GO. See "WAIT Command" on page 186 for more information on satisfying a WAIT.

Return Codes:

CNM_GOOD	0	Everything OK.
CNM_NO_STORAGE	24	Non-zero return code from the DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_MQS + X	1000 + X	Non-zero return code (X) from the DSIMQS macro. (See *NetView Customization: Using Assembler* for more information on the DSIMQS macro).
CNM_BAD_LCS + X	13000 + X	Non-zero return code (X) from the DSILCS macro. (See *NetView Customization: Using Assembler* for more information on the DSILCS macro).

Refer to *NetView Operation* for more information on the GO command.

QUEUE Command

The QUEUE command adds a text message to the operator input queue (OPERQ) of an HLL command processor or user exit routine running with the HLL QUEUED INPUT bit of HLLOPTS turned on.

Return Codes:

CNM_GOOD	0	Everything OK.
CNM_NO_STORAGE	24	Non-zero return code from the DSIGET macro.
CNM_BAD_MQS + X	1000 + X	Non-zero return code (X) from the DSIMQS macro.

Refer to *NetView Operation* for information about the QUEUE command.

RESET Command

High-level language support gives the user the option of specifying whether or not a command procedure is cancellable. Refer to the HLL run-time options (HLLOPTS) section of this manual for further detail.

If a command procedure is cancellable, it will behave according to the rules specified under the RESET command in the *NetView Operations* manual. The description for RESET NORMAL states that a command will be stopped at its next break point whenever RESET NORMAL is issued. A break point occurs in an HLL command procedure whenever an HLL service routine is invoked.

If a cancellable command procedure is reset via RESET NORMAL, it will terminate with a -5 return code, which will in turn, cancel its caller if the caller is also cancellable.

If the command procedure is non-cancellable, it will only be reset when RESET IMMED and RESET DUMP are issued. In the case where RESET NORMAL is issued, reset will behave as follows:

Whenever reset is issued, NetView turns on a "reset flag" which remains on until it is acted upon (a command procedure is reset). So, if RESET NORMAL is issued while a non-cancellable command procedure is running, the reset flag will remain on until the non-cancellable command procedure either calls or returns to a cancellable command procedure or uses the CNMINFI service routine to check RESETREQ.

A non-cancellable command procedure can check to see if an operator has attempted to reset it by using the RESETREQ function of the CNMINFI service routine. If a command procedure uses CNMINFI to check RESETREQ, the reset flag is set off and the decision to cancel becomes the responsibility of the command procedure checking RESETREQ. If the command procedure wishes to cancel itself, it must do so by returning with a -5 return code. If the command procedure terminates with a -5 return code, it will also cancel its caller if the caller is cancellable. If the caller is not cancellable, however, the caller will not be cancelled and the reset flag will not be set on. The reset flag is only set on as a result of RESET, LOGOFF, or CLOSE IMMED being issued.

Note: In the following examples, boxes represent command procedures, the words "can be cancelled" within the boxes means the command procedure is cancellable, and the words "cannot be cancelled" within the boxes means the command procedure is non-cancellable.

Examples:

In the following example, cancellable command procedure X calls cancellable command procedure Y which calls cancellable command procedure Z. RESET NORMAL is entered while command procedure Z is running.

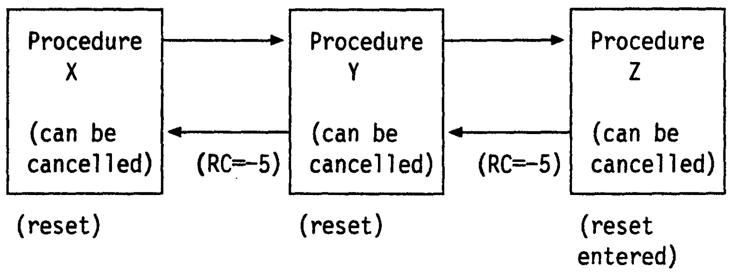


Figure 9. Example 1 of Command Procedure Cancelling

As a result of entering RESET NORMAL, Z is reset, Z returns -5 to Y, Y is reset, Y returns -5 to X, and X is reset.

Keep in mind that the HLL command procedures have to invoke HLL service routines in order for the reset bit to be checked. So, Y and X will continue executing if they do not invoke any more HLL service routines.

In the following example, cancellable command procedure X calls non-cancellable command procedure Y which calls cancellable command procedure Z. RESET is entered when command procedure Z is running.

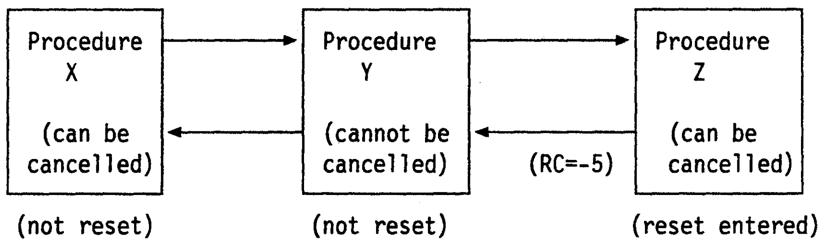


Figure 10. Example 2 of Command Procedure Cancelling

As a result of entering RESET NORMAL, Z is reset. Y receives a -5 return code from the call to Z.

In the following example, non-cancellable command procedure X calls non-cancellable command procedure Y, non-cancellable command procedure Y returns control to non-cancellable command procedure X, and non-cancellable command procedure X then calls cancellable command procedure Z. Reset is entered just as X begins execution.

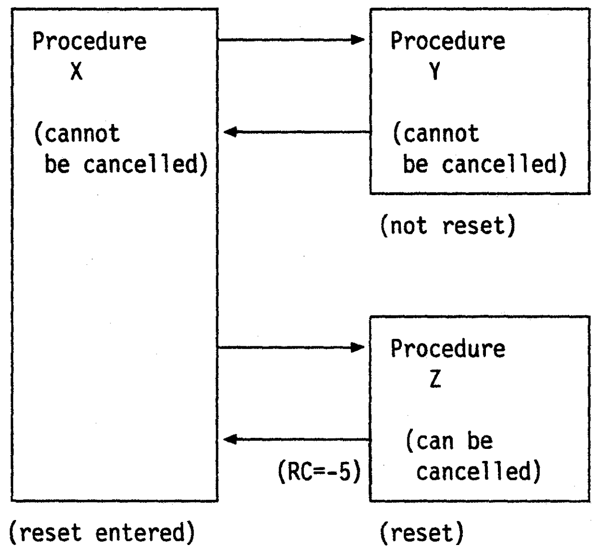


Figure 11. Example 3 of Command Procedure Cancelling

As a result of entering reset normal, X is not cancelled because it is non-cancellable, X goes ahead and calls Y, Y is not cancelled because Y is non-cancellable, Y returns control to X, and then X calls Z which is reset. Z returns control to X with a -5 return code. In this case, the reset flag was turned on when the user tried to reset X but was not turned off until it was acted upon. (Z was reset)

In the previous example, command procedure X could have checked the reset flag using the HLL service routine CNMINFI. If X had done this, the reset flag would have been turned off and Z would not have been reset.

IMPORTANT: It is recommended that you make your command procedures cancellable whenever possible. The non-cancellable option of high-level language support has been provided so that a user can code command procedures to do cleanup (such as free storage) before being cancelled.

Note: When a High-level language command procedure is cancelled, the clean-up done is equivalent to that done by STOP in PL/I and EXIT in C. See your PL/I or C manual to see what cleanup is done for you in these cases.

Recommended scenario when cleanup is needed:

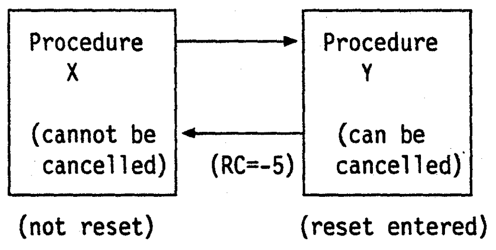


Figure 12. Example 4 of Command Procedure Cancelling

In this scenario, if reset is issued while Y is executing, Y will be terminated and X will stick around to do cleanup.

TRAP Command

The TRAP command lets the user specify message trapping criteria for HLL and REXX command procedures designed to trap messages. Once issued, all subsequent messages that match the conditions defined by the trapping criteria will be added to the message queue (TRAPQ). When used in conjunction with the WAIT FOR MESSAGES command and the CNMGETD service routine, the TRAP command allows the user to code command procedures to intercept and process (automate, display, suppress, etc.) certain messages. The message trapping criteria specified in the TRAP command defines the set of conditions that will satisfy subsequent WAIT FOR MESSAGES commands. TRAP can only be issued from command procedures written in a high-level language or REXX. All operands are order dependent.

```
TRAP [[AND]{SUPPRESS|DISPLAY}]
     [{MORE|ONLY}]
     MESSAGES [domainid1.]token1 [[domainid2.] token2...]

       OR

TRAP NO MESSAGES
```

Where:

AND

Can be used to make the TRAP command more readable. You can only use AND between TRAP and SUPPRESS or TRAP and DISPLAY.

SUPPRESS|DISPLAY

SUPPRESS	Indicates that any message matching a specified *token* should not be displayed on the operator's screen when received by NetView.
DISPLAY	Indicates that any message matching a specified *token* should be displayed on the operator's screen when received by NetView. DISPLAY is the default.

MORE|ONLY

MORE	Indicates that the specified *token*s should be added to the list of *token*s that was specified on a previous TRAP command. **Note:** Each message in the resulting list retains its own individual setting of the SUPPRESS|DISPLAY option. This will allow some messages in the list to be suppressed while others are displayed.
ONLY	Indicates that the specified *token*s replace the list of *token*s on a previous TRAP command. ONLY is the default.

MESSAGES

This is a required operand which indicates that the trapped items are messages. *domainid* (1 to 8 characters) is the domain ID of the message or messages to be trapped. *token* (1 to 10 characters) identifies the first token of the message or messages to be trapped. Most special characters are valid for *token*. However, the equal sign (=) is invalid for *token* and will result in a syntax error. There is no limit on the number of *token*s that can be specified.

Note: You may use a trailing asterisk (*) in the *domainid* or *token* as a wildcard character. Asterisk (*) can also be used as a character in a *token*.

For example, TRAP MESSAGES A*B* traps on *token*s beginning with A*B followed by any characters. If the (*) is followed by a character, then it is considered to be part of the *token*.

NO

Indicates that the list of *token*s that was specified on all previous TRAP commands should be removed. TRAP NO MESSAGES will clear the command procedure's list of messages to trap. No other operands are valid with NO.

Following are examples of how you can specify the messages you want to trap:

domainid.token	The command procedure traps any message whose domain identifier matches the 1 to 8 character *domainid*, and whose first token matches *token*.
dom.token*	The command procedure traps any message whose domain identifier matches the partial domain identifier specified by *dom** and whose first token matches *token*. For example, NCCF*.DSI604I means trap a DSI604I message from any domain with an identifier that starts with NCCF (such as NCCFA or NCCFB).
**.token*	The command procedure traps any message whose first token matches *token*. The message can be from any domain.
token	The command procedure traps any message whose first token matches *token*. (*domainid* is assigned the wildcard character (*)).
*tok**	The command procedure traps any message whose first token matches the partial token specified by *tok**. For example, DSI* means trap any messages whose first token begins with DSI, such as DSI604I or DSI028I.
	The command procedure traps all messages.

Usage Notes:

Each TRAP command without the MORE operand cancels and replaces the previous TRAP command. To add tokens to a previous TRAP command, use the MORE operand.

The issuance of a TRAP command does not clear the queue of messages trapped by the previous TRAP command. To clear the message queue, issue a FLUSHQ (See "CNMGETD (CNMGETDATA) — Data Queue Manipulation" on page 206).

The TRAP command must be issued from a command procedure running under an OST or NNT.

Immediate messages are not trapped; they will not be added to the message queue (TRAPQ).

TRAP AND SUPPRESS MESSAGES * is NOT the same as TRAP NO MESSAGES. TRAP AND SUPPRESS MESSAGES * traps all messages but does not display them on the operator's console.

Message trapping (TRAP command) takes precedence over message automation.

When trapping TAF messages, the domain ID is actually the session ID for that TAF session.

Return Codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_INVOCATION	4	TRAP was NOT issued from an HLL or REXX command procedure.
CNM_BAD_SYNTAX	12	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.
CNM_BAD_COMMAND	144	TRAP was NOT issued from a command procedure running under an OST or NNT.
CNM_BAD_MRBLD + X	18000 + X	Non-zero return code X. See values for X below.

Values for X:

	8	Request for storage has failed.

WAIT Command

When issued from a command procedure, the WAIT command will temporarily suspend processing of that command procedure until a specified event occurs. For an HLL command procedure, the event can be (1) one or more messages, (2) operator input, (3) data, (4) a certain period of time, or any combination of the four. The first occurrence of one of these events will satisfy the wait and processing will be resumed.

```
WAIT [n [{SECONDS|MINUTES}]]
     [[FOR] event[event...]]

           OR

WAIT CONTINUE
```

where:

n
The number of SECONDS or MINUTES that the command procedure waits for receipt of the messages specified on the TRAP command, receipt of operator input, or receipt of data before processing is resumed. The command procedure may also wait only for the specified time (no specified events) before processing is resumed. Valid ranges for *n* are 0 to 2,678,400 and 0 to 44,640 for SECONDS and MINUTES respectively.

SECONDS|MINUTES

SECONDS	The specified unit of time is seconds.
MINUTES	The specified unit of time is minutes.

FOR
Can be used to make the WAIT command more readable.

event
The event or events that the command procedure is waiting for.

MESSAGES	The command procedure waits for one or more messages before processing is resumed. The message criteria is specified using the TRAP command. See "TRAP Command" on page 183.
OPINPUT	The command procedure waits for operator input. Both the QUEUE and GO command queue data to the operator input queue. See "GO Command" and "QUEUE Command" on page 179 of this manual.
DATA	The command procedure waits for data sent by CNMSMSG with *smmsgtyp* = DATA. See "CNMSMSG (CNMSENDMSG) – Send Message or Command" on page 234.

CONTINUE
Continue waiting for additional messages, operator input, or data before command procedure processing is resumed.

Usage Notes:

In REXX, only messages are valid events.

The NetView operator's screen is refreshed whenever a message arrives or the enter key is pressed. If the screen is refreshed while an HLL command procedure is in a wait state, the pause and wait status indicators (P and W) are displayed in the upper right hand corner of the current command facility screen. The W indicator notifies the operator that the command procedure has halted its processing and is waiting for one or more messages, data, or a specified period of time. The P indicator notifies the operator that the command procedure has halted its processing and that one of the events it is waiting for is operator input. When the wait is satisfied, processing resumes and the indicators are cleared from the screen.

The operator input queue should always be flushed (FLUSHQ) after a WAIT command has been satisfied with operator input. Otherwise, subsequent WAIT FOR OPINPUT commands will not wait for operator input. See "CNMGETD (CNMGETDATA) – Data Queue Manipulation" on page 206 for more information on FLUSHQ.

The WAIT command can only be issued from an HLL or REXX command procedure running under an OST or NNT.

The WAIT command with no operands is invalid.

The following example illustrates the use of the TRAP and WAIT commands. If the initial WAIT command is satisfied after 5 seconds, the WAIT CONTINUE command will get control and the command procedure will continue to wait for the remainder of the time specified (15 seconds) in the previous WAIT COMMAND.

```
CNMCOMMAND DATA('TRAP MESSAGES MSG1 MSG2 MSG3');
           /* Trap messages with tokens MSG1, MSG2, MSG3          */
CNMCOMMAND DATA('WAIT 20 SECONDS FOR MESSAGES');
           /* Wait up to 20 seconds for the first trapped message */
CNMGETDATA FUNC(FLUSHQ) QUEUE(TRAPQ);
           /* Remove messages from the message queue              */
CNMCOMMAND DATA('WAIT CONTINUE');
           /* Continue waiting for the next trapped message       */
```

Unlike REXX, an HLL command procedure that issues the WAIT CONTINUE command can continue waiting for operator input and data. The following example illustrates this. If operator input is received within 12 seconds, the first wait will be satisfied and the command procedure will continue to wait for operator input for the remaining 18 seconds. The same is also true when waiting for data.

```
CNMCOMMAND DATA('WAIT 30 SECONDS FOR OPINPUT');
           /* Wait up to 30 seconds for operator input   */
CNMGETDATA FUNC(FLUSHQ) QUEUE(OPERQ);
           /* Remove the operator input from the queue   */
CNMCOMMAND DATA('WAIT CONTINUE');
           /* Continue waiting for more operator input   */
```

A WAIT CONTINUE command will satisfy the previous valid WAIT command. In the following example, the command procedure will continue to wait using the wait criteria from the initial WAIT command because the second WAIT command is invalid. If the initial WAIT command is satisfied after 3 seconds, the command procedure will continue to wait for messages for the remaining 7 seconds (since the WAIT 20 SECONDS FOR DATAA was invalid).

```
  CNMCOMMAND DATA('TRAP MESSAGES *');
             /* Trap all messages                                  */
  CNMCOMMAND DATA('WAIT 10 SECONDS FOR MESSAGES');
             /* Wait up to 10 seconds for the first trapped message */
  CNMGETDATA FUNC(FLUSHQ) QUEUE(TRAPQ);
             /* Remove messages from the message queue             */
  CNMCOMMAND DATA('WAIT 20 SECONDS FOR DATAA');
             /* Invalid wait for data (misspelled data)            */
  CNMGETDATA FUNC(FLUSHQ) QUEUE(DATAQ);
             /* Remove data from the data queue                    */
  CNMCOMMAND DATA('WAIT CONTINUE');
             /* Continue waiting for the next trapped message      */
```

Note: CNMGETD must complete successfully (HLBRC = 0) before a WAIT CONTINUE will process correctly.

Return Codes:

CNM_BAD_INVOCATION	4	WAIT was NOT issued from an HLL or REXX command procedure.
CNM_TOO_MANY	8	Too many operands.
CNM_BAD_SYNTAX	12	Syntax error.
CNM_BAD_COMMAND	144	WAIT was NOT issued from a command procedure running under an OST or NNT.
CNM_NO_TRAP	152	WAIT issued but TRAP was not issued. Must issue valid TRAP before issuing wait for messages.
CNM_NO_PREV_WAIT	248	No previous WAIT; WAIT CONTINUE invalid.

The following return codes are issued when a WAIT is satisfied. They are generated in place of a zero 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.

HLL Service Routine Reference

The following service routines may be invoked from a command processor or user 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 user exit routine in PL/I, the user may invoke an HLL service routine using the call or macro format. The PL/I macro format has been provided for those users that wish to code only the required parameters for a particular HLL service routine invocation. The user must code all of the parameters when using the PL/I call format or C invocation.

When invoking a service routine from an HLL command processor or user 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

CNMALTD enables the user to alter the contents of the top message on the initial data queue. Lines may be inserted, replaced or deleted.

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:

hlbptr
A four byte pointer field containing the address of the HLB control block.

adfunc
An eight byte character field which specifies the function to be performed. Required for all CNMALTD calls.

INSLINE	Insert a new line in the message in the specified queue. *adindex* specifies the line number that the new line will have after it has been inserted. The index value may be one greater than the number of lines currently in the message in order to add a line on the end. All parameters are required for this function.
REPLINE	Replace a line of the current message in the specified queue (if it exists). All parameters are required for this function.

DELLINE	Delete a line of the current message in the specified queue. The line specified by *adindex* (the index value) is physically removed from the queue. It is possible to 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 "CNMGETD (CNMGETDATA) – Data Queue Manipulation" on page 206). *adqueue* and *adindex* are required parameters for DELLINE. *adorigin* and *adbuf* are NOT required parameters for this function.

adbuf
A varying length character field containing the buffer to be inserted. Required with INSLINE and REPLINE but not used with DELLINE.

adorigin
A character field of fixed length n (where n > = 38) to contain an origin block. The user 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 user exit routine as an initial parameter. ORIG_BLOCK_LENGTH cannot be less than 38. Refer to DSIPORIG (Appendix A) or DSICORIG (Appendix C) for the PL/I and C mappings of an origin block. The user is responsible for updating the origin block (*adorigin*) to reflect changes made by CNMALTD. Required for INSLINE and REPLINE. Not required for DELLINE.

adqueue
A four 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 which invoked the HLL command processor through message automation or the message which drove DSIEX02A is on the initial data queue. Required for all functions.

adindex
A four byte integer field containing the number of the line of the message at the head of the queue to be manipulated. Required for all functions.

Usage Notes:

CNMALTD was primarily designed for use in DSIEX02A where it enables the user to alter messages before they are automated or displayed.

Reference "CNMSMSG (CNMSENDMSG) – Send Message or Command" on page 234 for the definitions of line types.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro.
CNM_BAD_FUNC	52	Invalid *adfunc* value.
CNM_BAD_QUEUE	72	Invalid *adqueue* value.
CNM_BAD_INDEX	76	Invalid *adindex* value.
CNM_QUEUE_EMPTY	80	The specified queue is empty.
CNM_BAD_ORIGBLOCK	84	Invalid value in ORIG_BLOCK_LENGTH.
CNM_BAD_LENGTH	88	The length of (*adbuf*) is greater than (>) 32729.
CNM_NOT_MLWTO	92	Message is not a multi-line message.
CNM_BAD_LINETYPE	96	Invalid line type. Must be C,D,E,L,F, or ' '.

CNMCELL (CNMSTRCELL) – Storage Cell

CNMCELL can be used to allocate and free storage cells from a previously allocated storage pool. The token obtained by CNMPOOL must first be passed to CNMCELL to identify the storage pool.

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
A four byte pointer field containing the address of the HLB control block.

pcfunc
An eight byte character field which specifies the function to be performed.

ALLOC	Allocate cell
FREE	Free cell

pctoken
A four byte integer field containing the token identifying the storage pool. Provided by caller for all functions (token returned from CNMPOOL).

pcstrptr
A four byte pointer field to contain the address of the cell. Returned to caller for ALLOC, provided by caller for FREE.

Usage Notes:

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.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro.
CNM_BAD_TOKEN	32	Invalid *pctoken*.
CNM_BAD_FUNC	52	Invalid *pcfunc*.
CNM_BAD_CLASS	112	Possible storage overlay. Report to IBM service.
CNM_BAD_ADDR	160	The storage pointed to by *pcstrptr* is not addressable.
CNM_NOT_IN_POOL	204	Cell not in storage pool.
CNM_BAD_CELL_ADDRESS	256	Address is not on valid cell boundary.
CNM_CELL_ALREADY_FREE	260	Cell has already been freed.

CNMCMD (CNMCOMMAND) – Execute NetView Commands

CNMCMD enables the user to execute a NetView command from an HLL command processor. If the called command is a long running command, then the caller is suspended until the long running command has completed. The caller regains control at the instruction following the CNMCMD invocation.

PL/I CALL FORMAT:
CALL CNMCMD(*hlbptr,cmdstr*)

PL/I MACRO FORMAT:
CNMCOMMAND DATA(*cmdstr*)

C INVOCATION:
void Cnmcmd(void **cmdstr*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

cmdstr
A varying length character field containing the NetView command (including its parameters) to be executed.

Usage Notes:

Commands are invoked with a HDRMTYPE of HDRTYPEC. This is consistent with the NetView command list language and REXX.

The return code from HLL command processors and NetView long running commands will be returned properly to HLL command processors through the CNMCMD 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') would allow the BROWSE screen to ROLL independently from the calling HLL command processor. See *NetView Operation* for more information on the CMD command.

CNMCMD will not process immediate commands (type=I in DSICMD). If an HLL command processor issues CNMCMD and the command to be executed is an immediate command, the command will fail with a return code of -3108. Refer to Example 2 in the Composite Return Code section of this chapter for an explanation of this return code.

A negative return code generated from CNMCMD indicates a failure in the CNMCMD service routine. Refer to the following list of return codes for further explanation. A positive return code generated from CNMCMD indicates a failure in the NetView command that was to be executed by CNMCMD. Refer to the return codes listed for the NetView command in *NetView Operation*. A (-5) return code generated from CNMCMD indicates that the NetView command currently executing was cancelled. In this case, it is recommended that the command processor do any necessary cleanup and exit setting of HLBRC to -5 to pass the RESET information to its caller. Refer to the "RESET Command" on page 180.

A (-1) return code generated from CNMCMD indicates an unexpected error in the called command procedure.

CNMCMD cannot be invoked from an HLL user exit routine.

CNMCMD cannot be invoked from an HLL command processor while holding a lock.

The NetView Service Point Command Service (SPCS) commands are not supported under the HLL API and must not be invoked by CNMCMD. See *NetView Customization: Writing Command Lists* for more information on SPCS commands.

Return codes:

CNM_GOOD	0	Everything OK.
−CNM_BAD_INVOCATION	−4	Not invoked from a command processor.
−CNM_NO_STORAGE	−24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
−CNM_BAD_LENGTH	−88	Command length is invalid.
−CNM_LOCKED	−208	CNMCMD issued while holding a lock.
X−CNM_BAD_EXCMS	X−3000	Non-zero return code X. See values for X below.
Z		Return code from executed command. See usage notes for return codes (-5) and (-1).

Values for X:

	−4	Non-zero return code, 4 (drop), from user exit.
	−100−Y	Non-zero retun code, Y, from DSICES macro. (See *NetView Customization: Using Assembler* for more information on the DSICES macro).
	−200−Y	Non-zero return code, Y, from DSILCS (CWB) macro. (See *NetView Customization: Using Assembler* for more information on the DSILCS macro).
	−300−Y	Non-zero return code, Y, from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
	−400−Y	From either DSIGET or DSILCS indicates storage failure. The user exit is not invoked. (See *NetView Customization: Using Assembler* for more information on the DSIGET or DSILCS macro).
	−500−Y	Non-zero return code, Y, from either DSIGET or DSILCS indicates storage failure. The user exit is not invoked. (See *NetView Customization: Using Assembler* for more information on the DSIGET or DSILCS macro).
	−600−Y	Non-zero return code, Y, from DSILCS (SWB) macro. (See *NetView Customization: Using Assembler* for more information on the DSILCS macro).

CNMCNMI (CNMI) – CNMI Access Under a DST

NetView's CNMI service enables HLL command processors (running under a DST with CNMI capability) to send and receive data across the CNMI. This service can be used in conjunction with CNMGETD to manipulate data on the CNMI solicited data queue (CNMIQ).

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:

hlbptr
A four byte pointer field containing the address of the HLB control block.

cnfunc
An eight byte character field which specifies the function to be performed. Required for all CNMCNMI calls.

SENDRESP
Send RU and expect only positive or negative response.

SENDRPLY
Send RU and expect reply RU or negative response.

cndata
A varying length character field containing the RU to be sent (beginning with an RH header). Required for all functions. RU length must be at least 3 bytes and no longer than 32,729 characters.

cndest
An eight byte character field which specifies the PU name. Required for all functions.

cntimout
A four byte integer field specifying the number of seconds to wait for reply/response. This is an optional parameter. If *cntimout* is not specified, the default is 0. If a timeout is specified, the RH header must indicate that the embedded NS RU solicits a reply. This will cause NetView to generate a PRID. (See *VTAM Programming*). For requests that generate multiple RU (chained) replies, *cntimout* only applies to the first RU in the chain.

Usage Notes:

CNMCNMI cannot be invoked 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.

CNMCNMI cannot be issued from an HLL user exit routine.

Responses to CNMI solicited data requests will be placed on the CNMI solicited data queue (CNMIQ).

The XITCI user exit routine will be invoked for both solicited and unsolicited data. Refer to Chapter 2 for more information on this exit. Also refer to Chapter 3 for a discussion on Unsolicited HLL Data Services Command Processors (DSCP). It is important to note that when the Unsolicited HLL DSCP receives control, the command buffer (CMDBUF) will contain the unsolicited data RU.

For more information on installing a DST, refer to Chapter 3 on page 21 in this manual.

Return Codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_INVOCATION	4	Not invoked from a command processor or not under a DST.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro.
CNM_BAD_RULENG	48	Invalid *cndata* length.
CNM_BAD_FUNC	52	Invalid *cnfunc*.
CNM_BAD_TIMEOUT	56	*cntimout* less than (<) 0.
CNM_NEED_PRID	60	Timeout specified but PRID not generated. The PRID generation bit in the RU must be set if a timeout is specified.
CNM_NEG_RESPONSE	64	Negative response received. (Sense code in HLBSENSE).
CNM_TIME_OUT	68	Timeout occured.
CNM_LOCKED	208	CNMI issued while holding a lock.
CNM_DST_FAILURE + X	2000 + X	Non-zero return code, X, which is the DSRB minor return code for solicited CNMI data. See *NetView Customization: Using Assembler*.
CNM_BAD_ZCSMS + (X * 100) + Y	20000 + (X * 100) + Y	Non-zero return code (major), X, and non-zero return code (minor), Y, from DSIZCSMS.

CNMCPYS (CNMCOPYSTR) – Copy Storage

CNMCPYS enables the user to copy storage from one address to another address. This service routine allows the copy operation to process without ABENDing if the source or destination is not addressable. However, the service routine will not protect you from overwriting storage if it is addressable.

PL/I CALL FORMAT:
CALL CNMCPYS (*hlbptr, csfrom, csto, cslen, cstype*)

PL/I MACRO FORMAT:
CNMCOPYSTR FROM(*csfrom*) TO(*csto*) LENG(*cslen*) COPYTYPE(*cstype*)

C INVOCATION:
void Cnmcpys(void **csfrom*, void **csto*, int *cslen*, char **cstype*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

csfrom
A four byte pointer field containing the address of the source data.

csto
A four byte pointer field to contain the address of the destination.

cslen
A four byte integer field containing the number of bytes of storage to be copied. Length = 0 to 16777215.

If the value specified by *cslen* is greater than the actual length of the specified *csto* buffer, a storage overlay could occur. Special care should be taken when deciding the value of *cslen*.

cstype
The type of copy to perform. Valid types follow:

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.

Usage Notes:

The length field of varying length buffers will not be set or altered by CNMCPYS.

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. This can be done by designating a variable as a pointer to a string, and then passing CNMCPYS the address of that pointer.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_LENGTH	88	*cslen* greater than (>) 16777215 or less than (<) 0. Copy not performed.
CNM_BAD_ADDR	160	The storage pointed to by *csto* is not addressable.
CNM_BAD_CSTYPE	252	Invalid *cstype*.
CNM_BAD_ESTAE	15000	Non-zero return code from ESTAE macro. Refer to *MVS Extended Architecture Supervisor Services and Macro Instructions* for more detail.

CNMGETA (CNMGETATTR) – Query Message Attributes

CNMGETA can be used to obtain attributes of messages on the initial data queue (IDATAQ). Values of attributes are returned to the user in character string form.

PL/I CALL FORMAT:
CALL CNMGETA(*hlbptr,ganame,gadata,gadatlen,gaqueue*)

PL/I MACRO FORMAT:
CNMGETATTR ITEM(*ganame*) DATA(*gadata*) LENG(*gadatlen*) QUEUE(*gaqueue*)

C INVOCATION:
void Cnmgeta(char **ganame*, void **gadata*, int *gadatlen*, int *gaqueue*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

ganame
An eight byte character field which specifies the attribute. Following is a list of valid attributes:

AREAID	Equivalent to &AREAID which provides a one-letter (A-Z) identifier for the area on the console screen that displays the message.
DESC	Equivalent to &DESC which provides the system descriptor codes in a binary series of on (1) and off (0) characters, representing the descriptor code bits in order.
JOBNAME	Equivalent to &JOBNAME which provides the 1 to 8 character MVS JOB name.
JOBNUM	Equivalent to &JOBNUM which provides the 8 character MVS JOB number.
MCSFLAG	Equivalent to &MCSFLAG which provides the system message flags in a binary series of on (1) and off (0) codes.
MSGTYP	Equivalent to &MSGTYP which provides the system message type as three consecutive binary characters.
REPLYID	Equivalent to &REPLYID which provides a three character reply identifier for WTOR command replies.
ROUTCDE	Equivalent to &ROUTCDE which provides the system routing codes in a binary series of on (1) and off (0) characters, representing the routing code bits in order.
SESSID	Equivalent to &SESSID which is the TAF session ID where the message originated.
SMSGID	Equivalent to &SMSGID which provides an 8 character value that identifies a particular instance of a message.
SYSCONID	Equivalent to &SYSCONID which provides the console number (in decimal) that receives the message.

SYSID	Equivalent to &SYSID which provides the identifier of the MVS system that sent the message.

gadata
A varying length character field to contain the resulting value for the specified attribute.

gadatlen
A four byte integer field containing the length of *gadata*. This is the maximum length of the area provided to receive the returned data. *gadatlen* is provided by the user.

If the value specified by *gadatlen* is less than the length of the data to be returned, the truncated data will be returned in *gadata* and a return code of CNM_DATA_TRUNC will be 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 will be 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. Special care should be taken when deciding the value of *gadatlen*.

gaqueue
A four byte integer field containing the number of the queue holding the message. Only attributes for the initial data queue (IDATAQ) are useful at this time.

Usage Notes:

The following NetView command list language variables are available in either the message buffer or the origin block. They are not accessible through CNMGETA.

&HDRMTYPE	Provides the 1 character NetView message type of the message and is equivalent to ORIG_MSG_TYPE.
&LINETYPE	Provides the multi-line write-to-operator (MLWTO) line type and is equivalent to ORIG_LINE_TYPE.
&MSGID	Is the message identifier of the message most recently received by NetView and is equivalent to ORIG_PROCESS.
&MSGORIGIN	Is the domain where the message most recently received by NetView orginated and is equivalent to ORIG_DOMAIN.
&MSGSTR	Is the message text of the message most recently received by NetView and is equivalent to the message string itself.

Note: Refer to *NetView Customization: Writing Command Lists* for further description of the NetView command list language variables.

Some of the attributes apply to all types of messages, while others apply to only certain types of messages. For example, JOBNAME is only meaningful for messages received from MVS via the subsystem interface.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_DATA_TRUNC	40	*gadatlen* was too small. Data truncated.
CNM_BAD_FUNC	52	Invalid *ganame*.
CNM_BAD_QUEUE	72	Invalid *gaqueue* value.
CNM_QUEUE_EMPTY	80	The specified queue is empty.
CNM_BAD_LENGTH	88	*gadatlen* less than (<) 0.
CNM_BAD_ADDR	160	The storage pointed to by *gadata* is not addressable.

CNMGETD (CNMGETDATA) – Data Queue Manipulation

CNMGETD can be used to manipulate input queues that may consist of single and multi-line messages.

PL/I CALL FORMAT:
CALL CNMGETD(*hlbptr,gdfunc,gdbuf,gdbuflen,gdorigin,gdqueue,gdindex*)

PL/I MACRO FORMAT:
CNMGETDATA FUNC(*gdfunc*) DATA(*gdbuf*) LENG(*gdbuflen*) ORIGIN(*gdorigin*)
QUEUE(*gdqueue*) LINE(*gdindex*)

C INVOCATION
void Cnmgetd(char **gdfunc*, void **gdbuf*, int *gdbuflen*, void **gdorigin*,
int *gdqueue*, int *gdindex*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

gdfunc
An eight byte character field which specifies the function to be performed.

GETMSG
Returns the first line of the next message in the specified queue. If one or more lines of a multi-line message has already been returned by GETLINE or GETMSG, the current GETMSG skips to the next message in the queue, discarding any skipped lines.

GETLINE
Returns the next line in the specified queue. This will cross message boundaries until the queue is empty. The user must check ORIG_LINE_TYPE to determine which line is last.

PEEKLINE
Returns a line from the message at the head of the specified queue whose line number is specified by *gdindex*. Flushed and gotten lines can be peeked unless the message was flushed or a line in a subsequent message was gotten.

FLUSHLIN
Skips over the next line of the specified queue. This will cross message boundaries until the queue is empty.

FLUSHMSG
Skips to the next message in the specified queue.

FLUSHQ
Discards all messages in the specified queue.

gdbuf
A varying length character field containing the buffer to be returned for GETMSG, GETLINE, and PEEKLINE. Required for GETLINE, GETMSG, PEEKLINE. Not required for FLUSHLIN, FLUSHMSG, FLUSHQ.

gdbuflen

A four byte integer field containing the length of *gdbuf*. This is the maximum length of the area provided to receive the returned message. *gdbuflen* is provided by the user. Required for GETMSG, GETLINE, PEEKLINE. Not required for FLUSHLIN, FLUSHMSG, FLUSHQ.

If the value specified by *gdbuflen* is less than the length of the message to be returned, the truncated message will be returned in *gdbuf* and a return code of CNM_DATA_TRUNC will be generated. The full length of the message that was truncated is stored in HLBLENG (*Hlbleng*).

Note: GET requests continue to advance the cursor independent of the truncation. A combination of PEEKLINE and FLUSHLIN|FLUSHMSG|FLUSHQ should be used when retrieving complete lines of unknown length.

If the value specified by *gdbuflen* is equal to or greater than the length of the message to be returned, and HLBRC (*Hlbrc*) = CNM_GOOD, the length of the returned message will be stored in HLBLENG (*Hlbleng*).

If the value specified by *gdbuflen* is greater than the length of the receiving data buffer (*gdbuf*), a storage overlay could occur. Special care should be taken when deciding the value of *gdbuflen.*

gdorigin

A character field of fixed length n (where n > = 38) to contain an origin block. The user must define an origin block (*gdorigin*) to be passed as a parameter to CNMGETD. This must be a separate structure from the origin block (ORIGBLCK) that was passed to the HLL command processor or user exit routine as an initial parameter. ORIG_BLOCK_LENGTH cannot be less than 38. Refer to DSIPORIG (Appendix A) and DSICORIG (Appendix C) for the PL/I and C mappings of an origin block. Required for GETLINE, GETMSG, PEEKLINE. Not required for FLUSHLIN, FLUSHMSG, FLUSHQ.

gdqueue

A four byte integer field containing the number (index) of the queue on which to perform the operation. Required for all functions. Following is a list of valid values:

TRAPQ	Queue 1	Message queue. Contains trapped messages. Refer to "TRAP Command" on page 183.
OPERQ	Queue 2	Operator input queue. Refer to "GO Command" and "QUEUE Command" on page 179.
DATAQ	Queue 3	Data queue. Contains data sent from another HLL command processor or user exit routine. Refer to "CNMSMSG (CNMSENDMSG) – Send Message or Command" on page 234.
IDATAQ	Queue 4	Initial data queue. Contains the full message which invoked the HLL command processor through message automation or the message which drove DSIEX02A.
CNMIQ	Queue 5	CNMI solicited data queue. Contains RU's solicited via the CNMI service routine. Chained RU's are treated like multi-line messages. Refer to "CNMCNMI (CNMI) – CNMI Access Under a DST" on page 199.

gdIndex
A four byte integer field containing the number (index) of the line of the message at the head of the queue to be manipulated. Required only for PEEKLINE.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_DATA_TRUNC	40	*gdbuflen* was too small. Data truncated.
CNM_BAD_FUNC	52	Invalid *gdfunc* value.
CNM_BAD_QUEUE	72	Invalid *gdqueue* value.
CNM_BAD_INDEX	76	Invalid *gdindex* value.
CNM_QUEUE_EMPTY	80	The specified queue is empty.
CNM_BAD_LENGTH	88	*gdbuflen* less than (<) 0.
CNM_BAD_ADDR	160	The storage pointed to by either *gdbuf* or *gdorigin* is not addressable.

CNMINFC (CNMINFOC) – Query NetView Character Information

CNMINFC allows you to obtain information about the current NetView environment. CNMINFC returns character data.

PL/I CALL FORMAT:
CALL CNMINFC (*hlbptr,icname,icdata,icdatlen*)

PL/I MACRO FORMAT:
CNMINFOC ITEM(*icname*) DATA(*icdata*) LENG(*icdatlen*)

C INVOCATION:
void Cnminfc(char **icname*, void **icdata*, int *icdatlen*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

icname
An eight byte character field which specifies the name of a variable. Following is a list of allowable names:

APPLID	Equivalent to &APPLID which is the NetView domain ID appended with a 3-character alphanumeric value assigned by NetView.
CLOCK	Current value returned by STCK instruction (not displayable).
DATE	Equivalent to &DATE which is the current date in the format *mm/dd/yy*.
DATETIME	Equivalent to &DATE followed by &TIME.
DOMAIN	Domain ID.
HCOPY	Equivalent to &HCOPY which is the name of the hardcopy log task started by the operator.
LU	Equivalent to &LU which is the logical unit name for the operator terminal.
NVVER	NetView version and release.
OPID	Equivalent to &OPID which is the ID of the operator issuing the call to CNMINFC.
OPSYSTEM	Equivalent to &OPSYSTEM which is a character string that indicates the operating system under which the HLL command processor or user exit routine is running.
PID	Process ID for this HLL command processor or user exit routine. Used for CNMSMSG with *smmsgtyp*=DATA (not displayable).
STARTIME	NetView start time (not displayable).

TASK	Equivalent to &TASK (OST|NNT|PPT|MNT|DST|OPT|HCT) which is the 3-character string depending on the task under which the HLL command processor or user exit routine is running.
TASKNAME	Name of the task under which the HLL command processor or user exit routine is running.
TIME	Equivalent to &TIME which is the CPU time in the format *hh:mm*.
VTAM	VTAM level.

Icdata
A varying length character field to contain the character data to be returned.

Icdatlen
A four byte integer field containing the length of *icdata*. This is the maximum length of the area provided to receive the returned data. *icdatlen* must be greater than zero and less than 32729 and is provided by the user.

If the value specified by *icdatlen* is less than the length of the data to be returned, the truncated data will be returned in *icdata* and a return code of CNM_DATA_TRUNC will be 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 will be 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. Special care should be taken when deciding the value of *icdatlen*.

Usage Notes:

Refer to *NetView Customization: Writing Command Lists* for a description of the NetView command list language variables.

CLOCK, PID, and STARTIME are eight character representations of the TOD-clock (time-of-day) value returned by the STCK instruction.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_DATA_TRUNC	40	*icdatlen* was too small. Data truncated.
CNM_BAD_FUNC	52	Invalid *icname*. Value unchanged.
CNM_BAD_LENGTH	88	*icdatlen* less than (<) 0.
CNM_BAD_ADDR	160	The storage pointed to be *icdata* is not addressable.

CNMINFI (CNMINFOI) – Query NetView Integer Information

CNMINFI enables the user to obtain information about the current NetView environment. CNMINFI returns integer data.

PL/I CALL FORMAT:
CALL CNMINFI(*hlbptr,iiname,iinumb*)

PL/I MACRO FORMAT:
CNMINFOI ITEM(*iiname*) DATA(*iinumb*)

C INVOCATION:
void Cnminfi(char **iiname*, int **iinumb*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

iiname
An eight byte character field which specifies the name of the variable. Following is a list of allowable names:

ABENDRTN*
If processing as a long running command ABEND routine, then return true. From DSITVB NetView control block (TVBABEND). This field is a *product-sensitive programming interface*.

ATTENDED*
If there is an operator associated with this task, then return true. An AUTOTASK is attended if it has been assigned to a console using the AUTOTASK command.

AWAITINP*
If waiting for operator input, then return true. From DSITVB NetView control block.

CLOSING*
If NetView is terminating, then return true. From DSIMVT NetView control block (MVTCLOSE). This field is a *product-sensitive programming interface*.

COLORS
Number of colors that can be displayed.

LOGOFRTN*
If processing as a long running command LOGOFF routine, then return true. From DSITVB NetView control block (TVBLOGOF). This field is a *product-sensitive programming interface*.

MVTUFLD
User field. From DSIMVT NetView control block (MVTUFLD). This field is a *product-sensitive programming interface*.

OPER3270*
If OST with a 327x display terminal attached, then return true.

RESETREQ*

If RESET or CANCEL was requested, then return true. From DSITVB NetView control block (TVBRESET). TVBRESET will be turned off as a result of this query. This field is a *product-sensitive programming interface*. Refer to "RESET Command" on page 180 for further detail.

SCRNSER

Return serial number of the screen update. From DSITIB NetView control block (TIBSCRSN). This field is a *product-sensitive programming interface*.

USEREXIT

If the integer value is 0, the environment is that of a command processor.

If the integer value is 2 through 15, the environment is that of a user exit.

If the integer value is one of the following, the environment is that of a user exit running under a DST. Following is a list of values of the user exits running under a DST.

USERDINT	**233**	DSM Initialization Exit
USERVINT	**234**	VSAM Initialization Exit
USERVINP	**235**	VSAM Input Exit
USERVOUT	**236**	VSAM Output Exit
USERCINP	**237**	CNMI Input Exit
USERCOUT	**238**	CNMI Output Exit
USERXLOG	**240**	External Log Exit
USERBINT	**241**	Sequential Log Initialization Exit
USERBOUT	**242**	Sequential Log Output Exit

Refer to DSIPCONS (Appendix A) or DSICCONS (Appendix C) for a list of constants useful when coding user exit routines.

Note: The *iinames* followed by an (*) contain Boolean values. 0 = False and 1=True.

iinumb

A four byte integer field to contain the integer value returned.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_FUNC	52	Invalid *iiname*. Value unchanged.
CNM_BAD_ADDR	160	The storage pointed to by *iinumb* is not addressable.

CNMKIO (CNMKEYIO) – Keyed File Access Under a DST

CNMKIO provides access to DST managed key-sequenced VSAM files from an HLL command processor. This service routine will only perform its function when invoked from an HLL command processor running under a DST task. Calls from other environments will be rejected.

PL/I CALL FORMAT:
CALL CNMKIO(*hlbptr,vsfunc,vsdata,vsdatlen,vskey,vsoption*)

PL/I MACRO FORMAT:
```
CNMKEYIO FUNC(vsfunc) DATA(vsdata) LENG(vsdatlen) KEY(vskey)
        OPTIONS(vsoption)
```

C INVOCATION:
```
void Cnmkio(char *vsfunc, void *vsdata, int vsdatlen, void *vskey,
            char *vsoption)
```

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

vsfunc
An eight byte character field which specifies the function to performed.

GET_EQ	Get record equal to key. **Note:** The key field must match exactly, including blanks.
GET_EH	Get record equal to or higher than key.
GET_NEXT	Get next record in ascending sequence.
GET_PREV	Get next record in descending sequence.
PUT	Write/rewrite record.
ERASE	Erase record.
ENDREQ	Cancel a request for update.

vsdata
A varying length character field to contain the buffer to be returned/written. Provided by the user for PUT, returned for GET, and is not required for ERASE or ENDREQ.

vsdatlen
A four byte integer field containing the length of *vsdata*. This is the maximum length of the area provided to receive the returned data. *vsdatlen* is provided by the user. Required only for GET.

If the value specified by *vsdatlen* is less than the length of the data to be returned, the truncated data will be returned in *vsdata* and a return code of CNM_DATA_TRUNC will be 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 will be 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. Special care should be taken when deciding the value of *vsdatlen*.

vskey
A varying length character field containing the VSAM key used for access to the requested data. Required for GET_EQ, GET_EH or ERASE/DIRECT. Not required for GET_NEXT, GET_PREV, PUT, ERASE/UPDATE, or ENDREQ.

vsoption
An eight byte character field that specifies the type of access to the file. Provided by user for all functions except ENDREQ.

UPDATE	Get record for update or replace. Erase record that was gotten for update. PUT/UPDATE and ERASE/UPDATE must be preceded by a successful GET/UPDATE. UPDATE can only be used with GET, PUT, and ERASE.
NOUPDATE	Record will not be updated. NOUPDATE can only be used with GET.
DIRECT	Put a new record directly to the file or erase a record directly from the file (without invoking GET first). PUT/DIRECT can only be used for a new record. If the record already exists, CNM_DUPL_KEY will be 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 will be returned. DIRECT can only be used with PUT and ERASE.

Usage Notes:

CNMKIO cannot be issued from an HLL user exit routine.

CNMKIO cannot be invoked from an HLL command processor while holding a lock.

For more information on VSAM files, refer to *VSAM Programming.*

For more information on installing a DST, refer to Chapter 3 on page 21 of this manual.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_INVOCATION	4	Not invoked from a command processor.
CNM_NOT_FOUND	20	Record with requested *vskey* not found.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro.
CNM_END_FILE	36	End of file encountered.
CNM_DATA_TRUNC	40	*vsdatlen* was too small. Data truncated.
CNM_BAD_FUNC	52	Invalid *vsfunc*.
CNM_BAD_LENGTH	88	*vsdatlen* less than (<) 0.
CNM_BAD_OPTION	128	Invalid *vsoption*.
CNM_BAD_ADDR	160	The storage pointed to by *vsdata* is not addressable.
CNM_BAD_COMBO	176	Invalid combination *vsfunc* and *vsoption*.
CNM_DUPL_KEY	200	Record with requested key already exists. Existing record is not changed.
CNM_LOCKED	208	CNMKIO issued while holding a lock.
(CNM_BAD_ZVSMS + X) * 256 + Y	(100 + X) * 256 + Y	Non-zero return code from DSIZVSMS. X is the major return code from DSIZVSMS. Y is the minor return code from DSIZVSMS.
CNM_DST_FAILURE + X	2000 + X	Non-zero return code, X, which is the DSRB minor return code for solicited CNMI data. See *NetView Customization: Using Assembler*.

CNMLK (CNMLOCK) – Control A Lock

CNMLK can be used to obtain, release, and test the control of a named lock. This service can be used to serialize access to resources shared by multiple tasks. CNMLK does not allow for serialization within a task. HLL command processors holding a lock may not use any services that can cause execution of an HLL command processor to be suspended.

PL/I CALL FORMAT:
CALL CNMLK(*hlbptr,lkfunc,lkname,lkscope,lkoption*)

PL/I MACRO FORMAT:
CNMLOCK FUNC(*lkfunc*) NAME(*lkname*) SCOPE(*lkscope*) OPTION(*lkoption*)

C INVOCATION:
void Cnmlk(char **lkfunc*, void **lkname*, char **lkscope*, char **lkoption*)

Where:

hlbptr
A four byte pointer field containing the HLB control block.

lkfunc
An eight byte character field which specifies the function to be performed.

UNLOCK	Release control of lock name.
LOCK	Obtain control of lock name.
TEST	Test if lock name is available.

lkname
A varying length character field to hold the user-defined name of lock. (Length is 1-12 characters). Required for all functions.

lkscope
An eight byte character field. This field is reserved for future use. The user should provide a null or blank value for all functions.

lkoption
An eight byte character field which specifies if the HLL command processor or user exit routine should wait for the lock to become available. Required only for LOCK.

WAIT	Wait until the lock is available. The task will be suspended.
NOWAIT	Do not wait if LOCK is not available. An appropriate return code will be issued (CNM_LOCKED or CNM_LOCK_INUSE).

Usage Notes:

It is not recommended to invoke CNMSMSG with *smdestyp* = OPER while holding a lock. There is a possibility that the operator task may be running with autowrap off and the HLL command processor or user exit routine might hang waiting for the operator to clear the screen, thus holding the lock for an indefinite period of time.

A hierarchical order on lock requests is used to prevent deadlock. The alphabetical order of the lock names defines the hierarchy. For example, assume the last lock request was for *lkname* = GVARIABLE. A new lock request for *lkname* = TVARIABLE will be successful since TVARIABLE is alphabetically greater than GVARIABLE. However, a new lock request for *lkname* = CVARIABLE will be unsuccessful since CVARIABLE is alphabetically less than GVARIABLE. Return code CNM_LOCKED will be generated and the lock request will fail.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_FUNC	52	Invalid *lkfunc*.
CNM_BAD_NAME	108	Length of *lkname* greater than (>) 12 or the specified *lkname* was not LOCKed.
CNM_BAD_OPTION	128	Invalid *lkoption*.
CNM_LOCKED	208	A previous request for control of *lkname* has been made for the same task. The task has control of the specified *lkname*.
CNM_LOCK_INUSE	212	*lkname* is not available; currently held by another task.
CNM_BAD_ENQ + X	21000 + X	Non-zero return code X from DSIENQ macro. See *MVS Extended Architecture Supervisor Services and Macro Instructions* for values for X.

CNMMEMO (CNMOPENMEM) – Open NetView Partitioned Data Set

CNMMEMO enables the user to open members of NetView partitioned data sets. A token identifying the open member is returned to the user. This token is passed to CNMMEMR to read records from the member and to CNMMEMC to close the member.

PL/I CALL FORMAT:
CALL CNMMEMO(*hlbptr,motoken,moddname,momemnam*)

PL/I MACRO FORMAT:
```
CNMOPENMEM TOKEN(motoken) DATASET(moddname)
           MEMBER(momemnam)
```

C INVOCATION:
void Cnmmemo(int **motoken*, char **moddname*, char **momemnam*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

motoken
A four byte integer field to contain the token to be used by subsequent CNMMEMR and CNMMEMC requests.

moddname
An eight byte character field which specifies the DD name of the partitioned data set. The NetView predefined partitioned data sets are:

- BNJMISC
- BNJPNL1
- BNJPNL2
- CNMPNL1
- DSICLD
- DSIMSG
- DSIPARM
- DSIPRF
- DSIVTAM

momemnam
An eight byte character field which specifies the name of the member.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_DDNAME	16	Invalid *moddname* (not found).
CNM_NOT_FOUND	20	*momemnam* not found.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_ADDR	160	The storage pointed to by *motoken* is not addressable.
CNM_BAD_DKS + X	10000 + X	Non-zero return code, X, from DSIDKS macro. (See *NetView Customization: Using Assembler* for more information on the DSIDKS macro).

CNMMEMR (CNMREADMEM) – Read NetView Partitioned Data Set

CNMMEMR allows 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 the read.

PL/I CALL FORMAT:
CALL CNMMEMR(*hlbptr,mrtoken,mrdata,mrdatlen*)

PL/I MACRO FORMAT:
CNMREADMEM TOKEN(*mrtoken*) DATA(*mrdata*) LENG(*mrdatlen*)

C INVOCATION:
void Cnmmemr(int *mrtoken*, void **mrdata*, int *mrdatlen*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

mrtoken
A four byte integer field containing the token returned by CNMMEMO.

mrdata
A varying length character field to contain the received record.

mrdatlen
A four byte integer field containing the length of *mrdata*. This is the maximum length of the area provided to receive the returned record. *mrdatlen* is provided by the user.

If the value specified by *mrdatlen* is less than the length of the record to be returned, the truncated record will be returned in *mrdata* and a return code of CNM_DATA_TRUNC will be generated. The full length of the record that was truncated 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 will be 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. Special care should be taken when deciding the value of *mrdatlen*.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_IOERROR	28	I/O error occurred.
CNM_BAD_TOKEN	32	Token not found.
CNM_END_FILE	36	End of file encountered.
CNM_DATA_TRUNC	40	*mrdatlen* was too small. Data truncated.
CNM_BAD_LENGTH	88	*mrdatlen* less than (<) 0.
CNM_BAD_ADDR	160	The storage pointed to by *mrdata* is not addressable.
CNM_BAD_DKS + X	10000 + X	Non-zero return code, X, from DSIDKS macro. (See *NetView Customization: Using Assembler* for more information on the DSIDKS macro).

CNMMEMC (CNMCLOSMEM) – Close NetView Partitioned Data Set

CNMMEMC enables the user 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 close. All members opened by CNMMEMO will automatically be closed at program termination.

PL/I CALL FORMAT:
CALL CNMMEMC(*hlbptr,mctoken*)

PL/I MACRO FORMAT:
CNMCLOSMEM TOKEN(*mctoken*)

C INVOCATION:
void Cnmmemc(int *mctoken*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

mctoken
A four byte integer field containing the token returned by CNMMEMO.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_TOKEN	32	Token not found.
CNM_BAD_DKS + X	10000 + X	Non-zero return code, X, from DSIDKS macro. (See *NetView Customization: Using Assembler* for more information on the DSIDKS macro).

CNMNAMS (CNMNAMESTR) — Named Storage

CNMNAMS allows the user to allocate, free, locate, and reallocate named areas of virtual storage.

PL/I CALL FORMAT:

```
CALL CNMNAMS(hlbptr,nsfunc,nsptr,nsname,nsleng,nsclass)
```

PL/I MACRO FORMAT:

```
CNMNAMESTR FUNC(nsfunc) STRPTR(nsptr) NAME(nsname) LENG(nsleng)
           CLASS(nsclass)
```

C INVOCATION:

```
void Cnmnams(char *nsfunc, void *nsptr, void *nsname, int *nsleng,
           int nsclass)
```

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

nsfunc
An eight byte character field which specifies the function to be performed:

ALLOC	Allocate named storage area.
FREE	Free/deallocate named storage area.
LOCATE	Locate existing named storage area.
REALLOC	Reallocate named storage area. Old data is preserved.

nsptr
A four byte pointer field containing the address of the named storage. Returned to caller for ALLOC, REALLOC, and LOCATE. Not required for FREE.

nsname
A varying length character field to contain the name of the storage area. Required for all functions (provided by the caller).

nsleng
A four byte integer field containing the size of the named storage area. Required by caller for ALLOC and REALLOC. Returned to caller for LOCATE. Not required for FREE.

nsclass
A four byte integer field containing the class of the named storage area. Required by caller for ALLOC and REALLOC. Not required for FREE or LOCATE.

0 =	residency of caller
1 =	31 bit storage
2 =	24 bit storage

Usage Notes:

Named storage areas provide a way of sharing data among different HLL command processors and user exit routines or among multiple invocations of a HLL command processor or user 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.

A named storage area is associated with the NetView subtask under which it was allocated. Named storage areas can only be manipulated (locate, free, reallocate) by HLL command processors and user 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.

If ALLOC is requested for a name that has already been allocated, the address of the existing area is returned along with a non-zero return code.

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 preserved up to the length specified by the *nsleng* parameter.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NOT_FOUND	20	REALLOC, FREE, or LOCATE requested but no previous ALLOC was done.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_FUNC	52	Invalid function.
CNM_BAD_LENGTH	88	*nsleng* less than (<) 0.
CNM_DUPL_NAME	104	*nsname* already allocated. Allocation not done.
CNM_BAD_NAME	108	Length of *nsname* greater than (>) 12.
CNM_BAD_CLASS	112	Invalid *nsclass*.
CNM_BAD_ADDR	160	The storage pointed to by *nsname* or *nsleng* is not addressable.
CNM_BAD_PUSH + X	4000 + X	Non-zero return code, X, from DSIPUSH macro. (See *NetView Customization: Using Assembler* for more information on the DSIPUSH macro).
CNM_BAD_POP+X	5000 + X	Non-zero return code, X, from DSIPOP macro. (See *NetView Customization: Using Assembler* for more information on the DSIPOP macro).

CNMPOOL (CNMSTRPOOL) – Storage Pool

CNMPOOL can be used to allocate, free, and locate storage pools. A storage pool is composed of one primary and zero or more secondary blocks of storage. Each block of storage consists of a specified number of cells (of equal size) which can be allocated/freed using CNMCELL. Storage pool services provide a way to effectively manage large numbers of fixed size storage elements.

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*) LENG(*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*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

spfunc
An eight byte character field which specifies the function to be performed.

ALLOC	Allocate pool
FREE	Free pool
LOCATE	Locate pool

sptoken
A four byte integer field to contain the token identifying the storage pool. Returned for ALLOC and LOCATE for use with CNMCELL service. Not required for FREE.

spname
A varying length character field containing the name of the storage pool. Required for all functions and provided by the caller.

spleng
A four byte integer field containing the size of each cell in the pool. Required for ALLOC. Not required for FREE or LOCATE.

sppricnt
A four byte integer field containing the number of cells in the primary block. Required only for ALLOC.

spseccnt
A four byte integer field containing the number of cells in the secondary block. Required only for ALLOC.

spclass
A four byte integer field containing the storage class of the pool. Required for ALLOC. Not required for FREE or LOCATE.

0 =	residency of caller
1 =	31 bit addressable
2 =	24 bit addressable

Usage Notes:

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.

All storage pool names must be unique within a given task.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NOT_FOUND	20	*spname* not found.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_FUNC	52	Invalid *spfunc.*
CNM_BAD_LENGTH	88	*spleng* less than (<) 4.
CNM_DUPL_NAME	104	*spname* already allocated. Allocation not done.
CNM_BAD_NAME	108	Length of *spname* greater than (>) 12.
CNM_BAD_CLASS	112	Invalid *spclass.*
CNM_BAD_ADDR	160	The storage pointed to by *sptoken* is not addressable.
CNM_BAD_PRI_COUNT	192	Invalid *sppricnt.*
CNM_BAD_SEC_COUNT	196	Invalid *spseccnt.*

CNMSCAN (CNMSSCAN) – Parse and Convert a Character String—(PL/I)

CNMSCAN can be used 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 of the receiving variables must have the same data type as its corresponding type specifier in the format string. The user can specify up to ten receiving variables in the argument list. The number of fields successfully parsed and converted is returned to the user 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 continues until all of the format specifications in the format string have been processed.

PL/I CALL FORMAT:
CALL CNMSCAN(*hlbptr,pastring,pattern,panumfld,pafield1,...,pafield10*)

PL/I MACRO FORMAT:
```
CNMSSCAN DATA(pastring) FORMAT(pattern) COUNT(panumfld)
         P1(pafield1),...,P10(pafield10)
```

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

pastring
A varying length character field containing the input string to be parsed and converted.

pattern
A varying length character field containing the format specifications. The format string (*pattern*) determines how the data elements in the input string (*pastring*) will be parsed and converted.

panumfld
A four byte integer field to contain the number of fields successfully parsed and converted. This field is returned to the user.

pafield1,...,pafield10
List of receiving variables. The last variable named in this list will receive the value of the last input field parsed and converted according to the last specification in the format string. Each of the variables named in this list must be declared to have the same data type as its corresponding type specifier in the format string. The user can specify up to ten variables to receive parsed and converted data.

Usage Notes:

The format string consists of a series of format specifications which are defined as follows:

- The character % that designates the beginning of each format specification. (Required for each format specification).
- The character * that indicates the data in the input string for this format specification should be read from the input string but not assigned 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 full-word) that indicates the size of the argument that the value of the parsed or converted input data will be assigned. (Optional)
- Any number of white space characters can be interspersed within or between format specifications for readability. However, blanks should not be inserted between { } unless this is the desired effect. (Optional)
- The type specifier that specifies the data type of the parsed or converted input data to be stored in the receiving variable. (Required)

The type specifier directs the conversion of the next input field. CNMSCAN places the result in the receiving variable, unless assignment suppression was specified 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 doesn't meet the criteria of the type specifier or until the width of the field, if specified, is reached.

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. The following type specifiers are valid:

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 will retrieve the next three characters of the input string. To skip over spaces before obtaining a character, use %1s. Refer to the discussion below 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 decimal value. Input is an optionally signed sequence of decimal digits. Any spaces in the input string preceding the decimal digits will be skipped. The decimal digits are delimited by the next non-decimal 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 has been reached, the value stored will be zero.

s Expect a character string. Any spaces in the input string preceding the character string will be skipped. The character string is delimited by a space. If a field width is not specified, the field width will default 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 will be skipped. The decimal digits are delimited by the next non-decimal 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 will be skipped. The hexidecimal digits are delimited by the next non-hexadecimal character in the input string.

{} Expect a string that is not delimited by spaces. Follow the { with a set of characters followed by a }. The corresponding input field is read to the first character that does NOT appear between {}. If the first character is a ¬ (or '5f'x), the effect is reversed; the input field is read to the first character that appears between {}.

{¬} parse until the end of the string.

{¬**a**} parse until the character 'a' is found.

{**a**} parse until any character other than an 'a' is found.

{} parse until any character is found.

Note: The receiving variable to contain the character string result must be declared as a varying length character string.

If your format string specifies more data elements to be parsed and converted than your input string contains, the results are unpredictable.

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.

CNMSCAN returns when it encounters a format specification it does not expect or when it reaches the end of the input string.

If CNMSCAN is invoked using the PL/I call format and all ten *pafields* are not specified, a warning message will be issued at compile time.

CNMSCAN can only be used in an HLL command processor or user exit routine written in PL/I.

The PARSEL2R command provides a function similar to CNMSCAN and remains available for use. However, because of its conversion capabilities, CNMSCAN is more suitable to HLL command processors.

Return Codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_ADDR	160	The storage pointed to by *panumfld* or *pafield1,...,pafield10* is not addressable.

CNMSCOP (CNMSCOPECK) – Scope Check for Security

CNMSCOP determines if the user is authorized to issue a specific command/keyword/value combination from a particular operator ID. The scope check is based on the scope of authority of the operator ID of the task executing CNMSCOP. Only NetView scope checking is performed. No attempt is made to determine if a resource is in the task (operator's) span of control.

PL/I CALL FORMAT:
CALL CNMSCOP(*hlbptr,sccmd,sckwd,scvalue*)

PL/I MACRO FORMAT:
CNMSCOPECK VERB(*sccmd*) KEYWORD(*sckwd*) VALUE(*scvalue*)

C INVOCATION:
void Cnmscop(char **sccmd*, char **sckwd*, char **scvalue*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

sccmd
An eight byte character field which specifies the verb of the command to be scope checked. Blanks or (*) imply that the command verb that invoked the HLL command processor will be used.

sckwd
An eight byte character field which specifies the keyword of the command to scope check. Blanks or (*) imply that no specific keyword is checked.

scvalue
An eight byte character field which specifies the value of *sckwd* to be scope checked. Blanks or (*) imply that no specific keyword value is to be checked.

Usage Notes:

If CNMSCOP is issued from an HLL user exit routine, *sccmd* must be specified. Otherwise, return code CNM_BAD_COMMAND will be generated.

Scope of commands, keywords, and keyword values is defined in DSICMD. Operator class is defined in each operator's profile. For further information see *NetView: Installation and Administration Guide*.

CNMSCOP will not check the validity of a command. It should only be used to verify if an operator has authorization to issue a particular command.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_COMMAND_NA	132	*sccmd* not authorized.
CNM_KEYWORD_NA	136	*sckwd* not authorized.
CNM_VALUE_NA	140	*scvalue* not authorized.
CNM_BAD_COMMAND	144	Invalid syntax of *sccmd* or *sccmd* was not found. Check for length greater than (>) 8 or invalid characters in *sccmd*. *sccmd* may be incorrectly defined in DSICMD. See usage notes if CNMSCOP was invoked from an HLL user exit routine.
CNM_BAD_KEYWORD	148	VALUE (*scvalue*) was specified without a keyword (*sckwd*). *sckwd* must be specified when *scvalue* is specified.
CNM_BAD_CES + X	9000 + X	Non-zero return code, X from DSICES macro. (See *NetView Customization: Using Assembler* for more information on the DSICES macro).
CNM_BAD_KVS + X	11000 + X	Non-zero return code, X from DSIKVS macro. (See *NetView Customization: Using Assembler* for more information on the DSIKVS macro).

CNMSMSG (CNMSENDMSG) — Send Message or Command

CNMSMSG enables the user to send a message or command to specific destinations in your network.

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 **smmsgtype*, char **smdestyp*,
char **smdestid*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

smtext
A varying length character field containing the message text. Required parameter.

- If *smmsgtyp* is MSG, MSG_C, MSG_L, MSG_D, MSG_E, or MSG_F then *smtext* is the text of the message.
- If *smmsgtyp* is COMMAND then *smtext* is either a command procedure name or a NetView command.
- If *smmsgtyp* is REQUEST then *smtext* is the name of the HLL command processor that should send the data.
- If *smmsgtyp* is DATA then the process ID must be concatenated with the data. *smtext* must be specified as follows:

```
smtext = ORIGIN->ORIG_PROCESS||'text'
```

smmsgtyp
An eight byte character field which specifies the message type. Required parameter. Values are:

MSG	Single line message. ORIG_LINE_TYPE=' '.
MSG_C	Control line message. ORIG_LINE_TYPE='C'.
MSG_L	Label line message. ORIG_LINE_TYPE='L'.
MSG_D	Data line message. ORIG_LINE_TYPE='D'.
MSG_E	End of multi-line message. ORIG_LINE_TYPE='E'.
MSG_F	MSG_D and MSG_E combined. ORIG_LINE_TYPE='F'.
COMMAND	Command to be executed. **Note:** The command is asynchronously scheduled for execution.

REQUEST	Request for data.
	Note: 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.
DATA	Non-printable 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 user exit routine if there are multiple activations of the same HLL command processor or user exit routine. The data returned from CNMSMSG with *smmsgtyp* = DATA can be read by CNMGETD from the data queue (DATAQ).

smdestyp

An eight byte character field which specifies the destination type. Required parameter. Values are:

OPER	Operator task invoking this service routine
TASK	Another task
SYSOP	System console
NETVLOG	NetView log
EXTLOG	External log (e.g. SMF)
SEQLOG	Sequential log
AUTHRCV	Authorized receiver
OPCLASS	All operators in group

smdestid

Destination ID. Required when *smdestyp* is EXTLOG, SEQLOG, TASK or OPCLASS.

When *smdestyp* is EXTLOG, SEQLOG or TASK, *smdestid* is the name of the destination task. An asterisk (*) may be used 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 Operation* manual for further detail.

Note: PPT is accepted as a synonym for the PPT task (xxxxxPPT) where xxxxx is the domain ID in the local NetView.

The following table shows message and destination type combinations which are not allowed:

Table 2. Invalid Message and Destination Type Combinations

	OPER	TASK	SYSOP	NETVLOG	EXTLOG	SEQLOG	AUTHRCV	OPCLASS
MSG								
MSG_C			x		x	x		
MSG_L			x		x	x		
MSG_D			x		x	x		
MSG_E			x		x	x		
MSG_F			x		x	x		
COMMAND	x		x	x	x	x	x	x
REQUEST	x		x	x	x	x	x	x
DATA	x		x	x	x	x	x	x

Usage Notes:

Approximately 32K of data may be sent by CNMSMSG.

CNMSMSG will generate a return code of CNM_BAD_INVOCATION when invoked from DSIEX04 or DSIEX09. CNMSMSG can be invoked from DSIEX02A if *smdestyp* is TASK. All other invocations of CNMSMSG from DSIEX02A will generate a return code of CNM_BAD_INVOCATION.

The EXCMD command queues a NetView command to another task where it is executed. CNMSMSG also allows the user to send a command to another task in a similar way by specifying *smmsgtyp*=COMMAND, *smdestyp*=TASK, and the desired *smtext* and *smdestid*.

Messages sent to a console with *smmsgtyp* = MSG_C, MSG_D, MSG_E, MSG_F or MSG_L will be truncated if they are longer than the screen width of that console.

The user 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 will be held on the screen if the data lines for that multi-line message cause the screen to wrap.

It is not recommended to invoke CNMSMSG with *smdestyp* = OPER while holding a lock. There is a possibility that the operator task may be running with autowrap off and the HLL command processor or user exit routine might hang waiting for the operator to clear the screen, thus holding the lock for an indefinite period of time.

Return codes:

CNM_GOOD	0	Everything OK.
CNM_BAD_INVOCATION	4	Not invoked from an allowed user exit.
CNM_NO_STORAGE	24	Non-zero return code from DSIGET macro. (See *NetView Customization: Using Assembler* for more information on the DSIGET macro).
CNM_BAD_LENGTH	88	*smtext* is too long.
CNM_BAD_MSGTYP	116	Invalid message type.
CNM_BAD_DESTYPE	120	Invalid destination type.
CNM_TYP_CONFLICT	124	Conflict between message and destination type.
CNM_LOG_INACTIVE	216	WLS failure--log was inactive.
CNM_TASK_INACTIVE	220	MQS failure--task was inactive.
CNM_BAD_MQS + X	1000 + X	Bad return code, X, from DSIMQS message queue macro. (See *NetView Customization: Using Assembler* for more information on the DSIMQS macro).
CNM_BAD_WLS + X	6000 + X	Bad return code, X, from DSIWLS write to log macro. (See *NetView Customization: Using Assembler* for more information on the DSIWLS macro).
CNM_BAD_PSS + X	7000 + X	Bad return code, X, from DSIPSS presentation services macro. (See *NetView Customization: Using Assembler* for more information on the DSIPSS macro).
CNM_BAD_WTO + X	8000 + X	Bad return code, X, from DSIWCS write to operator macro. (See *NetView Customization: Using Assembler* for more information on the DSIWCS macro).

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 and HLL user exit routines. The following types of variable pools can be accessed from HLL command processors or HLL user exit routines:

- The HLL command processor or user exit routine's own pool.
- The variable pool of the calling command procedure if the current HLL command processor or user exit routine was called from a command procedure.
- The task global pool shared by all command procedures and HLL user exit routines running under a NetView task.
- The common global pool shared by all command procedures and HLL user exit routines running in a NetView address space.

PL/I CALL FORMAT:
CALL CNMVARS(*hlbptr,cvfunc,cvdata,cvdatlen,cvname,cvpool*)

PL/I MACRO FORMAT:
CNMVARPOOL FUNC(*cvfunc*) NAME(*cvname*) POOL(*cvpool*) DATA(*cvdata*)
LENG(*cvdatlen*)

C INVOCATION:
void Cnmvars(char **cvfunc*, void **cvdata*, int *cvdatlen*, void **cvname*,
char **cvpool*)

Where:

hlbptr
A four byte pointer field containing the address of the HLB control block.

cvfunc
An eight byte character field which specifies the function to be performed. Required for all CNMVARS calls.

PUT	Set variable value.
GET	Get variable value.
DCL	Declare local variable to belong to one of the global pools, or reset it to the local pool. (You cannot declare a variable to belong to the caller's pool).

cvdata
A varying length character field containing the value of the named variable. Required for PUT and GET. Not used for DCL.

cvdatlen
A four byte integer field containing the length of *cvdata*. This is the maximum length of the area provided to receive the returned data. *cvdatlen* is provided by the user. Required only for GET.

If the value specified by *cvdatlen* is less than the length of the data to be returned, the truncated data will be returned in *cvdata* and a return code of

CNM_DATA_TRUNC will be generated. The full length of the data that was truncated is stored in HLBLENG (*Hlbleng*).

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 will be stored in HLBLENG (*Hlbleng*).

If the value specified by *cvdatlen* is greater than the length of the receiving data buffer (*cvdata*), a storage overlay could occur. Special care should be taken when deciding the value of *cvdatlen*.

cvname
A varying length character field which specifies the name of variable. Required for all functions.

Valid characters are A-Z, 0-9, @, #, $, ¢, ., !, ?, and underscore. The first character of *cvname* must not be a number or a period.

cvpool
An eight byte character field which specifies the variable pool. Required for all functions.

LOCAL	The local pool of the current HLL command processor or user exit routine.
TGLOBAL	Task global.
CGLOBAL	Common global.
CALLER	The local pool of the calling command procedure or HLL user exit routine (if one exists).

Usage Notes:

The user has the ability to 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 to 11 characters. In REXX, local variable names can be 1 to 250 characters while global variables must be 1 to 31 characters. In HLL, all variable names (local and global) are restricted to a length of 1 to 31.

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; refer to the parameter *cvname* above.

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 will be issued.

You do not have to initially PUT a value into the calling HLL command processor or user exit routine's LOCAL pool before issuing a PUT in the called HLL command processor or user exit routine's CALLER pool. When control is returned back to the calling HLL command processor or user exit routine, a GET can be issued for the same variable name in the LOCAL pool to retrieve the value set in the called HLL command processor or user exit routine's CALLER pool.

Any PUT's in an HLL command processor or user exit routine's CALLER pool will change the value of the same variable name in the calling command procedure or HLL user exit routine's LOCAL pool.

You can get three different values (*cvdata*) in the same variable name (*cvname*) if you specify different pools (*cvpool*). For example:

```
CNMVARS FUNC('PUT') NAME(x) POOL('LOCAL') DATA('cvdata1');
CNMVARS FUNC('PUT') NAME(x) POOL('CGLOBAL') DATA('cvdata2');
CNMVARS FUNC('PUT') NAME(x) POOL('TGGLOBAL') DATA('cvdata3');
```

The calling HLL command processor or user exit routine's LOCAL pool is the same as the called HLL command processor or user exit routine's CALLER pool.

DCL can be useful when a HLL command processor or user exit routine invokes VIEW. The user must insure that the variables are properly declared to the corresponding common or task global pools. Otherwise, the variables used may be from the LOCAL pool and the VIEW screen will not be automatically updated.

Task and common global variables must be declared to their respective pools before invoking VIEW from an HLL command processor or user exit routine. Otherwise, VIEW will not pick up the values. See the following examples:

Example 1:

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 VIEW panel from an HLL command processor or user exit routine. Before invoking VIEW, the following MUST be coded in the HLL command processor or user exit routine:

```
         CNMVARS FUNC('DCL') NAME(cvname) POOL('CGLOBAL')
or       CNMVARS FUNC('DCL') NAME(cvname) POOL('TGLOBAL').
```

Example 2:

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 changed within an HLL command processor or user exit routine and then displayed on a VIEW panel. Before invoking VIEW, the following MUST be coded in the HLL command processor or user exit routine:

```
        CNMVARS FUNC('PUT') NAME(cvname) POOL('CGLOBAL') DATA(cvdata)
        CNMVARS FUNC('DCL') NAME(cvname) POOL('CGLOBAL')
or
        CNMVARS FUNC('PUT') NAME(cvname) POOL('TGLOBAL') DATA(cvdata)
        CNMVARS FUNC('DCL') NAME(cvname) POOL('TGLOBAL')
```

Note: The DCL may precede the PUT.

Example 3:

An HLL command processor or user exit routine has set values for either task or common variables. Before invoking VIEW, the following must be coded in the HLL command processor or user 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 will be blank for that variable.

Example 4:

It may be necessary to declare (DCL) a variable back to the local pool. For example, assume you have the same variable name in both the local and common global pool. If you have just invoked 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 the following:

```
CNMVARS FUNC('DCL') NAME(cvname) POOL('LOCAL')
```

Return codes:

CNM_GOOD	0	Everything OK.
CNM_NOT_FOUND	20	*cvname* not found.
CNM_DATA_TRUNC	40	*cvdatlen* was too small. Data truncated.
CNM_BAD_FUNC	52	Invalid *cvfunc*.
CNM_BAD_LENGTH	88	*cvdatlen* less than (<) 0 or *cvdata* greater than (>) 255.
CNM_BAD_NAME	108	Invalid *cvname*.
CNM_BAD_POOL	156	Invalid *cvpool*.
CNM_BAD_ADDR	160	The storage pointed to by *cvdata* is not addressable.
CNM_BAD_CDS + X	14000 + X	Non-zero return code, X. See values for X below.

Values for X:

	4	Invalid variable name.
	8	Variable name already defined in dictionary.
	12	Insufficient storage.
	20	Value length limit was exceeded.
	28	No command procedure related to current action.
	32	Data was truncated.

Appendixes

Appendix A. PL/I Control Blocks and Include Files

This appendix describes the PL/I Control Blocks and Include files needed to write in PL/I.

DSIPLI

```
/*********************************************************************/
/*                                                                   */
/*  NAME = DSIPLI                                                    */
/*                                                                   */
/*  DESCRIPTIVE NAME = Main HLL PL/I Include File                    */
/*                                                                   */
/*  5665-362 for MVS/XA                                              */
/*  5664-204 for VM                                                  */
/*  5666-343 for VSE                                                 */
/*  THIS PRODUCT CONTAINS                                            */
/*  "RESTRICTED MATERIAL OF IBM"                                     */
/*  (c) COPYRIGHT IBM CORP 1989                                      */
/*  ALL RIGHTS RESERVED                                              */
/*  LICENSED MATERIALS-PROPERTY OF IBM                               */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                              */
/*  NUMBER G120-2083                                                 */
/*                                                                   */
/*  STATUS = NetView V1R3                                            */
/*                                                                   */
/*  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                                                  */
/*                                                                   */
/*********************************************************************/
%INCLUDE DSIPCONS;                 /* Constants                      */
%INCLUDE DSIPHLB;                  /* Mapping of HLB                 */
%INCLUDE DSIPORIG;                 /* Mapping of Origin block        */
%INCLUDE DSIPHLLS;                 /* HLL service rtn. definitions   */
%INCLUDE DSIPCNM;                  /* HLL return code constants      */
```

DSIPCONS

```
/**********************************************************************/
/*                                                                    */
/*  NAME = DSIPCONS                                                   */
/*                                                                    */
/*  DESCRIPTIVE NAME = HLL PL/I Constants                             */
/*                                                                    */
/*  5665-362 for MVS/XA                                               */
/*  THIS PRODUCT CONTAINS                                             */
/*  "RESTRICTED MATERIAL OF IBM"                                      */
/*  (c) COPYRIGHT IBM CORP 1989                                       */
/*  ALL RIGHTS RESERVED                                               */
/*  LICENSED MATERIALS-PROPERTY OF IBM                                */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                               */
/*  NUMBER G120-2083                                                  */
/*                                                                    */
/*  STATUS = NetView Release 3                                        */
/*                                                                    */
/*  FUNCTION = This file contains the definitions for constants      */
/*    that are helpful when coding HLL modules in 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                                                   */
/*                                                                    */
/**********************************************************************/
```

```
/*********************************************************************/
/* Constants common across HLL services                              */
/*********************************************************************/
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);

/*********************************************************************/
/* Constants for calls to CNMALTD                                    */
/*********************************************************************/
DCL INSLINE  CHAR(8) INIT('INSLINE '),
    REPLINE  CHAR(8) INIT('REPLINE '),
    DELLINE  CHAR(8) INIT('DELLINE ');
/*********************************************************************/
/* Constants for CNMCPYS                                             */
/*********************************************************************/
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 GETMSG   CHAR(8) INIT('GETMSG  '),
    GETLINE  CHAR(8) INIT('GETLINE '),
    PEEKLINE CHAR(8) INIT('PEEKLINE'),
    FLUSHLIN CHAR(8) INIT('FLUSHLIN'),
    FLUSHMSG CHAR(8) INIT('FLUSHMSG'),
    FLUSHQ   CHAR(8) INIT('FLUSHQ  ');

/*********************************************************************/
/* Constants for CNMCNMI                                             */
/*********************************************************************/
DCL SENDRESP CHAR(8) INIT('SENDRESP'),
    SENDRPLY CHAR(8) INIT('SENDRPLY');

/*********************************************************************/
/* 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 CNMSMSG                                             */
/*********************************************************************/
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 '),
    EXTLOG   CHAR(8) INIT('EXTLOG  '),
    SEQLOG   CHAR(8) INIT('SEQLOG  '),
    AUTHRCV  CHAR(8) INIT('AUTHRCV '),
    OPCLASS  CHAR(8) INIT('OPCLASS ');

/*********************************************************************/
/* 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   '),
    ENDREQ   CHAR(8) INIT('ENDREQ  '),
    UPDATE   CHAR(8) INIT('UPDATE  '),
    NOUPDATE CHAR(8) INIT('NOUPDATE'),
    DIRECT   CHAR(8) INIT('DIRECT  ');
```

```
/*********************************************************************/
/* Constants for user 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),
    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);
```

DSIPHLB

```
/*********************************************************************/
/*                                                                   */
/*  NAME = DSIPHLB                                                   */
/*                                                                   */
/*  DESCRIPTIVE NAME = HLL PL/I Mapping of DSIHLB                    */
/*                                                                   */
/*  5665-362 for MVS/XA                                              */
/*  THIS PRODUCT CONTAINS                                            */
/*  "RESTRICTED MATERIAL OF IBM"                                     */
/*  (c) COPYRIGHT IBM CORP 1989                                      */
/*  ALL RIGHTS RESERVED                                              */
/*  LICENSED MATERIALS-PROPERTY OF IBM                               */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                              */
/*  NUMBER G120-2083                                                 */
/*                                                                   */
/*  STATUS = NetView Release 3                                       */
/*                                                                   */
/*  FUNCTION = This files contains a PL/I mapping of DSIHLB, an     */
/*    internal control block.                                        */
/*                                                                   */
/*  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 DSIHLB BASED(HLBPTR),
      3 HLBLEN     FIXED BIN(31),  /* Length of HLB               */
      3 HLBWKA     PTR,            /* Pointer to WKA for API modules*/
      3 HLBHLLS    PTR,            /* Pointer to HLLS             */
      3 HLBTIB     PTR,            /* Pointer to TIB              */
      3 HLBUSER    PTR,            /* User Word                   */
      3 HLBRC      FIXED BIN(31),  /* Return code from last API call*/
      3 HLBLENG    FIXED BIN(31),  /* Length of data returned if
                                      HLBRC = 0.  Otherwise, length
                                      of data that would have been
                                      returned if truncation had
                                      not occurred.               */
      3 HLBSENSE   BIT(32),        /* Sense code from CNMI        */
      3 HLBRSRV    BIT(32),        /* Reserved                    */
      3 HLBFFDCA   CHAR(48);       /* First failure data capture  */
```

DSIPORIG

```
/*******************************************************************/
/*                                                                 */
/*  NAME = DSIPORIG                                                */
/*                                                                 */
/*  DESCRIPTIVE NAME = HLL PL/I Origin Block Mapping               */
/*                                                                 */
/*  5665-362 for MVS/XA                                            */
/*  THIS PRODUCT CONTAINS                                          */
/*  "RESTRICTED MATERIAL OF IBM"                                   */
/*  (c) COPYRIGHT IBM CORP 1989                                    */
/*  ALL RIGHTS RESERVED                                            */
/*  LICENSED MATERIALS-PROPERTY OF IBM                             */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                            */
/*  NUMBER G120-2083                                               */
/*                                                                 */
/*  STATUS = NetView Release 3                                     */
/*                                                                 */
/*  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                                                */
/*                                                                 */
/*******************************************************************/


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

```
/**********************************************************************/
/*                                                                    */
/*  NAME = DSIPHLLS                                                   */
/*                                                                    */
/*  DESCRIPTIVE NAME = PL/I Definitions for HLL Service Routines      */
/*                                                                    */
/*  5665-362 for MVS/XA                                               */
/*  THIS PRODUCT CONTAINS                                             */
/*  "RESTRICTED MATERIAL OF IBM"                                      */
/*  (c) COPYRIGHT IBM CORP 1989                                       */
/*  ALL RIGHTS RESERVED                                               */
/*  LICENSED MATERIALS-PROPERTY OF IBM                                */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                               */
/*  NUMBER G120-2083                                                  */
/*                                                                    */
/*  STATUS = NetView Release 3                                        */
/*                                                                    */
/*  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 DSIHLLS 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 CNMNAMS 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 BINARY(31),
                     *, FIXED BINARY(31), FIXED BINARY(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 CNMINFC ENTRY(PTR, CHAR(8), CHAR(*) VARYING, FIXED BIN(31))
                     OPTIONS(ASM,INTER,RETCODE),
      3 HLLSFIL7 CHAR(4),
      3 CNMINFI 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 CNMMEMO ENTRY(PTR, FIXED BIN(31), CHAR(8), CHAR(8))
                     OPTIONS(ASM,INTER,RETCODE),
      3 HLLSFILA CHAR(4),
      3 CNMMEMR ENTRY(PTR, FIXED BIN(31), CHAR(*) VARYING,
                     FIXED BIN(31))
                     OPTIONS(ASM,INTER,RETCODE),
      3 HLLSFILB CHAR(4),
      3 CNMMEMC 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),
      3 CNMSCAN ENTRY(PTR, CHAR(*) VARYING, CHAR(*) VARYING,
                     FIXED BIN(31),*,*,*,*,*,*,*,*,*,*)
                     OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILE CHAR(4),
        3 CNMKIO  ENTRY(PTR, CHAR(8), CHAR(*) VARYING, FIXED BIN(31),
                        CHAR(*) VARYING, CHAR(8) )
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILF CHAR(4),
        3 CNMSCOP ENTRY(PTR, CHAR(8), CHAR(8), CHAR(8))
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILG CHAR(4),
        3 CNMCPYS ENTRY(PTR, PTR, PTR, FIXED BIN(31),CHAR(8))
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILH CHAR(4),
        3 CNMLK   ENTRY(PTR, CHAR(8), CHAR(*) VARYING, CHAR(8),CHAR(8))
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILI CHAR(4),
        3 CNMPOOL ENTRY(PTR, CHAR(8), FIXED BIN(31,0),CHAR(*) VARYING,
                        FIXED BIN(31,0), FIXED BIN(31),
                        FIXED BIN(31),FIXED BIN(31))
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILJ CHAR(4),
        3 CNMALTD ENTRY(PTR, CHAR(8), CHAR(*) VARYING, *, FIXED BIN(31),
                        FIXED BIN(31))
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILK CHAR(4),
        3 CNMCELL ENTRY(PTR, CHAR(8), FIXED BIN(31), PTR)
                        OPTIONS(ASM,INTER,RETCODE),
        3 HLLSFILL 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 = '''''';
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 = '''''';
IF ¬PARMSET(ORIGIN) THEN ORIGIN = '''''';
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 CNMSENDMSG ENTRY;
%CNMSENDMSG:PROC(DATA,MSGTYPE,DESTTYPE,DEST) STATEMENT RETURNS(CHAR);
DCL (DATA,MSGTYPE,DESTTYPE,DEST,RTNSTR) CHAR;
IF ¬PARMSET(DEST) THEN DEST = '''''';
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 CNMINFC(HLBPTR,'||ITEM||','||DATA||','||LENG||');';
RETURN(RTNSTR);
%END;
%DCL CNMINFOI ENTRY;
%CNMINFOI:PROC(ITEM,DATA) STATEMENT RETURNS(CHAR);
DCL (ITEM,DATA,RTNSTR) CHAR;
RTNSTR = 'CALL CNMINFI(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 CNMMEMO(HLBPTR,'||TOKEN||','||DATASET||','||MEMBER||');';
RETURN(RTNSTR);
%END;

%DCL CNMREADMEM ENTRY;
%CNMREADMEM:PROC(TOKEN,DATA,LENG) STATEMENT RETURNS(CHAR);
DCL (TOKEN,DATA,LENG,RTNSTR) CHAR;
RTNSTR = 'CALL CNMMEMR(HLBPTR,'||TOKEN||','||DATA||','||LENG||');';
RETURN(RTNSTR);
%END;

%DCL CNMCLOSMEM ENTRY;
%CNMCLOSMEM:PROC(TOKEN) STATEMENT RETURNS(CHAR);
DCL (TOKEN,RTNSTR) CHAR;
RTNSTR = 'CALL CNMMEMC(HLBPTR,'||TOKEN||');';
RETURN(RTNSTR);
%END;

%DCL CNMI ENTRY;
%CNMI:PROC(FUNC,DATA,DEST,TIMEOUT) STATEMENT RETURNS(CHAR);
DCL (FUNC,DATA,DEST,TIMEOUT,RTNSTR) CHAR;
IF ¬PARMSET(TIMEOUT) THEN TIMEOUT = '0';
RTNSTR = 'CALL CNMCNMI(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';
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 = '''''';
IF ¬PARMSET(LENG)    THEN LENG = '0';
IF ¬PARMSET(KEY)     THEN KEY = '''''';
IF ¬PARMSET(OPTIONS) THEN OPTIONS = '''''';
DCL (FUNC,DATA,LENG,KEY,OPTIONS,RTNSTR) CHAR;
RTNSTR = 'CALL CNMKIO(HLBPTR,'||FUNC||','||DATA||','||LENG||','
            ||KEY||','||OPTIONS||');';
RETURN(RTNSTR);
%END;

%DCL CNMSCOPECK ENTRY;
%CNMSCOPECK:PROC(VERB,KEYWORD,VALUE) STATEMENT RETURNS(CHAR);
DCL (VERB,KEYWORD,VALUE,RTNSTR) CHAR;
IF ¬PARMSET(VERB)    THEN VERB = '''''';
IF ¬PARMSET(KEYWORD) THEN KEYWORD = '''''';
IF ¬PARMSET(VALUE)   THEN VALUE = '''''';
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 CNMCPYS(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 = '''''';
IF ¬PARMSET(OPTION) THEN OPTION = '''''';
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 CNMALTDATA ENTRY;
CNMALTDATA:PROC(FUNC,DATA,ORIGIN,QUEUE,LINE) STATEMENT RETURNS(CHAR);
CL (FUNC,DATA,ORIGIN,QUEUE,LINE,RTNSTR) CHAR;
F ¬PARMSET(DATA)   THEN DATA = '''''';
F ¬PARMSET(ORIGIN) THEN ORIGIN = '''''';
TNSTR = 'CALL CNMALTD(HLBPTR,'||FUNC||','||DATA||','||ORIGIN||','
         ||QUEUE||','||LINE||');';
TURN(RTNSTR);
ND;

 L CNMSTRCELL ENTRY;
 MSTRCELL:PROC(FUNC,TOKEN,STRPTR) STATEMENT RETURNS(CHAR);
  (FUNC,TOKEN,STRPTR,RTNSTR) CHAR;
 TR = 'CALL CNMCELL(HLBPTR,'||FUNC||','||TOKEN||','||STRPTR||');';
 RN(RTNSTR);
  ;
```

DSIPCNM

```
/*******************************************************************/
/*                                                                 */
/*  NAME = DSIPCNM                                                 */
/*                                                                 */
/*  DESCRIPTIVE NAME = HLL PL/I Return Codes                       */
/*                                                                 */
/*  5665-362 for MVS/XA                                            */
/*  THIS PRODUCT CONTAINS                                          */
/*  "RESTRICTED MATERIAL OF IBM"                                   */
/*  (c) COPYRIGHT IBM CORP 1989                                    */
/*  ALL RIGHTS RESERVED                                            */
/*  LICENSED MATERIALS-PROPERTY OF IBM                             */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                            */
/*  NUMBER G120-2083                                               */
/*                                                                 */
/*  STATUS = NetView Release 3                                     */
/*                                                                 */
/*  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_NEG_RESPONSE   FIXED; %CNM_NEG_RESPONSE   =64;
%DCL CNM_TIME_OUT       FIXED; %CNM_TIME_OUT       =68;
%DCL CNM_BAD_QUEUE      FIXED; %CNM_BAD_QUEUE      =72;
%DCL CNM_BAD_INDEX      FIXED; %CNM_BAD_INDEX      =76;
%DCL CNM_QUEUE_EMPTY    FIXED; %CNM_QUEUE_EMPTY    =80;
%DCL CNM_BAD_ORIGBLOCK  FIXED; %CNM_BAD_ORIGBLOCK  =84;
%DCL CNM_BAD_LENGTH     FIXED; %CNM_BAD_LENGTH     =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_DUPL_NAME      FIXED; %CNM_DUPL_NAME      =104;
%DCL CNM_BAD_NAME       FIXED; %CNM_BAD_NAME       =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_WA=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      52;
%DCL CNM_BAD_CELL_ADDRESS FIXED; %CNM_BAD_CELL_ADDRESS=256;
%DCL CNM_CELL_ALREADY_FREE FIXED; %CNM_CELL_ALREADY_FREE=260;
```

```
%DCL CNM_BAD_MQS          FIXED; %CNM_BAD_MQS          =1000;
%DCL CNM_DST_FAILURE      FIXED; %CNM_DST_FAILURE      =2000;
%DCL CNM_BAD_EXCMS        FIXED; %CNM_BAD_EXCMS        =3000;
%DCL CNM_BAD_PUSH         FIXED; %CNM_BAD_PUSH         =4000;
%DCL CNM_BAD_POP          FIXED; %CNM_BAD_POP          =5000;
%DCL CNM_BAD_WLS          FIXED; %CNM_BAD_WLS          =6000;
%DCL CNM_BAD_PSS          FIXED; %CNM_BAD_PSS          =7000;
%DCL CNM_BAD_WTO          FIXED; %CNM_BAD_WTO          =8000;
%DCL CNM_BAD_CES          FIXED; %CNM_BAD_CES          =9000;
%DCL CNM_BAD_DKS          FIXED; %CNM_BAD_DKS          =10000;
%DCL CNM_BAD_KVS          FIXED; %CNM_BAD_KVS          =11000;
%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_MRBLD        FIXED; %CNM_BAD_MRBLD        =18000;

%DCL CNM_BAD_ZCSMS        FIXED; %CNM_BAD_ZCSMS        =20000;
%DCL CNM_BAD_ENQ          FIXED; %CNM_BAD_ENQ          =21000;
```

Appendix B. PL/I Samples

This appendix contains a table of the PL/I samples that are shipped with NetView in SYS1.CNMSAMP. When data set names are referred to in this appendix, two names are given, such as PTMPPLT (CNMS4200). The first name is the alias name, and the name in parenthesis 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.

The following steps allow you to enter the member names as commands:

1. Compile and link edit the samples using the alias name.
2. Delete the (*) in column one of the appropriate CMDMDL statement in DSICMD to be able to execute the alias name as a command. No entries are needed in DSICMD for user exits.
3. NetView must be recycled to pick up the DSICMD changes.

Notes:

1. See the prologues of the samples for information about how certain samples are related and special cases for user exit routines.
2. Each alias name for PL/I begins with the letter P.
3. The alias name is the same as the procedure name, which is limited to seven characters, in PL/I.

This appendix also contains a description of each sample, and coded samples of a user exit routine and two command processors.

PL/I Samples Table

The following table refers to the PL/I samples that are shipped with NetView. The table contains the function, the alias name, and the name of the member in SYS1.CNMSAMP.

Sample function	PL/I Alias	Sample CNMSAMP
Template for commands and user exit routines	PTMPPLT	CNMS4200
Sample DSIEX03 to set a global variable	PEXIT3	CNMS4210
Uses CNMSMSG to send data	PSNDDAT	CNMS4211
Uses WAIT FOR DATA	PWATDAT	CNMS4212
Sample DSIEX02A changes a WTO to an MLWTO	PEXIT2A	CNMS4213
Uses CNMCNMI to forward RUs to a PU	PCNMI	CNMS4214
Uses CNMKIO for I/O to VSAM	PKEYIO	CNMS4215
HLL command using CNMSCOP for scope checking	PSCOPCK	CNMS4216
Display full screen VIEW panel	PFLVIEW	CNMS4217
Activates LU and uses TRAP and WAIT to determine if activation is successful	PACTLU	CNMS4218
Uses CNMSMSG to log text to a sequential log	PSEQLOG	CNMS4219
DST initialization exit USERVSAM DST	PXITDI	CNMS4220
Primes VSAM empty data set for USERVSAM DST	PXITVN	CNMS4221
Sends a request to USERVSAM DST	PSNDDST	CNMS4222
Processes VSAM requests under USERVSAM DST	PDOVSAM	CNMS4223
Primes VSAM empty data set for PKEYIO	PPRIME	CNMS4224

PL/I Samples Description

For each sample, a description of the function and the HLL service routines utilized are given.

PTMPPLT (CNMS4200)

This sample is a template for commands and user exit routines in PL/I.

This sample is included in Chapter 5 on page 37.

PEXIT3 (CNMS4210)

This is a sample DSIEX03 that sets a task global variable. This global variable will contain the value of the last time that a command other than PSNDDAT was entered under an OST. PWATDAT and PSNDDAT are used to interrogate this value.

The HLL service routines utilized in this sample are: CNMINFC (CNMINFOC), CNMVARS (CNMVARPOOL).

PSNDDAT (CNMS4211)

This sample uses CNMSMSG to send data. The sample is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example is to find 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 (it has received the data it was waiting for), and parses and displays the data.

The HLL service routines utilized in this sample are: CNMVARS (CNMVARPOOL), CNMSMSG (CNMSENDMSG), CNMINFC (CNMINFOC).

PWATDAT (CNMS4212)

This sample uses WAIT FOR DATA. The sample is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example is to find 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 (it has received the data it was waiting for), and parses and displays the data.

The HLL service routines utilized in this sample are: CNMSMSG (CNMSENDMSG), CNMSCAN (CNMSSCAN), CNMCMD (CNMCOMMAND), CNMGETD (CNMGETDATA).

PEXIT2A (CNMS4213)

This sample exit converts a WTO to an MLWTO by adding two lines to the single-line WTOs that are driving the exit.

The HLL service routines utilized in this sample are: CNMGETD (CNMGETDATA), CNMALTD (CNMALTDATA).

This sample is included in "Sample User Exit" on page 269.

PCNMI (CNMS4214)

This sample uses CNMCNMI to forward RUs to a PU. NetView provides the CNMCNMI service routine for use in communicating with devices in the network through the Communications Network Management Interface (CNMI). Any data that is returned may be accessed using the CNMGETD service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

This sample 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 prologue of the sample contains instructions for set up.

The HLL service routines utilized in this sample are: CNMSCAN (CNMSSCAN), CNMCNMI (CNMI), CNMGETD (CNMGETDATA), CNMSMSG (CNMSENDMSG).

PKEYIO (CNMS4215)

This sample illustrates how to code a NetView HLL command processor that allows I/O to a VSAM file via the CNMKIO service routine. It must execute on a DST. To run this command on a DST, either use the EXCMD command or the CNMSMSG service routine (with a type of COMMAND). The prologue of the sample explains how to set up a DST.

The HLL service routines utilized in this sample are: CNMKIO (CNMKEYIO), CNMSMSG (CNMSENDMSG).

PSCOPCK (CNMS4216)

This sample illustrates the scope checking capabilities provided by NetView. This sample scope checks keywords and values of the PSCOPCK command. In this sample, the user is required to set up the following elements for the command: operator ID, operator classes that can access the command, and operator profile. See the prologue of the sample for more information. This command yields a message informing the operator if he is not authorized to use the keyword and value specified when invoking the command.

The HLL service routines utilized in this sample are: CNMSCAN (CNMSSCAN), CNMSCOP (CNMSCOPECK), CNMSMSG (CNMSENDMSG).

This sample is included in "Sample Command Processor for Scope Checking" on page 272.

PFLVIEW (CNMS4217)

This sample illustrates the usage of the full screen VIEW command processor.

The HLL service routines utilized in this sample are: CNMCMD (CNMCOMMAND), CNMVARS (CNMVARPOOL).

PACTLU (CNMS4218)

This sample illustrates 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 utilized in this sample are: CNMSCAN (CNMSSCAN), CNMCMD (CNMCOMMAND), CNMGETD (CNMGETDATA), CNMSMSG CNMSENDMSG).

PSEQLOG (CNMS4219)

This sample uses CNMSMSG to log text to a sequential log. The prologue of the sample contains instructions for set up.

The HLL service routines utilized in this sample are: CNMSCAN (CNMSSCAN), CNMINFC (CNMINFOC), CNMSMSG (CNMSENDMSG).

This sample is included in "Sample Command Processor for Sequential Logging" on page 276.

PXITDI (CNMS4220)

This sample is a DST initialization exit. The sample illustrates the DST initialization exit that is used by the USERVSAM DST.

The HLL service routines utilized in this sample are: CNMVARS (CNMVARPOOL), CNMSMSG (CNMSENDMSG).

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 named 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 Command List Language variable (defined by the caller) to be set to the retrieved value.

The HLL service routines utilized in this sample are: CNMSCAN (CNMSSCAN), CNMSMSG (CNMSENDMSG), CNMVARS (CNMVARPOOL), CNMGETD (CNMGETDATA), CNMCMD (CNMCOMMAND).

PDOVSAM (CNMS4223)

This sample is an HLL Data Services Command Processor which runs under the sample Data Services Task (task ID 'USERVSAM'). It processes 'PUT' or 'GET' requests sent by the PSNDDST sample, and will write or read 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 utilized in this sample are: CNMSCAN (CNMSSCAN), CNMSMSG (CNMSENDMSG), CNMKIO (CNMKEYIO).

PPRIME (CNMS4224)

This sample primes a VSAM empty data set for PKEYIO.

PL/I Coded Samples

This section contains an example of a user exit routine and two command processors.

Sample User Exit

This sample is an example of user exit DSIEX02A.

```
PEXIT2A:  PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/*********************************************************************/
/*                                                                   */
/*   (C) COPYRIGHT IBM CORP. 1989                                    */
/*                                                                   */
/*   IEBCOPY   SELECT MEMBER=((CNMS4213,PEXIT2A,R))                  */
/*                                                                   */
/*   Descriptive Name: High Level Language PL/I DSIEX02A             */
/*                           Example                                 */
/*                                                                   */
/*     Function:                                                     */
/*                                                                   */
/*     This DSIEX02A adds two lines to the single line WTOs          */
/*     that are driving the exit.                                    */
/*                                                                   */
/*     Introduction:                                                 */
/*                                                                   */
/*     To convert a single line WTO into a multi-line WTO,           */
/*     you must change the line types accordingly:                   */
/*                                                                   */
/*     In                    Out                                     */
/*     ---                   ---                                     */
/*     msg_type = "E",msg_type="=" line_type="C" (Control)           */
/*        and a              msg_type="=" line_type="D" (Data)       */
/*     line_type of " " msg_type="=" line_type="F" (Final)           */
/*                                                                   */
/*     Note:                                                         */
/*     A Msg_type of "E" implies that the message was externally     */
/*     generated via a WTO.  For example MVS D T would send          */
/*     a Msg_type of "E".  The Msg_type of "=" implies a user        */
/*     generated MLWTO was issued.                                   */
/*                                                                   */
/*   Dependencies:  None                                             */
/*                                                                   */
/*   Restrictions: Only processes messages issued under mainline     */
/*                   processing.                                     */
/*                                                                   */
/*   Language:  PL/I                                                 */
/*                                                                   */
/*   Input:                                                          */
/*      1)  4-byte pointer to the HLB control block                  */
/*      2)  varying length character string of command (or message   */
/*          for user exits) that invoked this procedure              */
/*      3)  40-byte parameter list which describes the origin of     */
/*          the request that caused execution of this procedure.     */
/*                                                                   */
/*   Output:                                                         */
/*      Messages to various tasks.                                   */
/*                                                                   */
/*   Return Codes: returned in HLBRC                                 */
/*      0 = normal exit                                              */
/*                                                                   */
/*   External Module References:  None                               */
/*                                                                   */
/*   Change Activity:                                                */
/*      date,author: Description of changes                          */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*  Parameter 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;              /* Builtin function              */
ORIGIN=ADDR(ORIGBLCK);             /* Address the Orig Block        */

/*********************************************************************/
/*                                                                   */
/*  Other Declarations                                               */
/*                                                                   */
/*********************************************************************/
DCL GETBLOCK CHAR(40);             /* Area for the Orig Block       */
DCL DATAIN   CHAR(255) VAR;        /* Message that drove the exit   */
```

```
/**********************************************************************/
/*                                                                    */
/*  Execution                                                         */
/*                                                                    */
/**********************************************************************/
GETPTR=ADDR(GETBLOCK);             /* Address the Orig block         */
CNMGETDATA                         /* 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 ='E' THEN /*WTO response to MVS command     */
  DO;
    GETPTR->ORIG_MSG_TYPE ='=';  /* Set msg type to MLWTO           */
    GETPTR->ORIG_LINE_TYPE='C';    /* Set line type to control       */
    CNMALTDATA                     /* Replace the text ...           */
      FUNC('REPLINE')              /* ...function is replace         */
      QUEUE(IDATAQ)                /* ...initial data queue          */
      DATA('Change WTO to MLWTO, WTO="'||DATAIN||'"')
                                   /* ...text of new message         */
      ORIGIN(GETBLOCK)             /* ...use Peeked Orig block       */
      LINE(1);                     /* ...replace the first line      */

    GETPTR->ORIG_LINE_TYPE='D';    /* Set line type to data          */
    CNMALTDATA                     /* Insert a new line...           */
      FUNC('INSLINE')              /* ...function is insert          */
      QUEUE(IDATAQ)                /* ...initial data queue          */
      DATA('Add a data line to the MLWTO')
      ORIGIN(GETBLOCK)             /* ...use Peeked Orig block       */
      LINE(2);                     /* ...add a line                  */

    GETPTR->ORIG_LINE_TYPE='F';  /* Set line type to final line    */
    CNMALTDATA                     /* Insert a new line...           */
      FUNC('INSLINE')              /* ...function is insert          */
      QUEUE(IDATAQ)                /* ...initial data queue          */
      DATA('Add an end of MLWTO msg')
      ORIGIN(GETBLOCK)             /* ...use Peeked Orig block       */
      LINE(3);                     /* ...add a line                  */
    HLBRC=CNM_GOOD;                /* Issue clean rc...              */
  END;
END PEXIT2A;
```

Sample Command Processor for Scope Checking

This sample is an example of a command processor for scope checking.

```
PSCOPCK:  PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/*******************************************************************/
/*                                                                 */
/*   (C) COPYRIGHT IBM CORP. 1989                                  */
/*                                                                 */
/*   IEBCOPY   SELECT MEMBER=((CNMS4216,PSCOPCK,R))                */
/*                                                                 */
/*   Descriptive Name: High Level Language PL/I                    */
/*                     Scope Check Example                         */
/*                                                                 */
/*   Function:                                                     */
/*                                                                 */
/* The following is an example of the scope checking capabilities  */
/* provided by NetView.  In this example, the user is required to  */
/* set up the following elements for the command (shown below):    */
/*  (1)  operator id                                               */
/*  (2)  operator classes that can access the command              */
/*  (3)  operator profile                                          */
/*                                                                 */
/* The command gives the return code that the scope check service  */
/* routine returned to the operator.                               */
/*                                                                 */
/* The syntax that this command checks for is:                     */
/*                                                                 */
/*     PSCOPCK PARMx(VALx)                                         */
/*                                                                 */
/* The following is the setup for the scope check example:         */
/*                                                                 */
/*   In DSIPARM(DSICMD):                                           */
/*              Define the operator classes that can access        */
/*                the command, its keywords, and its keyword values.*/
/*                                                                 */
/*              The example below says that the command HLLSCOPE   */
/*                 can be executed by operators in scope class     */
/*                 1 and 2.  Scope class 1 can issue any keyword   */
/*                 or keyword value, but scope class 2 cannot use  */
/*                 the value of VAL1 with keyword PARM2, and scope */
/*                 class 2 cannot issue PARM3 at all.              */
/*                                                                 */
/*   Example:                                                      */
/*                                                                 */
/*     HLLSCOPE  CMDMDL    MOD=HLLMOD,RES=N,TYPE=RD                */
/*               CMDCLASS  1,2                                     */
/*     PARM2     KEYCLASS  1,2                                     */
/*     VAL1      VALCLASS  1                                       */
/*     PARM3     KEYCLASS  1                                       */
/*     VAL1      VALCLASS  1                                       */
/*                                                                 */
/*   In DSIPARM(DSIOPF)                                              */
/*     Define the operator ids and the profiles that the operator   */
/*       ids can use.                                                */
/*                                                                   */
/*   Example:                                                        */
/*     JOE      OPERATOR  PASSWORD=USER                              */
/*              PROFILEN  DSIPROF3                                   */
/*                                                                   */
/*   In DSIPRF(profilename)                                          */
/*     Define the operator class value that will correspond to the  */
/*     profile that the operator logs on with.                       */
/*                                                                   */
/*   Example:                                                        */
/*     In the DSIPRF dataset, member name DSIPROF3,                  */
/*                                                                   */
/*     DSIPROF3   PROFILE                                            */
/*                OPCLASS 3                                          */
/*                END                                                */
/*                                                                   */
/*   Restrictions:  None                                             */
/*                                                                   */
/*   Language:  PL/I                                                 */
/*                                                                   */
/*   Input:                                                          */
/*      1)  4-byte pointer to the HLB control block                  */
/*      2)  varying length character string of command (or message  */
/*          for user exits) that invoked this procedure              */
/*      3)  40-byte parameter list which describes the origin of    */
/*          the request that caused execution of this procedure.    */
/*                                                                   */
/*   Output:                                                         */
/*      Messages describing the scope of the operator.               */
/*                                                                   */
/*   Return Codes: returned in HLBRC                                 */
/*      0 = normal exit                                              */
/*     -5 = cancelled                                                */
/*                                                                   */
/*   External Module References:  None                               */
/*                                                                   */
/*   Change Activity:                                                */
/*      date,author: description of changes                          */
/*********************************************************************/
```

```
/*********************************************************************/
/*                                                                   */
/*  Parameter 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;               /* Builtin function              */
ORIGIN=ADDR(ORIGBLCK);              /* Address the Orig Block        */

/*********************************************************************/
/*                                                                   */
/*  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                                                        */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*               Scan the keyword and the value                      */
/*********************************************************************/
CALL CNMSCAN(HLBPTR,                /* Parse the input ...           */
             CMDBUF,                /* ...command line is the input  */
                                    /* SYNTAX OF COMMAND IS:         */
                                    /*  CMDNAME KEYWORD(KEYVALUE)    */
                                    /*                               */
```

```
                                   /* Scan for the:                */
          '%S'||                   /* ...command name              */
          '%*{ }'||                /* ...skip over leading blanks  */
          '%{¬(}'||                /* ...keyword up to "("         */
          '%*C'||                  /* ...skip over "("             */
          '%{¬)}',                 /* ...keyvalue up to ")"        */
          CNT,                     /* ...number strings parsed     */
          CMDNAMEV,                /* ...command goes here         */
          KEYWORDV,                /* ...keyword goes here         */
          KEYVALUEV);              /* ...keyvalue goes in here     */
CMDNAME=CMDNAMEV;                  /* Get fixed length value       */
KEYWORD=KEYWORDV;                  /* Get fixed length value       */
KEYVALUE=KEYVALUEV;                /* Get fixed length value       */
IF CNT=3 THEN                      /* Enough parms specified?      */
  CALL CNMSCOP(HLBPTR,             /* Scope check the input...     */
           CMDNAME,                /* ...the command               */
           KEYWORD,                /* ...the keyword               */
           KEYVALUE);              /* ...the value                 */

ELSE                               /* Not enough parms specified   */
  HLBRC=CNM_BAD_INVOCATION;        /* Set rc                       */


/*******************************************************************/
/*                 Inform user of the return code results...       */
/*******************************************************************/
  SELECT;
    WHEN(HLBRC=CNM_GOOD)
      DO;                          /* Operator                     */
                                   /*      has                     */
                                   /*          passed              */
      END;                         /*              scope checking  */
    WHEN(HLBRC=CNM_KEYWORD_NA)
      CALL CNMSMSG(HLBPTR,' Not authorized to use KEYWORD '||KEYWORD,
              'MSG','OPER','');
    WHEN(HLBRC=CNM_VALUE_NA)
      CALL CNMSMSG(HLBPTR,' Not authorized to use VALUE   '||KEYVALUE,
              'MSG','OPER','');
    WHEN(HLBRC=CNM_BAD_INVOCATION)
      CALL CNMSMSG(HLBPTR,' Not enough parms specified',
              'MSG','OPER','');
    OTHERWISE
      CALL CNMSMSG(HLBPTR,' RC not recognized...'||HLBRC,
              'MSG','OPER','');
  END;
  HLBRC=CNM_GOOD;                /* Clear RC                     */
END PSCOPCK;
```

Sample Command Processor for Sequential Logging

This sample is an example of a command processor to log text to a sequential log.

```
PSEQLOG:  PROC(HLBPTR,CMDBUF,ORIGBLCK) OPTIONS(MAIN,REENTRANT);
/********************************************************************/
/*                                                                  */
/*   (C) COPYRIGHT IBM CORP. 1989                                   */
/*                                                                  */
/* IEBCOPY   SELECT MEMBER=((CNMS4219,PSEQLOG,R))                   */
/*                                                                  */
/*   Descriptive Name:  High Level Language PL/I Sequential         */
/*                      Logging Example                             */
/*                                                                  */
/*      Function:                                                   */
/*                                                                  */
/*      Write the text passed to this command procedure via the     */
/*      command line to the log.                                    */
/*                                                                  */
/*   The syntax of this command is:                                 */
/*                                                                  */
/*      PSEQLOG LOGTEXT                                             */
/*                                                                  */
/*   Dependencies:  NONE                                            */
/*                                                                  */
/*   Restrictions:  NONE                                            */
/*                                                                  */
/*   Language::  PL/I                                               */
/*                                                                  */
/*   Installation:                                                  */
/*                                                                  */
/*   (1)  ASSEMBLE AND LINKEDIT THIS MODULE AMODE=31,RMODE=ANY      */
/*        TYPE=RENT                                                 */
/*   (2)  ALLOC PRIMARY AND SECONDARY SEQUENTIAL DATA SET           */
/*   (3)  USE DD NAMES IN NETVIEW PROC OR THE ALLOCATE COMMAND      */
/*        TO ALLOCATE THE DATA SETS TO NETVIEW.                     */
/*        ALLOCATE THE DATA SETS AS                                 */
/*            SQLOGP & SQLOGS                                       */
/*   (4)  ADD THE FOLLOWING STATEMENT TO DSIDMN                     */
/*       TASK   MOD=DSIZDST,TSKID=SQLOGTSK,MEM=SQLOGMEM,PRI=3,INIT=Y*/
/*   (5)  ADD THE FOLLOWING MEMBER (SQLOGMEM) TO DSIPARM            */
/*         DSTINIT FUNCT=OTHER,DSRBO=1                              */
/*         DSTINIT PBSDN=SQLOGP                                     */
/*         DSTINIT SBSDN=SQLOGS                                     */
/*         LOGINIT AUTOFLIP=YES,RESUME=NO                           */
/*   (6)  ADD THE FOLLOWING CMDMDL TO DSICMD                        */
/*         PSEQLOG CMDMDL MOD=PSEQLOG,TYPE=R,RES=N                  */
/*                                                                  */
/*   Input:                                                          */
/*      1)  4-byte pointer to the HLB control block                  */
/*      2)  varying length character string of command (or message  */
/*          for user exits) that invoked this procedure              */
/*      3)  40-byte parameter list which describes the origin of     */
/*          the request that caused execution of this procedure.     */
/*                                                                   */
/*   Output:                                                         */
/*      Writes input to a sequential log                             */
/*                                                                   */
/*   Return Codes: returned in HLBRC                                 */
/*      0 = normal exit                                              */
/*     -5 = canceled                                                 */
/*                                                                   */
/*   External Module References:  None                               */
/*                                                                   */
/*   Change Activity:                                                */
/*      date,author: description of changes                          */
/*                                                                   */
/*********************************************************************/
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      */
ORIGIN=ADDR(ORIGBLCK);             /* Address the Orig Block         */
/*********************************************************************/
/*                                                                   */
/*  Customization starts here.....                                   */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*  Declares                                                         */
/*                                                                   */
/*********************************************************************/
DCL logtask CHAR(8)                /* Sequential log task id -       */
       INIT('SQLOGTSK');           /*  specified in DSIDMN           */
DCL domain  CHAR(8) var;           /* Domain name                    */
DCL logtime CHAR(8) var;           /* Time                           */
DCL opid    CHAR(8) var;           /* Operator name                  */
DCL logtext CHAR(229) var;         /* Holds text to be logged        */
DCL loghdr  CHAR(27) INIT((27)' '); /* Holds some header info        */
DCL logbfr  CHAR(256) var;         /* Holds the buffer to be logged */
DCL errmsg  CHAR(80)  var;         /* Holds error message            */
DCL rc      CHAR(16)  var;         /* Character form of return code */
dcl parmcnt fixed bin(31);         /* Number of elemets parsed       */
```

```
/*********************************************************************/
/*                                                                   */
/*  Execution                                                        */
/*                                                                   */
/*********************************************************************/

 CNMSSCAN DATA(cmdbuf)            /* Scan the input                 */
          FORMAT('%*S%*C%{¬}')    /* skipping the cmd name    %*S */
                                  /* and the following blank  %*C */
                                  /* the remainder is the text      */
                                  /* to be logged             %{¬} */
          COUNT(parmcnt)          /* number of elements scanned     */
          P1(logtext);            /* scan text into logtext         */

 CNMINFOC ITEM('DOMAIN')          /* Get the domain name            */
          DATA(domain) LENG(8);
 CNMINFOC ITEM('TIME')            /* and the time                   */
          DATA(logtime) LENG(8);
 CNMINFOC ITEM('OPID')            /* and the operator id            */
          DATA(opid) LENG(8);

 substr(loghdr,1) = domain;       /* Put domain name in header      */
 substr(loghdr,10) = logtime;     /* ditto for time                 */
 substr(loghdr,19) = opid;        /* and opid                       */

 logbfr = loghdr || logtext;      /* Concat header and text         */

 CNMSENDMSG DATA(logbfr)          /* Text is in logbfr              */
            MSGTYPE('MSG')        /* message type is 'MSG'          */
            DESTTYPE('SEQLOG')    /* destination is sequential log */
            DEST(logtask);        /* name of task is in logtask     */
/*********************************************************************/
/*                  Inform user of the return code results...        */
/*********************************************************************/
 if hlbrc ¬= 0 then do;
   rc = char(hlbrc);
   errmsg = 'SLOG000 ERROR, RC from SENDMSG = '||
      substr(rc,length(rc)-4);
   CNMSENDMSG DATA(errmsg) MSGTYPE('MSG')
     DESTTYPE('TASK') DEST(origin->orig_task);
  end;

END PSEQLOG;
```

Appendix C. C Control Blocks and Include Files

This appendix describes the C Control Blocks and Include files needed to write in C.

DSIC

```
/*********************************************************************/
/*                                                                   */
/*  NAME = DSIC                                                      */
/*                                                                   */
/*  DESCRIPTIVE NAME = Main HLL C Include File                       */
/*                                                                   */
/*  5665-362 for MVS/XA                                              */
/*  THIS PRODUCT CONTAINS                                            */
/*  "RESTRICTED MATERIAL OF IBM"                                     */
/*  (c) COPYRIGHT IBM CORP 1989                                      */
/*  ALL RIGHTS RESERVED                                              */
/*  LICENSED MATERIALS-PROPERTY OF IBM                               */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                              */
/*  NUMBER G120-2083                                                 */
/*                                                                   */
/*  STATUS = NetView Release 3                                       */
/*                                                                   */
/*  FUNCTION = 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.                           */
/*                                                                   */
/*  NOTES = see below                                                */
/*                                                                   */
/*    DEPENDENCIES = none                                            */
/*                                                                   */
/*  RESTRICTIONS = none                                              */
/*                                                                   */
/*    REGISTER CONVENTIONS = not applicable                          */
/*                                                                   */
/*    PATCH LABEL = not applicable                                   */
/*                                                                   */
/*  MODULE TYPE = include file                                       */
/*                                                                   */
/*    PROCESSOR = C                                                  */
/*                                                                   */
/*  EXTERNAL REFERENCES = none                                       */
/*                                                                   */
/*  CHANGE ACTIVITY                                                  */
/*                                                                   */
/*********************************************************************/
#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 "dsiccall.h"       /* HLL function definitions       */
#include "dsiccnm.h"        /* HLL return code constants      */
```

DSICCONS

```
/**********************************************************************/
/*                                                                    */
/*  NAME = DSICCONS                                                   */
/*                                                                    */
/*  DESCRIPTIVE NAME = HLL C Constants                                */
/*                                                                    */
/*  5665-362 for MVS/XA                                               */
/*  THIS PRODUCT CONTAINS                                             */
/*  "RESTRICTED MATERIAL OF IBM"                                      */
/*  (c) COPYRIGHT IBM CORP 1989                                       */
/*  ALL RIGHTS RESERVED                                               */
/*  LICENSED MATERIALS-PROPERTY OF IBM                                */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                               */
/*  NUMBER G120-2083                                                  */
/*                                                                    */
/*  STATUS = NetView Release 3                                        */
/*                                                                    */
/*  FUNCTION = This file contains the definitions for constants      */
/*    that are helpful when coding HLL modules in 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                                                   */
/*                                                                    */
/**********************************************************************/

/**********************************************************************/
/* Constants common across HLL services                               */
/**********************************************************************/
#define ZERO          0x00
#define TRAPQ      1
#define OPERQ      2
#define DATAQ      3
#define IDATAQ     4
#define CNMIQ      5
```

```
/*******************************************************************/
/* Constants for calls to Cnmaltd                                  */
/*******************************************************************/
#define INSLINE    "INSLINE "
#define REPLINE    "REPLINE "
#define DELLINE    "DELLINE "

/*******************************************************************/
/* Constants for Cnmcpys                                           */
/*******************************************************************/
#define FIXTOFIX   "FIXTOFIX"
#define FIXTOVAR   "FIXTOVAR"
#define VARTOFIX   "VARTOFIX"
#define VARTOVAR   "VARTOVAR"

/*******************************************************************/
/* Constants for calls to Cnmgetd                                  */
/*******************************************************************/
#define GETMSG     "GETMSG  "
#define GETLINE    "GETLINE "
#define PEEKLINE   "PEEKLINE"
#define FLUSHLIN   "FLUSHLIN"
#define FLUSHMSG   "FLUSHMSG"
#define FLUSHQ     "FLUSHQ  "

/*******************************************************************/
/* Constants for Cnmcnmi                                           */
/*******************************************************************/
#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 Cnmnams, Cnmpool 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 Cnmsmsg                                             */
/*********************************************************************/
#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 EXTLOG     "EXTLOG  "
#define SEQLOG     "SEQLOG  "
#define AUTHRCV    "AUTHRCV "
#define OPCLASS    "OPCLASS "
#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 ENDREQ      "ENDREQ  "
#define UPDATE      "UPDATE  "
#define NOUPDATE    "NOUPDATE"
#define DIRECT      "DIRECT  "

/**********************************************************************/
/* Constants for user 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                                  */
/**********************************************************************/
#define NOHEXCNV     0
#define CNVTOHEX     1
```

```
/********************************************************************/
/*                                                                  */
/*  NAME = DSICVARC                                                 */
/*                                                                  */
/*  DESCRIPTIVE NAME = HLL C Varying Length Character Stings        */
/*                                                                  */
/*  5665-362 for MVS/XA                                             */
/*  THIS PRODUCT CONTAINS                                           */
/*  "RESTRICTED MATERIAL OF IBM"                                    */
/*  (c) COPYRIGHT IBM CORP 1989                                     */
/*  ALL RIGHTS RESERVED                                             */
/*  LICENSED MATERIALS-PROPERTY OF IBM                              */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                             */
/*  NUMBER G120-2083                                                */
/*                                                                  */
/*  STATUS = NetView Release 3                                      */
/*                                                                  */
/*  FUNCTION = DSIVARCH 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

```
/*********************************************************************/
/*                                                                   */
/*  NAME = DSICHLB                                                   */
/*                                                                   */
/*  DESCRIPTIVE NAME = HLL C Mapping of DSIHLB                       */
/*                                                                   */
/*  5665-362 for MVS/XA                                              */
/*  THIS PRODUCT CONTAINS                                            */
/*  "RESTRICTED MATERIAL OF IBM"                                     */
/*  (c) COPYRIGHT IBM CORP 1989                                      */
/*  ALL RIGHTS RESERVED                                              */
/*  LICENSED MATERIALS-PROPERTY OF IBM                               */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                              */
/*  NUMBER G120-2083                                                 */
/*                                                                   */
/*  STATUS = NetView Release 3                                       */
/*                                                                   */
/*  FUNCTION = This file contains a C mapping of DSIHLB, an          */
/*    internal control block.                                        */
/*                                                                   */
/*  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     Hlblen;                  /* Length of HLB               */
  int     *Hlbwka;                 /* Pointer to WKA for API modules*/
  int     *Hlbhlls;                /* Pointer to HLLS (¬ used by C) */
  int     *Hlbtib;                 /* Pointer to TIB               */
  int     *Hlbuser;                /* User word                    */
  int     Hlbrc;                   /* Return code from last API call*/
  int     Hlbleng;                 /* Length of data returned if
                                      Hlbrc = 0.  Otherwise, length
                                      of data that would have been
                                      returned if truncation had
                                      not occurred.                */
  unsigned int Hlbsense;           /* Sense code from CNMI         */
  unsigned int Hlbrsrv;            /* Reserved                     */
  char         Hlbffdca??(48??);   /* First failure data capture   */
               } Dsihlb;
```

DSICORIG

```
/**********************************************************************/
/*                                                                    */
/*  NAME = DSICORIG                                                   */
/*                                                                    */
/*  DESCRIPTIVE NAME = HLL C Origin Block Mapping                     */
/*                                                                    */
/*  5665-362 for MVS/XA                                               */
/*  THIS PRODUCT CONTAINS                                             */
/*  "RESTRICTED MATERIAL OF IBM"                                      */
/*  (c) COPYRIGHT IBM CORP 1989                                       */
/*  ALL RIGHTS RESERVED                                               */
/*  LICENSED MATERIALS-PROPERTY OF IBM                                */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                               */
/*  NUMBER G120-2083                                                  */
/*                                                                    */
/*  STATUS = NetView Release 3                                        */
/*                                                                    */
/*  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;
```

DSICCALL

```
/********************************************************************/
/*                                                                  */
/*  NAME = DSICCALL                                                 */
/*                                                                  */
/*  DESCRIPTIVE NAME =  HLL C Service Routine Definitions           */
/*                                                                  */
/*  5665-362 for MVS/XA                                             */
/*  THIS PRODUCT CONTAINS                                           */
/*  "RESTRICTED MATERIAL OF IBM"                                    */
/*  (c) COPYRIGHT IBM CORP 1989                                     */
/*  ALL RIGHTS RESERVED                                             */
/*  LICENSED MATERIALS-PROPERTY OF IBM                              */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                             */
/*  NUMBER G120-2083                                                */
/*                                                                  */
/*  STATUS = NetView Release 3                                      */
/*                                                                  */
/*  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                                                 */
/*                                                                  */
/********************************************************************/
```

```
/*********************************************************************/
/* Preprocessor directives                                           */
/*   Each of the HLL service routines uses OS linkage.               */
/*********************************************************************/
#pragma linkage(Cnmaltd, OS)
#pragma linkage(Cnmcell, OS)
#pragma linkage(Cnmcmd,  OS)
#pragma linkage(Cnmcnmi, OS)
#pragma linkage(Cnmcpys, OS)
#pragma linkage(Cnmgeta, OS)
#pragma linkage(Cnmgetd, OS)
#pragma linkage(Cnminfc, OS)
#pragma linkage(Cnminfi, OS)
#pragma linkage(Cnmkio,  OS)
#pragma linkage(Cnmlk,   OS)
#pragma linkage(Cnmmemo, OS)
#pragma linkage(Cnmmemr, OS)
#pragma linkage(Cnmmemc, OS)
#pragma linkage(Cnmnams, OS)
#pragma linkage(Cnmpool, OS)
#pragma linkage(Cnmscop, OS)
#pragma linkage(Cnmsmsg, OS)
#pragma linkage(Cnmvars, 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 Cnmcnmi(Dsihlb **hlbptr, char *cnfunc, void *cndata,
            char *cndest, int cntimout);
void Cnmcpys(Dsihlb **hlbptr, void *csfrom, void *csto, int cslen,
            char *cstype);
void Cnmgeta(Dsihlb **hlbptr, char *ganame, void *gadata,
            int gadatlen, int gaqueue);
void Cnmgetd(Dsihlb **hlbptr, char *gdfunc, void *gdbuf,
            int gdbuflen, void *gdorigin, int gdqueue, int gdindex);
void Cnminfc(Dsihlb **hlbptr, char *icname, void *icdata,
            int icdatlen);
void Cnminfi(Dsihlb **hlbptr, char *iiname, int *iinumb);
void Cnmkio(Dsihlb **hlbptr, char *vsfunc, void *vsdata,
            int vsdatlen, void *vskey, char *vsoption);
void Cnmlk(Dsihlb **hlbptr, char *lkfunc, void *lkname,
            char *lkscope, char *lkoption);
void Cnmmemo(Dsihlb **hlbptr, int *motoken, char *moddname,
            char *momemnam);
```

```
void Cnmmemr(Dsihlb **hlbptr, int mrtoken, void *mrdata, int mrdatlen);
void Cnmmemc(Dsihlb **hlbptr, int mctoken);
void Cnmnams(Dsihlb **hlbptr, char *nsfunc, void *nsptr,
            void *nsname, int *nsleng, int nsclass);
void Cnmpool(Dsihlb **hlbptr, char *spfunc, int *sptoken, void *spname,
            int spleng, int sppricnt, int spseccnt, int spclass);
void Cnmscop(Dsihlb **hlbptr, char *sccmd, char *sckwd, char *scvalue);
void Cnmsmsg(Dsihlb **hlbptr, void *smtext, char *smmsgtype,
            char *smdestyp, char *smdestid);
void Cnmvars(Dsihlb **hlbptr, char *cvfunc, void *cvdata,
            int cvdatlen, void *cvname, char *cvpool);

/*******************************************************************/
/* Function declarations                                           */
/*   Functions provided for use with varying length character      */
/*   strings.                                                      */
/*******************************************************************/
void *Cnmvlc(void *vstring, short convert, char *istring, ...);
void *Cnmnvlc(void *vstring, short convert, int length, char *istring);

/*******************************************************************/
/* 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 Cnmcmd(cmdstr)                                         \
        Cnmcmd(&Hlbptr,cmdstr)

#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)

#define Cnmgetd(gdfunc,gdbuf,gdbuflen,gdorigin,                \
                gdqueue,gdindex)                               \
        Cnmgetd(&Hlbptr,gdfunc,gdbuf,gdbuflen,gdorigin,        \
                gdqueue,gdindex)

#define Cnminfc(icname,icdata,icdatlen)                        \
        Cnminfc(&Hlbptr,icname,icdata,icdatlen)
```

```
#define Cnminfi(iiname,iinumb)                                          \
        Cnminfi(&Hlbptr,iiname,iinumb)

#define Cnmkio(vsfunc,vsdata,vsdatlen,vskey,vsoption)                   \
        Cnmkio(&Hlbptr,vsfunc,vsdata,vsdatlen,vskey,vsoption)

#define Cnmlk(lkfunc,lkname,lkscope,lkoption)                           \
        Cnmlk(&Hlbptr,lkfunc,lkname,lkscope,lkoption)

#define Cnmmemo(motoken,moddname,momemnam)                              \
        Cnmmemo(&Hlbptr,motoken,moddname,momemnam)

#define Cnmmemr(mrtoken,mrdata,mrdatlen)                                \
        Cnmmemr(&Hlbptr,mrtoken,mrdata,mrdatlen)

#define Cnmmemc(mctoken)                                                \
        Cnmmemc(&Hlbptr,mctoken)

#define Cnmnams(nsfunc,nsptr,nsname,nsleng,nsclass)                     \
        Cnmnams(&Hlbptr,nsfunc,nsptr,nsname,nsleng,nsclass)

#define Cnmpool(spfunc,sptoken,spname,spleng,                           \
                sppricnt,spseccnt,spclass)                              \
        Cnmpool(&Hlbptr,spfunc,sptoken,spname,spleng,                   \
                sppricnt,spseccnt,spclass)

#define Cnmscop(sccmd,sckwd,scvalue)                                    \
        Cnmscop(&Hlbptr,sccmd,sckwd,scvalue)

#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)
```

DSICCNM

```
/********************************************************************/
/*                                                                  */
/*  NAME = DSICCNM                                                  */
/*                                                                  */
/*  DESCRIPTIVE NAME = HLL C Return Codes                           */
/*                                                                  */
/*  5665-362 for MVS/XA                                             */
/*  THIS PRODUCT CONTAINS                                           */
/*  "RESTRICTED MATERIAL OF IBM"                                    */
/*  (c) COPYRIGHT IBM CORP 1989                                     */
/*  ALL RIGHTS RESERVED                                             */
/*  LICENSED MATERIALS-PROPERTY OF IBM                              */
/*  REFER TO COPYRIGHT INSTRUCTION FORM                             */
/*  NUMBER G120-2083                                                */
/*                                                                  */
/*  STATUS = NetView Release 3                                      */
/*                                                                  */
/*  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
#define CNM_TIME_OUT            68
#define CNM_BAD_QUEUE           72
#define CNM_BAD_INDEX           76
#define CNM_QUEUE_EMPTY         80
#define CNM_BAD_ORIGBLOCK       84
#define CNM_BAD_LENGTH          88
#define CNM_NOT_MLWTO           92
#define CNM_BAD_LINETYPE        96
#define CNM_NOCUR_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_TVB_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_OPINPUT_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_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
#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_ZVSMS           100
```

Appendix D. C Samples

This appendix contains a table of the C samples that are shipped with NetView in SYS1.CNMSAMP. When data set names are referred to in this appendix, two names are given, such as CTMPPLT (CNMS4201). The first name is the alias name, and the name in parenthesis 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.

The following steps allow you to enter the member names as commands:

1. Compile and link edit the samples using the alias name.
2. Delete the (*) in column one of the appropriate CMDMDL statement in DSICMD to be able to execute the alais name as a command. No entries are needed in DSICMD for user exits.
3. NetView must be recycled to pick up the DSICMD changes.

Notes:

1. See the prologues of the samples for information about how certain samples are related and special cases for user 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 a user exit routine and two command processors.

C Samples Table

The following table refers to the C samples that are shipped with NetView. The table contains the function, the alias name, and the name of the member in SYS1.CNMSAMP.

Sample function	C sample	
	Alias	CNMSAMP
Template for commands and user exit routines	CTMPPLT	CNMS4201
Sample DSIEX03 to set a global variable	CEXIT3	CNMS4240
Uses Cnmsmsg to send data	CSNDDAT	CNMS4241
Uses WAIT FOR DATA	CWATDAT	CNMS4242
Sample DSIEX02A changes a WTO to an MLWTO	CEXIT2A	CNMS4243
Uses Cnmcnmi to forward RUs to a PU	CCNMI	CNMS4244
Uses Cnmkio for I/O to VSAM	CKEYIO	CNMS4245
HLL command using Cnmscop for scope checking	CSCOPCK	CNMS4246
Display full screen VIEW panel	CFLVIEW	CNMS4247
Activates LU and uses TRAP and WAIT to determine if activation is successful	CACTLU	CNMS4248
Uses Cnmsmsg to log text to a sequential log	CSEQLOG	CNMS4249
DST initialization exit for USERVSAM DST	CXITDI	CNMS4250
Primes VSAM empty data set for USERVSAM DST	CXITVN	CNMS4251
Sends a request to USERVSAM DST	CSNDDST	CNMS4252
Processes VSAM requests under USERVSAM DST	CDOVSAM	CNMS4253
Primes VSAM empty data set for CKEYIO	CPRIME	CNMS4254

C Samples Description

For each sample, a description of the function and the HLL service routines utilized are given.

CTMPPLT (CNMS4201)

This sample is a template for commands and user exit routines in C.

This sample is included in Chapter 9 on page 107

CEXIT3 (CNMS4240)

This is a sample DSIEX03 that sets a task global variable. This global variable will contain the value of the last time that a command other than CSNDDAT was entered under an OST. CWATDAT and CSNDDAT are used to interrogate this value.

The HLL service routines utilized in this sample are: *Cnminfc, Cnmvars*.

CSNDDAT (CNMS4241)

This sample uses *Cnmsmsg* to send data. The sample is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example 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 utilized in this sample are: *Cnmvars, Cnmsmsg, Cnminfc*.

CWATDAT (CNMS4242)

This sample uses WAIT FOR DATA. The sample is part of an example of sending messages with a type of request, waiting on the response, and parsing the results.

The purpose of the example 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 utilized in this sample are: *Cnmsmsg, Cnmcmd, Cnmgetd*.

CEXIT2A (CNMS4243)

This sample exit converts a WTO to an MLTWO by adding two lines to the single-line WTOs that are driving the exit.

The HLL service routines utilized in this sample are: *Cnmgetd, Cnmaltd*.

This sample is included in "Sample User Exit" on page 300.

CCNMI (CNMS4244)

This sample uses *Cnmcnmi* to forward RUs to a PU. NetView provides the *Cnmcnmi* service routine for use in communicating with devices in the network through the Communications Network Management Interface (CNMI). Any data that is returned may be accessed using the *Cnmgetd* service routine to retrieve records from the CNMI solicited data queue (CNMIQ).

This sample 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 prologue of the sample contains instructions for set up.

The HLL service routines utilized in this sample are: *Cnmcnmi, Cnmgetd, Cnmsmsg.*

CKEYIO (CNMS4245)

This sample illustrates how to code a NetView HLL command processor that allows I/O to a VSAM file via the *Cnmkio* routine. It must execute on a DST. To run this command on a DST, either use the EXCMD command or the *Cnmsmsg* service routine (with a type of COMMAND). The prologue of the sample explains how to set up a DST.

The HLL service routines utilized in this sample are: *Cnmkio, Cnmsmsg.*

CSCOPCK (CNMS4246)

This sample illustrates the scope checking capabilities provided by NetView. This sample scope checks keywords and values of the CSCOPCK command. In this sample, the user is required to set up the following elements for the command: operator ID, operator classes that can access the command, and operator profile. See the prologue of the sample for more information. This command yields a message informing the operator if he is not authorized to use the keyword and value specified when invoking the command.

The HLL service routines utilized in this sample are: *Cnmscop, Cnmsmsg.*

This sample is included in "Sample Command Processor for Scope Checking" on page 304.

CFLVIEW (CNMS4247)

This sample illustrates the usage of the full screen VIEW command processor.

The HLL service routines utilized in this sample are: *Cnmcmd, Cnmvars.*

CACTLU (CNMS4248)

This sample illustrates 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 utilized in this sample are: *Cnmcmd, Cnmgetd, Cnmsmsg.*

CSEQLOG (CNMS4249)

This sample uses *Cnmsmsg* to log text to a sequential log. The prologue of the sample contains instructions for set up.

The HLL service routines utilized in this sample are: *Cnminfc, Cnmsmsg.*

This sample is included in "Sample Command Processor for Sequential Logging" on page 309.

CXITDI (CNMS4250)

This sample is a DST initialization exit. The sample illustrates the DST initialization exit that is used by the USERVSAM DST.

The HLL service routines utilized in this sample are: *Cnmvars, Cnmsmsg.*

CXITVN (CNMS4251)

This sample primes a VSAM empty data set for the USERVSAM DST.

ͻSNDDST (CNMS4252)

This sample sends a 'PUT' or 'GET' request to the sample HLL Data Services Command Processor named 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 utilized in this sample are: *Cnmsmsg, Cnmvars, Cnmgetd, Cnmcmd.*

)VSAM (CNMS4253)

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 CSNDDST sample, and will write or read an 11 character value associated with an 11 character key to the sample DST's VSAM data set. The prologue of CDOVSAM contains instructions on installing the sample 'USERVSAM' Data Services Task.

The HLL service routines utilized in this sample are: *Cnmsmsg, Cnmkio.*

Ξ (CNMS4254)

This sample primes a VSAM empty data set for CKEYIO.

C Coded Samples

This section contains an example of a user exit routine and two command processors.

Sample User Exit

This sample is an example of user exit DSIEX02A.

```
/*********************************************************************/
/*                                                                   */
/*   (C) COPYRIGHT IBM CORP. 1989                                    */
/*                                                                   */
/* IEBCOPY   SELECT MEMBER=((CNMS4243,CEXIT2A,R))                    */
/*                                                                   */
/*   Descriptive Name: High Level Language C DSIEX02A Example        */
/*                                                                   */
/*      Function:                                                    */
/*                                                                   */
/*      This DSIEX02A adds two lines to the single line WTOs         */
/*      that are driving the exit.                                   */
/*                                                                   */
/*      Introduction:                                                */
/*                                                                   */
/*      To convert a single line WTO into a multi-line WTO,          */
/*      you must change the line types accordingly:                  */
/*                                                                   */
/*      In                   Out                                     */
/*      ---                  ---                                     */
/*      msg_type = "E", msg_type="=" line_type="C" (Control)         */
/*         and a            msg_type="=" line_type="D" (Data)        */
/*      line_type of " " msg_type="=" line_type="F" (Final)          */
/*                                                                   */
/*      Note:                                                        */
/*      A Msg_type of "E" implies that the message was externally    */
/*      generated via a WTO.  For example MVS D T would send         */
/*      a Msg_type of "E".  The Msg_type of "=" implies a user       */
/*      generated MLWTO was issued.                                  */
/*                                                                   */
/*   Dependencies: None                                              */
/*                                                                   */
/*   Restrictions: Only processes messages issued under              */
/*                 mainline processing.                              */
/*                                                                   */
/*   Language:  C                                                    */
/*                                                                   */
/*   Input:                                                          */
/*      1)  4-byte pointer to the HLB control block                  */
/*      2)  varying length character string of command (or message   */
/*          for user exits) that invoked this procedure              */
/*      3)  40-byte parameter list which describes the origin of     */
/*          the request that caused execution of this procedure.     */
/*                                                                   */
/*   Output:                                                        */
/*      Messages to various tasks.                                  */
/*                                                                  */
/*   Return Codes: returned in HLBRC                                */
/*      0 = normal exit                                             */
/*                                                                  */
/*   External Module References:  None                              */
/*                                                                  */
/*   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              */
Dsivarch *Cmdbuf;                /* Pointer to command buffer       */
Dsiorig  *Origblck;              /* Pointer to Orig block           */

main(int argc, char *argv??(??))
{
  /******************************************************************/
  /* Internal data definitions                                      */
  /******************************************************************/
  Dsivarch datain,               /* data returned by cnmgetd        */
           workbuf;              /* replacemnt buf passed to cnmaltd*/

  Dsiorig getblock;              /* orig block                      */
```

```
/********************************************************************/
/* Convert parameter pointers from character to hex addresses      */
/********************************************************************/

sscanf (argv??(1??),"%x",&Hlbptr);
sscanf(argv??(2??),"%x",&Cmdbuf);
sscanf(argv??(3??),"%x",&Origblck);

 .

/********************************************************************/
/* Initialization                                                   */
/********************************************************************/

/********************************************************************/
/*                                                                  */
/*  Customization starts here ...                                   */
/*                                                                  */
/********************************************************************/

/********************************************************************/
/*                                                                  */
/*  Execution                                                       */
/*                                                                  */
/********************************************************************/

Cnmgetd(PEEKLINE,              /* ...function is get a message    */
        &datain,               /* ...result goes here             */
        256,                   /* ...max input length             */
        &getblock,             /* ...must provide a work area     */
        IDATAQ,                /* ...get the first line of message*/
        1);                    /* ...get the first line of message*/

if (getblock.Orig_msg_type == 'E') /* WTO response to MVS command  */
  {
   getblock.Orig_msg_type = '=';  /* set msg type to MLWTO         */
   getblock.Orig_line_type = 'C'; /* set line type to control      */

   datain.buffer??(datain.size??) = '\0'; /* append null to data   */
                               /* ...retrieved by cnmgetd         */

                               /* put replacement buffer in       */
   Cnmvlc(&workbuf,            /* varying length character strng..*/
          0,                   /* do not convert to hex           */
          "Change WTO to MLWTO, WTO= '%s'",datain.buffer);
```

```
                             /* replace the text...            */
    Cnmaltd(REPLINE,         /* ...function is replace         */
            &workbuf,        /* ...text of new message         */
            &getblock,       /* ...use peeked Origin block     */
            IDATAQ,          /* ...initial data queue          */
            1);              /* ...replace the first line      */

    getblock.Orig_line_type = 'D'; /* set line type to data    */

                             /* put line to add to MLWTO in    */
    Cnmvlc(&workbuf,         /* varying length character strng..*/
           0,                /* do not convert to hex          */
           "Add a data line to MLWTO"); /* line to add         */

    Cnmaltd(INSLINE,         /* ...function is insert          */
            &workbuf,        /* ...text of new message         */
            &getblock,       /* ...use peeked Origin block     */
            IDATAQ,          /* ...initial data queue          */
            2);              /* ...add a line                  */

    getblock.Orig_line_type = 'F';/* set line type to final line   */

                             /* put line to add to MLWTO in    */
    Cnmvlc(&workbuf,         /* varying length character strng..*/
           0,                /* do not convert to hex          */
           "Add an end of MLWTO message"); /* line to add      */

    Cnmaltd(INSLINE,         /* ...function is insert          */
            &workbuf,        /* ...text of new message         */
            &getblock,       /* ...use peeked Origin block     */
            IDATAQ,          /* ...initial data queue          */
            3);              /* ...add a line                  */

    }
Hlbptr->Hlbrc == CNM_GOOD;
}
```

Sample Command Processor for Scope Checking

This sample is an example of a command processor for scope checking.

```
/*********************************************************************/
/*                                                                   */
/*   (C) COPYRIGHT IBM CORP. 1989                                    */
/*                                                                   */
/* IEBCOPY   SELECT MEMBER=((CNMS4246,CSCOPCK,R))                    */
/*                                                                   */
/*   Descriptive Name:High Level Language C Scope Check              */
/*                      Example                                      */
/*                                                                   */
/*   Function:                                                       */
/*                                                                   */
/* The following is an example of the scope checking capabilities    */
/* provided by NetView.  In this example, the user is required to    */
/* set up the following elements for the command (shown below):     */
/*  (1)  operator id                                                 */
/*  (2)  operator classes that can access the command                */
/*  (3)  operator profile                                            */
/*                                                                   */
/* The command gives the return code that the scope check service    */
/* routine returned to the operator.                                 */
/*                                                                   */
/* The syntax that this command checks for is:                       */
/*                                                                   */
/*     CSCOPCK PARMx(VALx)                                           */
/*                                                                   */
/* The following is the setup for the scope check example:           */
/*                                                                   */
/*   In DSIPARM(DSICMD):                                             */
/*              Define the operator classes that can access          */
/*                the command, its keywords, and its keyword values.*/
/*                                                                   */
/*              The example below says that the command CSCOPCK      */
/*                 can be executed by operators in scope class       */
/*                 1 and 2.  Scope class 1 can issue any keyword     */
/*                 or keyword value, but scope class 2 cannot use    */
/*                 the value of VAL1 with keyword PARM2, and scope   */
/*                 class 2 cannot issue PARM3 at all.                */
/*                                                                   */
/*   Example:                                                        */
/*                                                                   */
/*     CSCOPCK   CMDMDL    MOD=CSCOPCK,RES=N,TYPE=RD                 */
/*               CMDCLASS  1,2                                       */
/*     PARM2     KEYCLASS  1,2                                       */
/*     VAL1      VALCLASS  1                                         */
/*     PARM3     KEYCLASS  1                                         */
/*     VAL1      VALCLASS  1                                         */
/*                                                                   */
/*   In DSIPARM(DSIOPF)                                             */
/*     Define the operator ids and the profiles that the operator   */
/*       ids can use.                                               */
/*                                                                  */
/*   Example:                                                       */
/*     JOE      OPERATOR  PASSWORD=USER                             */
/*              PROFILEN  DSIPROF3                                  */
/*                                                                  */
/*   In DSIPRF(profilename)                                         */
/*     Define the operator class value that will correspond to the  */
/*     profile that the operator logs on with.                      */
/*                                                                  */
/*   Example:                                                       */
/*     In the DSIPRF dataset, member name DSIPROF3,                 */
/*                                                                  */
/*     DSIPROF3   PROFILE                                           */
/*                OPCLASS 3                                         */
/*                END                                               */
/*                                                                  */
/*   Restrictions:  None                                            */
/*                                                                  */
/*   Language: C                                                    */
/*                                                                  */
/*   Input:                                                         */
/*      1)  4-byte pointer to the HLB control block                 */
/*      2)  varying length character string of command (or message  */
/*          for user exits) that invoked this procedure             */
/*      3)  40-byte parameter list which describes the origin of    */
/*          the request that caused execution of this procedure.    */
/*                                                                  */
/*   Output:                                                        */
/*      Messages describing the scope of the operator.              */
/*                                                                  */
/*   Return Codes: returned in HLBRC                                */
/*      0 = normal exit                                             */
/*     -5 = cancelled                                               */
/*                                                                  */
/*   External Module References: None                               */
/*                                                                  */
/*   Change Activity:                                               */
/*      date,author: description of changes                         */
/********************************************************************/
#pragma runopts (NOEXECOPS,NOSTAE,NOSPIE,ISASIZE(4K),ISAINC(4K))
```

```
/*********************************************************************/
/* Standard include files
/*********************************************************************/
#include <string.h>        /* String functions            */
#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            */
Dsivarch *Cmdbuf;                /* Pointer to command buffer     */
Dsiorig  *Origblck;              /* Pointer to Origin block       */


main(int argc, char *argv??(??))
{
  /*******************************************************************/
  /* 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                      */


  /*******************************************************************/
  /* Convert parameter pointers from character to hex addresses     */
  /*******************************************************************/
  sscanf(argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

  /*******************************************************************/
  /*                                                                 */
  /*  Execution                                                      */
  /*                                                                 */
  /*******************************************************************/
```

```
/********************************************************************/
/*            Scan the keyword and the value                        */
/********************************************************************/
                                  /* Syntax of command is:   */
                                  /* CMDNAME KEYWORD(KEYVALUE)     */

                                  /* Parse the command buffer for: */

 token = strtok((char *) &(Cmdbuf->buffer)," ");  /* ...COMMAND   */
 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);
   }

 token = strtok(NULL,"(");        /* ...keyword                   */
 if (token != NULL)
   {
   len = strlen(token);            /* Get length of keyword        */
   strcpy(keyword,token);          /* Save keyword                 */
   if (len < 8)                    /* Pad with blanks?             */
      strncat(keyword,"        ",8 - len);
   }

 token = strtok(NULL,")");       /* ...value                      */
 if (token != NULL)
   {
   len = strlen(token);           /* Get length of keyvalue        */
   strcpy(keyvalue,token);        /* Save keyvalue                 */
   if (len < 8)                   /* Pad with blanks?              */
      strncat(keyvalue,"        ",8 - len);
   }

 if (token != NULL)               /* enough parms?                 */
                                  /* Scope check the input...      */
   Cnmscop(cmdname,               /* ...the command                */
           keyword,               /* ...the keyword                */
           keyvalue);             /* ...the value                  */
 else                             /* not enough parms specified    */
   Hlbptr->Hlbrc = CNM_BAD_INVOCATION;/* set bad rc          */
   /****************************************************************/
   /*        Inform user of the return code results...            */
   /****************************************************************/
    if (Hlbptr->Hlbrc == CNM_BAD_INVOCATION)
     {
       Cnmvlc(&msgbuf,0,"Not enough parms specified");
       Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
     }
    else
      if (Hlbptr->Hlbrc == CNM_GOOD)
       {
         Cnmvlc(&msgbuf,0,"Operator has passed scope checking");
         Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
       }
      else
        if (Hlbptr->Hlbrc == CNM_KEYWORD_NA)
         {
           Cnmvlc(&msgbuf,0,"Not authorized to use KEYWORD %s",
                            keyword);
           Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
         }
        else
          if (Hlbptr->Hlbrc == CNM_VALUE_NA)
           {
             Cnmvlc(&msgbuf,0,"Not authorized to use value %s",
                              keyvalue);
             Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
           }
          else
            {
              Cnmvlc(&msgbuf,0,"RC not recognized...%d",Hlbptr->Hlbrc);
              Cnmsmsg(&msgbuf,MSG,OPER,NULLCHAR);
            }
Hlbptr->Hlbrc == CNM_GOOD;          /* clear return code          */
}
```

Sample Command Processor for Sequential Logging

This sample is an example of a command processor to log text to a sequential log.

```
/*********************************************************************/
/*                                                                   */
/*   (C) COPYRIGHT IBM CORP. 1989                                    */
/*                                                                   */
/* IEBCOPY   SELECT MEMBER=((CNMS4249,CSEQLOG,R))                    */
/*                                                                   */
/*   Descriptive Name: High Level Language C sequential              */
/*                     Logging Example                               */
/*                                                                   */
/*      Function:                                                    */
/*                                                                   */
/*      Write the text passed to this command procedure via the      */
/*      command line to the log.                                     */
/*                                                                   */
/*   The syntax of this command is:                                  */
/*                                                                   */
/*      CSEQLOG logtext                                              */
/*                                                                   */
/*   Dependencies:  None                                             */
/*                                                                   */
/*   Restrictions: The length of the text to be logged combined      */
/*                 with the command name (CSEQLOG) can be no more    */
/*                 than 255 characters.                              */
/*                                                                   */
/*   Language:  C                                                    */
/*                                                                   */
/*   Iinstallation:                                                  */
/*                                                                   */
/*   (1)  ASSEMBLE AND LINKEDIT THIS MODULE AMODE=31,RMODE=ANY       */
/*        TYPE=RENT                                                  */
/*   (2)  ALLOC PRIMARY AND SECONDARY SEQUENTIAL DATA SET            */
/*   (3)  USE DD NMAES IN NETVIEW PROC OR THE ALLOCATE COMMAND       */
/*        TO ALLOCATE THE DATA SETS TO NETVIEW.                      */
/*        ALLOCATE THE DATA SETS AS                                  */
/*            SQLOGP & SQLOGS                                        */
/*   (4)  ADD THE FOLLOWING STATEMENT TO DSIDMN                      */
/*       TASK   MOD=DSIZDST,TSKID=SQLOGTSK,MEM=SQLOGMEM,PRI=3,INIT=Y*/
/*   (5)  ADD THE FOLLOWING MEMBER (SQLOGMEM) TO DSIPARM             */
/*         DSTINIT FUNCT=OTHER,DSRBO=1                               */
/*         DSTINIT PBSDN=SQLOGP                                      */
/*         DSTINIT SBSDN=SQLOGS                                      */
/*         LOGINIT AUTOFLIP=YES,RESUME=NO                            */
/*   (6)  ADD THE FOLLOWING CMDMDL TO DSICMD                         */
/*         CSEQLOG CMDMDL MOD=CSEQLOG,TYPE=R,RES=N                   */
/*                                                                   */
/*   Input:                                                            */
/*      1)  4-byte pointer to the HLB control block                    */
/*      2)  varying length character string of command (or message    */
/*          for user exits) that invoked this procedure                */
/*      3)  40-byte parameter list which describes the origin of       */
/*          the request that caused execution of this procedure.       */
/*                                                                     */
/*   Output:                                                           */
/*      Writes input to a sequential log                               */
/*                                                                     */
/*   Return Codes: returned in HLBRC                                   */
/*      0 = normal exit                                                */
/*     -5 = canceled                                                   */
/*                                                                     */
/*   External Module References:  None                                 */
/*                                                                     */
/*   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 <stdio.h>        /* Standard i/o library                      */
#include <stdarg.h>       /* Standard args                             */
#include <string.h>       /* Standard args                             */

#define DOMAIN   "DOMAIN  "
#define DATETIME "DATETIME"
#define OPID     "OPID    "
#define SQTASK   "SQLOGTSK"      /* sequential log task id             */
/***********************************************************************/
/* NetView high level language include files                           */
/***********************************************************************/
#include "dsic.h"         /* Include HLL macros                        */
```

```
/*********************************************************************/
/* External data definitions                                         */
/*********************************************************************/
Dsihlb   *Hlbptr;                /* Pointer to the HLB               */
Dsivarch *Cmdbuf;                /* Pointer to command buffer        */
Dsiorig  *Origblck;              /* Pointer to Orig block            */

main(int argc, char *argv??(??))
{
  /*******************************************************************/
  /* Internal data definitions                                       */
  /*******************************************************************/
  Dsivarch domain,                /* store domain name               */
           logtime,               /* store time                      */
           opid,                  /* store operator name             */
           logbfr,                /* store the buffer to be logged */
           message;               /* store error message             */

  char logtext??(256??),          /* store text passed on cmd line */
       *token,                    /* used to parse command buffer  */
       *textptr;

  int size;

  /*******************************************************************/
  /* Convert parameter pointers from character to hex addresses     */
  /*******************************************************************/
  sscanf (argv??(1??),"%x",&Hlbptr);
  sscanf(argv??(2??),"%x",&Cmdbuf);
  sscanf(argv??(3??),"%x",&Origblck);

  /*******************************************************************/
  /* Initialization                                                  */
  /*******************************************************************/

  /*******************************************************************/
  /*                                                                 */
  /*  Customization starts here ...                                  */
  /*                                                                 */
  /*******************************************************************/

  /*******************************************************************/
  /*                                                                 */
  /*  Execution                                                      */
  /*                                                                 */
  /*******************************************************************/
```

```
 /*******************************************************************/
 /*  Parse command buffer for text to log                           */
 /*******************************************************************/

 textptr = strchr((char *) &(Cmdbuf->buffer),' ');
 size = 0;

 if (textptr != NULL)          /* no text provided?                */
   {
   textptr++;
   size = Cmdbuf->size - (textptr - ((char *) &(Cmdbuf->buffer)));
   }

 if ((size > 0) && (Cmdbuf->size < 255))
  {                            /*if text provided and not too big*/

   Cnminfc(DOMAIN,&domain,8);   /* get the domain name             */
   Cnminfc(DATETIME,&logtime,8);/* get the time                    */
   Cnminfc(OPID,&opid,8);       /* get the opid                    */

   memmove(&logtext??(0??),&(domain.buffer),domain.size);
   logtext??(domain.size??) = ' ';
   memmove(&logtext??(domain.size+1??),&(logtime.buffer),logtime.size);
   logtext??(domain.size+logtime.size+1??) = ' ';
   memmove(&logtext??(domain.size+logtime.size+2??),
           &(opid.buffer),opid.size);
   logtext??(domain.size+logtime.size+opid.size+2??) = ' ';
   memmove(&logtext??(domain.size+logtime.size+opid.size+3??),
   textptr,size);

                                /* build the logtext in a varying  */
   Cnmnvlc(&logbfr,             /* length character string...      */
         0,                     /* ...do not convert to hex        */
         domain.size+logtime.size+opid.size+size+3, /* ...size     */
         logtext);              /* ...text to be logged            */

                                /* log the buffer...               */
   Cnmsmsg(&logbfr,             /* ...text to be logged            */
           MSG,                 /* ...type is message              */
           SEQLOG,              /* ...dest type is sequential log  */
           SQTASK);             /* ...task defined by SQTASK       */

/********************************************************************/
/*   Inform user of the return code results if Cnmsmsg fails        */
/********************************************************************/
```

```
      if (Hlbptr->Hlbrc ¬= CNM_GOOD)/*if the return code was not zero   */
       {
                                   /* build message in varying length */
       Cnmvlc(&message,            /* chararacter string...           */
              0,                   /* ...do not convert to hex        */
              "SLOG000 Error: Return code from Cnmsmsg = %d",
              Hlbptr->Hlbrc);      /* ...message text                 */

       Cnmsmsg(&message,           /* display message...              */
               MSG,                /* ...type is message              */
               OPER,               /* ...send to oper that issued slog*/
               NULLCHAR);          /* ...not used                     */
       }
     }
   else
     if (size <= 0)
       {                           /* No text to log                  */
                                   /* put error message in varying    */
       Cnmvlc(&message,            /* length character string...      */
              0,                   /* ...do not convert to hex        */
              "No text has been provided for logging"); /* message text*/

                                   /* display message...              */
       Cnmsmsg(&message,           /* ...message to display           */
               MSG,                /* ...type is message              */
               OPER,               /* ...send to invoking operator    */
               NULLCHAR);          /* ...not used                     */
     }
   else if (Cmdbuf->size > 255)
     {                             /* text too long                   */
                                   /* put error message in varying    */
     Cnmvlc(&message,              /* length character string...      */
            0,                     /* ...do not convert to hex        */
            "Text to be logged is too long");        /* message text*/

                                   /* display message...              */
     Cnmsmsg(&message,             /* ...message to display           */
             MSG,                  /* ...type is message              */
             OPER,                 /* ...send to invoking operator    */
             NULLCHAR);            /* ...not used                     */
    }
  }
```

Glossary, Bibliography, and Index

Glossary

This glossary defines important NCP, NetView, NetView/PC, SSP, and VTAM abbreviations and terms. It includes information from the *IBM Dictionary of Computing*, SC20-1699. Definitions from the *American National Dictionary for Information Processing* are identified by an asterisk (*). Definitions from draft proposals and working papers under development by the International Standards Organization, Technical Committee 97, Subcommittee 1 are identified by the symbol **(TC97)**. Definitions from the *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the Consultative Committee on International Telegraph and Telephone of the International Telecommunication Union, Geneva, 1980 are preceded by the symbol **(CCITT/ITU)**. Definitions from published sections of the *ISO Vocabulary of Data Processing*, developed by the International Standards Organization, Technical Committee 97, Subcommittee 1 and from published sections of the *ISO Vocabulary of Office Machines*, developed by subcommittees of ISO Technical Committee 95, are preceded by the symbol **(ISO)**.

For abbreviations, the definition usually consists only of the words represented by the letters; for complete definitions, see the entries for the words.

Reference Words Used in the Entries

The following reference words are used in this glossary:

Deprecated term for. Indicates that the term should not be used. It refers to a preferred term, which is defined.

Synonymous with. Appears in the commentary of a preferred term and identifies less desirable or less specific terms that have the same meaning.

Synonym for. Appears in the commentary of a less desirable or less specific term and identifies the preferred term that has the same meaning.

Contrast with. Refers to a term that has an opposed or substantively different meaning.

See. Refers to multiple-word terms that have the same last word.

See also. Refers to related terms that have similar (but not synonymous) meanings.

abend. Abnormal end of task.

abnormal end of task (abend). Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

ACB. (1) In VTAM, access method control block. (2) In NCP, adapter control block.

ACB name. (1) The name of an ACB macroinstruction. (2) A name specified in the ACBNAME parameter of a VTAM APPL statement. Contrast with *network name*.

accept. For a VTAM application program, to establish a session with a logical unit (LU) in response to a CINIT request from a system services control point (SSCP). The session-initiation request may begin when a terminal user logs on, a VTAM application program issues a macroinstruction, or a VTAM operator issues a command. See also *acquire (1)*.

access method control block (ACB). A control block that links an application program to VSAM or VTAM.

accounting exit routine. In VTAM, an optional installation exit routine that collects statistics about session initiation and termination.

ACF/NCP. Advanced Communications Function for the Network Control Program. Synonym for *NCP*.

ACF/SSP. Advanced Communications Function for the System Support Programs. Synonym for *SSP*.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method. Synonym for *VTAM*.

acquire. (1) For a VTAM application program, to initiate and establish a session with another logical unit (LU). The acquire process begins when the application program issues a macroinstruction. See also *accept*. (2) To take over resources that were formerly controlled by an access method in another domain, or to resume control of resources that were controlled by this domain but released. Contrast with *release*. See also *resource takeover*.

activate. To make a resource of a node ready to perform the functions for which it was designed. Contrast with *deactivate*.

active. (1) The state a resource is in when it has been activated and is operational. Contrast with *inactive, pending,* and *inoperative*.. (2) Pertaining to a major or minor node that has been activated by VTAM. Most resources are activated as part of VTAM start processing or as the result of a VARY ACT command.

adapter control block (ACB). In NCP, a control block that contains line control information and the states of I/O operations for BSC lines, SS lines, or SDLC links.

adaptive session pacing. Synonym for *adaptive session-level pacing.*

adaptive session-level pacing. A form of session-level pacing in which session components exchange pacing windows that may vary in size during the course of a session. This allows transmission to adapt dynamically to variations in availability and demand of buffers on a session by session basis. Session pacing occurs within independent stages along the session path according to local congestion at the intermediate nodes. Synonymous with *adaptive session pacing*. See *pacing*, *session-level pacing*, and *virtual route pacing.*

alert. (1) In SNA, a record sent to a system problem management focal point to communicate the existence of an alert condition. (2) In the NetView program, a high priority event that warrants immediate attention. This data base record is generated for certain event types that are defined by user-constructed filters.

alias name. A name defined in a host used to represent a logical unit name, logon mode table name, or class-of-service name in another network. This name is defined to a name translation program when the alias name does not match the real name. The alias name translation program is used to associate the real and alias names.

allocate. A logical unit (LU) 6.2 application program interface (API) verb used to assign a session to a conversation for the conversation's use. Contrast with *deallocate*.

API. Application program interface.

application program. (1) A program written for or by a user that applies to the user's work. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application program interface (API). (1) The formally defined programming language interface between an IBM system control program or licensed program and its user. (2) The interface through which an application program interacts with an access method. In VTAM, it is the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

attaching device. Any device that is physically connected to a network and can communicate over the network.

authorization exit routine. In VTAM, an optional installation exit routine that approves or disapproves requests for session initiation.

authorized receiver. In the NetView program, an authorized operator who receives all the unsolicited and authorized-receiver messages not assigned to a specific operator.

automatic logon. (1) A process by which VTAM automatically creates a session-initiation request to establish a session between two logical units (LUs). The session will be between a designated primary logical unit (PLU) and a secondary logical unit (SLU) that is neither queued for nor in session with another PLU. See also *controlling application program* and *controlling logical unit.* (2) In VM, a process by which a virtual machine is initiated by other than the user of that virtual machine. For example, the primary VM operator's virtual machine is activated automatically during VM initialization.

available. In VTAM, pertaining to a logical unit that is active, connected, enabled, and not at its session limit.

begin bracket. In SNA, the value (binary 1) of the begin-bracket indicator in the request header (RH) of the first request in the first chain of a bracket; the value denotes the start of a bracket. Contrast with *end bracket*. See also *bracket*.

BIU segment. In SNA, the portion of a basic information unit (BIU) that is contained within a path information unit (PIU). It consists of either a request/response header (RH) followed by all or a portion of a request/response unit (RU), or only a portion of an RU.

blocking of PIUs. In SNA, an optional function of path control that combines multiple path information units (PIUs) into a single basic transmission unit (BTU).

boundary function. (1) A capability of a subarea node to provide protocol support for attached peripheral nodes, such as: (a) interconnecting subarea path control and peripheral path control elements, (b) performing session sequence numbering for low-function peripheral nodes, and (c) providing session-level pacing support. (2) The component that provides these capabilities. See also *boundary node*, *network addressable unit (NAU)*, *peripheral path control*, *subarea node*, and *subarea path control*.

boundary node. (1) A subarea node with boundary function. See *subarea node* (including illustration). See also *boundary function*. (2) The programming component that performs FID2 (format identification type 2) conversion, channel data link control, pacing, and channel or device error recovery procedures for a locally attached station. These functions are similar to those performed by a network control program for an NCP-attached station.

bracket. In SNA, one or more chains of request units (RUs) and their responses that are exchanged between the two LU-LU half-sessions and that represent a transaction between them. A bracket must be completed before another bracket can be started. Examples of brackets are data base inquiries/replies, update transactions, and remote job entry output sequences to work stations. See also *begin bracket* and *end bracket*.

browse. A way of looking at a file that does not allow you to change it.

buffer. A portion of storage for temporarily holding input or output data.

call. (1) * (ISO) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (2) To transfer control to a procedure, program, routine, or subroutine. (3) The actions necessary to make a connection between two stations. (4) To attempt to contact a user, regardless of whether the attempt is successful.

CALLOUT. The logical channel type on which the data terminal equipment (DTE) can send a call, but cannot receive one.

calling. * (ISO) The process of transmitting selection signals in order to establish a connection between data stations.

CCP. Configuration control program facility.

CDRM. Cross-domain resource manager.

CDRSC. Cross-domain resource.

chain. (1) A group of logically linked records. (2) See *RU chain*.

channel. * A path along which signals can be sent, for example, data channel, output channel. See *data channel* and *input/output channel*. See also *link*.

channel-attached. (1) Pertaining to the attachment of devices directly by input/output channels to a host processor. (2) Pertaining to devices attached to a controlling unit by cables, rather than by telecommunication lines. Contrast with *link-attached*. Synonymous with *local*.

character-coded. Synonym for *unformatted*.

class of service (COS). In SNA, a designation of the path control network characteristics, such as path security, transmission priority, and bandwidth, that apply to a particular session. The end user designates class of service at session initiation by using a symbolic name that is mapped into a list of virtual routes, any one of which can be selected for the session to provide the requested level of service.

cleanup. A network services request, sent by a system services control unit (SSCP) to a logical unit (LU), that causes a particular LU-LU session with that LU to be ended immediately and without the participation of either the other LU or its SSCP.

CLIST. Command list.

CNM. Communication network management.

code point. In the NetView/PC program and in the NetView program, a 1- or 2-byte hexadecimal value that indexes a text string stored at an alert receiver and is used by the alert receiver to create displays of alert information.

command. (1) A request from a terminal for the performance of an operation or the execution of a particular program. (2) In SNA, any field set in the transmission header (TH), request header (RH), and sometimes portions of a request unit (RU), that initiates an action or that begins a protocol; for example: (a) Bind Session (session-control request unit), a command that activates an LU-LU session, (b) the change-direction indicator in the RH of the last RU of a chain, (c) the virtual route reset window indicator in a FID4 transmission header. See also *VTAM operator command*.

command facility. The component of the NetView program that is a base for command processors that can monitor, control, automate, and improve the operation of a network.

command list. A list of commands and statements designed to perform a specific function for the user. Command lists can be written in REXX or in NetView command list language.

command procedure. Either a command processor written in a high-level language (HLL) or a command list. See also *command list* and *command processor*.

command processor. (1) A program that performs an operation specified by a command. (2) In the NetView program, a user-written module designed to perform a specific function. Command processors, which can be written in assembler or a high-level language (HLL), are invoked as commands.

communication line. Deprecated term for *telecommunication line* and *transmission line*.

communication management configuration host node. The type 5 host processor in a communication management configuration that does all network-control functions in the network except for the control of devices channel-attached to data hosts. Synonymous with *communication management host*. Contrast with *data host node*.

communication management host. Synonym for *communication management configuration host node*. Contrast with *data host*.

communication network management (CNM). The process of designing, installing, operating, and managing the distribution of information and controls among end users of communication systems.

communication network management (CNM) application program. A VTAM application program that issues and receives formatted management services request units for physical units. For example, the NetView program.

communication network management (CNM) interface. The interface that the access method provides to an application program for handling data and commands associated with communication system management. CNM data and commands are handled across this interface.

communication network management (CNM) processor. A program that manages one of the functions of a communications system. A CNM processor is executed under control of the NetView program.

component. A command that (a) controls the terminal's screen (using the DSIPSS macro (TYPE=ASYPANEL) or the VIEW command), (b) allows the operator to enter NetView commands, and (c) can resume when such commands are complete.

composite end node (CEN). A group of nodes made up of a single type 5 node and its subordinate type 4 nodes that together support type 2.1 protocols. To a type 2.1 node, a CEN appears as one end node. For example, NCP and VTAM act as a composite end node.

configuration. (1) (TC97) The arrangement of a computer system or network as defined by the nature, number, and the chief characteristics of its functional units. The term may refer to a hardware or a software configuration. (2) The devices and programs that make up a system, subsystem, or network. (3) In CCP, the arrangement of controllers, lines, and terminals attached to an IBM 3710 Network Controller. Also, the collective set of item definitions that describe such a configuration.

configuration control program (CCP) facility. An SSP interactive application program facility by which configuration definitions for the IBM 3710 Network Controller can be created, modified, and maintained.

configuration restart. In SNA, one of the types of network services in the control point (CP) and in the physical unit (PU); configuration services activate, deactivate, and maintain the status of physical units, links, and link stations. Configuration services also shut down and restart network elements and modify path control routing tables and address-translation tables. See also *maintenance services*, *management services*, *network services*, and *session services*.

connection. Synonym for *physical connection*.

control block. (1) (ISO) A storage area used by a computer program to hold control information. (2) In the IBM Token-Ring Network, a specifically formatted block of information provided from the application program to the Adapter Support Interface to request an operation.

control program (CP). The VM operating system that manages the real processor's resources and is responsible for simulating System/370s for individual users.

controlling application program. In VTAM, an application program with which a secondary logical unit (other than an application program) is automatically put in session whenever the secondary logical unit is available. See also *automatic logon* and *controlling logical unit*.

controlling logical unit. In VTAM, a logical unit with which a secondary logical unit (other than an application program) is automatically put in session whenever the secondary logical unit is available. A controlling logical unit can be either an application program or a device-type logical unit. See also *automatic logon* and *controlling application program*.

converted command. An intermediate form of a character-coded command produced by VTAM through use of an unformatted system services definition table. The format of a converted command is fixed; the unformatted system services definition table must be constructed in such a manner that the character-coded command (as entered by a logical unit) is converted into the predefined, converted command format. See also *unformatted*.

COS. Class of service.

cross-domain. In SNA, pertaining to control of resources involving more than one domain.

cross-domain resource (CDRSC). A resource owned by a cross-domain resource manager (CDRM) in another domain but known by the CDRM in this domain by network name and associated CDRM.

cross-domain resource manager (CDRM). In VTAM, the function in the system services control point (SSCP) that controls initiation and termination of cross-domain sessions.

data channel. Synonym for *input/output channel*. See *channel*.

data flow control (DFC) layer. In SNA, the layer within a half-session that (1) controls whether the half-session can send, receive, or concurrently send and receive request units (RUs); (2) groups related RUs into RU chains; (3) delimits transactions via the bracket protocol; (4) controls the interlocking of requests and responses in accordance with control modes specified at session activation; (5) generates sequence numbers; and (6) correlates requests and responses.

data host. Synonym for *data host node*. Contrast with *communication management configuration host*.

data host node. In a communication management configuration, a type 5 host node that is dedicated to processing applications and does not control network resources, except for its channel-attached or communication adapter-attached devices. Synonymous with *data host*. Contrast with *communication management configuration host node*.

data link. In SNA, synonym for *link*.

data link control (DLC) layer. In SNA, the layer that consists of the link stations that schedule data transfer over a transmission medium connecting two nodes and perform error control for the link connection. Examples of data link control are SDLC for serial-by-bit link connection and data link control for the System/370 channel.

data services command processor (DSCP). A component that structures a request for recording and retrieving data in the application program's data base and for soliciting data from a device in the network.

data services manager (DSM). A function in the NetView program that provides VSAM services for data storage and retrieval.

data services task (DST). The NetView subtask that gathers, records, and manages data in a VSAM file and/or a network device that contains network management information.

data set. The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

ddname. Data definition name.

deactivate. To take a resource of a node out of service, rendering it inoperable, or to place it in a state in which it cannot perform the functions for which it was designed. Contrast with *activate*.

deallocate. A logical unit (LU) 6.2 application program interface (API) verb that terminates a conversation, thereby freeing the session for a future conversation. Contrast with *allocate*.

definite response (DR). In SNA, a value in the form-of-response-requested field of the request header. The value directs the receiver of the request to return a response unconditionally, whether positive or negative, to that request. Contrast with *exception response* and *no response*.

definition statement. (1) In VTAM, the statement that describes an element of the network. (2) In NCP, a type of instruction that defines a resource to the NCP. See Figure 13, Figure 14, and Figure 15. See also *macroinstruction*.

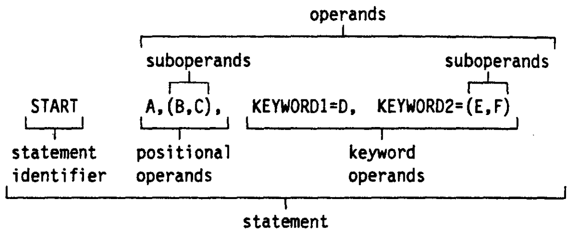


Figure 13. Example of a Language Statement

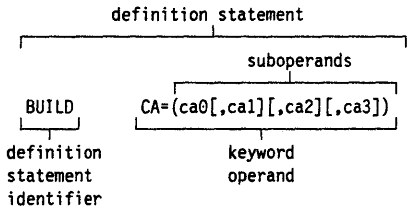


Figure 14. NCP Examples

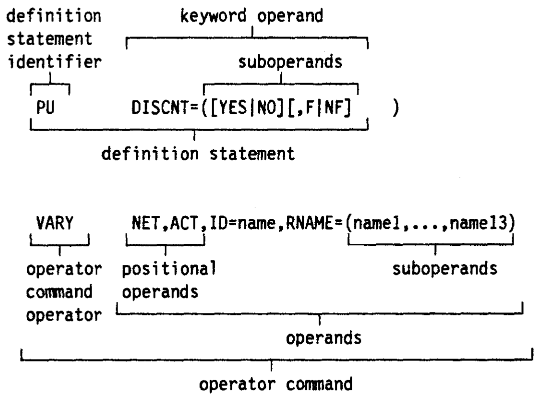

definition statement identifier
keyword operand
suboperands
PU
DISCNT=([YES|NO][,F|NF])
definition statement
VARY
NET,ACT,ID=name,RNAME=(name1,...,name13)
operator command operator
positional operands
suboperands
operands
operator command

Figure 15. VTAM Examples

device. An input/output unit such as a terminal, display, or printer. See *attaching device*.

directory. In VM, a control program (CP) disk that defines each virtual machine's normal configuration.

display. (1) To present information for viewing, usually on a terminal screen or a hard-copy device. (2) A device or medium on which information is presented, such as a terminal screen. (3) Deprecated term for *panel*.

domain. (1) An access method, its application programs, communication controllers, connecting lines, modems, and attached terminals. (2) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests. See *system services control point domain* and *type 2.1 node control point domain.*. See also *single-domain network* and *multiple-domain network*.

domain operator. In a multiple-domain network, the person or program that controls the operation of the resources controlled by one system services control point. Contrast with *network operator* (2).

double-byte character set (DBCS). A character set, such as Japanese, in which each character is represented by a two-byte code.

downstream. In the direction of data flow from the host to the end user. Contrast with *upstream*.

drop. In the IBM Token-Ring Network, a cable that leads from a faceplate to the to the distribution panel in a wiring closet. When the IBM Cabling System is used with the IBM Token-Ring Network, a drop may form part of a lobe.

DSCP. Data services command processor.

DSM. Data services manager.

DST. Data services task.

dump. (1) Computer printout of storage. (2) To write the contents of all or part of storage to an external medium as a safeguard against errors or in connection with debugging. (3) (ISO) Data that have been dumped.

EBCDIC. * Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

echo. The return of characters to the originating SS device to verify that a message was sent correctly.

ED. Enciphered data.

element. (1) A field in the network address. (2) The particular resource within a subarea identified by the element address. See also *subarea*.

Emulation Program (EP). An IBM control program that allows a channel-attached 3705 or 3725 communication controller to emulate the functions of an IBM 2701 Data Adapter Unit, an IBM 2702 Transmission Control, or an IBM 2703 Transmission Control. See also *network control program*.

enciphered data (ED). Data whose meaning is concealed from unauthorized users.

end bracket. In SNA, the value (binary 1) of the end bracket indicator in the request header (RH) of the first request of the last chain of a bracket; the value denotes the end of the bracket. Contrast with *begin bracket*. See also *bracket*.

end node. A type 2.1 node that does not provide any intermediate routing or session services to any other node. For example, APPC/PC is an end node. See *composite end node*, *node*, and *type 2.1 node*.

entry point. An SNA node that provides distributed network management support. It may be a type 2, type 2.1, type 4, or type 5 node. It sends SNA-formatted network management data about itself and the resources it controls to a focal point for centralized processing, and it receives and executes focal point initiated commands to manage and control its resources.

EP. Emulation Program.

ER. (1) Explicit route. (2) Exception response.

ESTAE. Extended specify task abnormal exit.

event. (1) In the NetView program, a record indicating irregularities of operation in physical elements of a network. (2) An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as an input/output operation.

exception response (ER). In SNA, a value in the form-of-response-requested field of a request header (RH). An exception response is sent only if a request is unacceptable as received or cannot be processed. Contrast with *definite response* and *no response*. See also *negative response*.

EXEC. In a VM operating system, a user-written command file that contains CMS commands, other user-written commands, and execution control statements, such as branches.

exit routine. Any of several types of special-purpose user-written routines. See *accounting exit routine, authorization exit routine, logon-interpret routine, virtual route selection exit routine, EXLST exit routine,* and *RPL exit routine*.

EXLST exit routine. In VTAM, a routine whose address has been placed in an exit list (EXLST) control block. The addresses are placed there with the EXLST macroinstruction, and the routines are named according to their corresponding operand; hence DFASY exit routine, TPEND exit routine, RELREQ exit routine, and so forth. All exit list routines are coded by the VTAM application programmer. Contrast with *RPL exit routine*.

explicit route (ER). In SNA, the path control network elements, including a specific set of one or more transmission groups, that connect two subarea nodes. An explicit route is identified by an origin subarea address, a destination subarea address, an explicit route number, and a reverse explicit route number. Contrast with *virtual route (VR)*. See also *path* and *route extension*.

extended specify task abnormal exit (ESTAE). An MVS macroinstruction that provides recovery capability and gives control to the user-specified exit routine for processing, diagnosing an abend, or specifying a retry address.

field-formatted. Pertaining to a request or response that is encoded into fields, each having a specified format such as binary codes, bit-significant flags, and symbolic names. Contrast with *character-coded*.

flow control. In SNA, the process of managing the rate at which data traffic passes between components of the network. The purpose of flow control is to optimize the rate of flow of message units, with minimum congestion in the network; that is, to neither overflow the buffers at the receiver or at intermediate routing nodes, nor leave the receiver waiting for more message units. See also *adaptive session-level pacing, pacing, session-level pacing*, and *virtual route pacing*.

focal point. An entry point that provides centralized management and control for other entry points for one or more network management categories.

formatted system services. A portion of VTAM that provides certain system services as a result of receiving a field-formatted command, such as an Initiate or Terminate command. Contrast with *unformatted system services (USS)*. See also *field-formatted*.

frame. (1) The unit of transmission in some local area networks, including the IBM Token-Ring Network. It includes delimiters, control characters, information, and checking characters. (2) In SDLC, the vehicle for every command, every response, and all information that is transmitted using SDLC procedures.

full-screen mode. A form of panel presentation in the NetView program where the contents of an entire terminal screen can be displayed at once. Full-screen mode can be used for fill-in-the-blanks prompting. Contrast with *line mode*.

generation. The process of assembling and link editing definition statements so that resources can be identified to all the necessary programs in a network.

generic alert. Encoded alert information that uses code points (defined by IBM and possibly customized by users or application programs) stored at an alert receiver, such as the NetView program.

group. In the NetView/PC program, to identify a set of application programs that are to run concurrently.

half-session. In SNA, a component that provides function management data (FMD) services, data flow control, and transmission control for one of the sessions of a network addressable unit (NAU). See also *primary half-session* and *secondary half-session*.

hard copy. A printed copy of machine output in a visually readable form; for example, printed reports, listings, documents, summaries, or network logs.

hardware monitor. The component of the NetView program that helps identify network problems, such as hardware, software, and microcode, from a central control point using interactive display techniques.

help panel. An online display that tells you how to use a command or another aspect of a product. See *task panel*.

high-level language (HLL). A programming language that does not reflect the structure of any particular computer or operating system. For NetView Release 3, the high-level languages are PL/I and C.

hierarchy. In the NetView program, the resource types, display types, and data types that make up the organization, or levels, in a network.

host node. A node providing an application program interface (API) and a common application interface. See *boundary node*, *node*, *peripheral node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

immediate command. In the NetView program, a command (such as GO, CANCEL, or RESET) that can be executed while a regular command is being processed.

inactive. Describes the state of a resource that has not been activated or for which the VARY INACT command has been issued. Contrast with *active*. See also *inoperative*.

information (I) format. A format used for information transfer.

initiate. A network services request sent from a logical unit (LU) to a system services control point (SSCP) requesting that an LU-LU session be established.

inoperative. The condition of a resource that has been active, but is not. The resource may have failed, received an INOP request, or is suspended while a reactivate command is being processed. See also *inactive*.

input/output channel. (1) (ISO) In a data processing system, a functional unit that handles the transfer of data between internal and peripheral equipment. (2) In a computing system, a functional unit, controlled by a processor, that handles the transfer of data between processor storage and local peripheral devices. Synonymous with *data channel*. See *channel*. See also *link*.

interface. * A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs.

item. In CCP, any of the components, such as communication controllers, lines, cluster controllers, and terminals, that comprise an IBM 3710 Network Controller configuration.

JCL. Job control language.

job control language (JCL). * A problem-oriented language designed to express statements in a job that are used to identify the job or describe its requirements to an operating system.

Kanji. An ideographic character set used in Japanese. See also *double-byte character set*.

keyword. (1) (TC97) A lexical unit that, in certain contexts, characterizes some language construction. (2) * One of the predefined words of an artificial language. (3) One of the significant and informative words in a title or document that describes the content of that document. (4) A name or symbol that identifies a parameter. (5) A part of a command operand that consists of a specific character string (such as DSNAME=). See also *definition statement* and *keyword operand*. Contrast with *positional operand*.

keyword operand. An operand that consists of a keyword followed by one or more values (such as DSNAME=HELLO). See also *definition statement*. Contrast with *positional operand*.

keyword parameter. A parameter that consists of a keyword followed by one or more values.

LCS. Link connection subsystem.

line. See *communication line*.

line mode. A form of screen presentation in which the information is presented a line at a time in the message area of the terminal screen. Contrast with *full-screen mode*.

link. In SNA, the combination of the link connection and the link stations joining network nodes; for example: (1) a System/370 channel and its associated protocols, (2) a serial-by-bit connection under the control of Synchronous Data Link Control (SDLC). A link connection is the physical medium of transmission. A link, however, is both logical and physical. Synonymous with *data link*. See Figure 16 on page 325.

link-attached. Pertaining to devices that are physically connected by a telecommunication line. Contrast with *channel-attached*. Synonymous with *remote*.

link connection subsystem (LCS). The sequence of link connection components (LCCs) that belong to a link connection and are managed by one LCSM.

load module. (ISO) A program unit that is suitable for loading into main storage for execution; it is usually the output of a linkage editor.

local. Pertaining to a device that is attached to a controlling unit by cables, rather than by a telecommunication line. Synonymous with *channel-attached*.

local address. In SNA, an address used in a peripheral node in place of an SNA network address and transformed to or from an SNA network address by the boundary function in a subarea node.

logical unit (LU). In SNA, a port through which an end user accesses the SNA network and the functions provided by system services control points (SSCPs). An LU can support at least two sessions—one with an SSCP and one with another LU—and may be capable of supporting many sessions with other LUs. See also *network addressable unit (NAU)*, *peripheral LU*, *physical unit (PU)*, *system services control point (SSCP)*, *primary logical unit (PLU)*, and *secondary logical unit (SLU)*.

logical unit (LU) services. In SNA, capabilities in a logical unit to: (1) receive requests from an end user and, in turn, issue requests to the system services control point (SSCP) in order to perform the requested functions, typically for session initiation; (2) receive requests from the SSCP, for example to activate LU-LU sessions via Bind Session requests; and (3) provide session presentation and other services for LU-LU sessions. See also *physical unit (PU) services*.

logical unit (LU) 6.2. A type of logical unit that supports general communication between programs in a distributed processing environment. LU 6.2 is characterized by (1) a peer relationship between session partners, (2) efficient utilization of a session for multiple transactions, (3) comprehensive end-to-end error processing, and (4) a generic application program interface (API) consisting of structured verbs that are mapped into a product implementation.

logmode table. Synonym for *logon mode table*.

logoff. In VTAM, an unformatted session termination request.

logon. In VTAM, an unformatted session initiation request for a session between two logical units. See

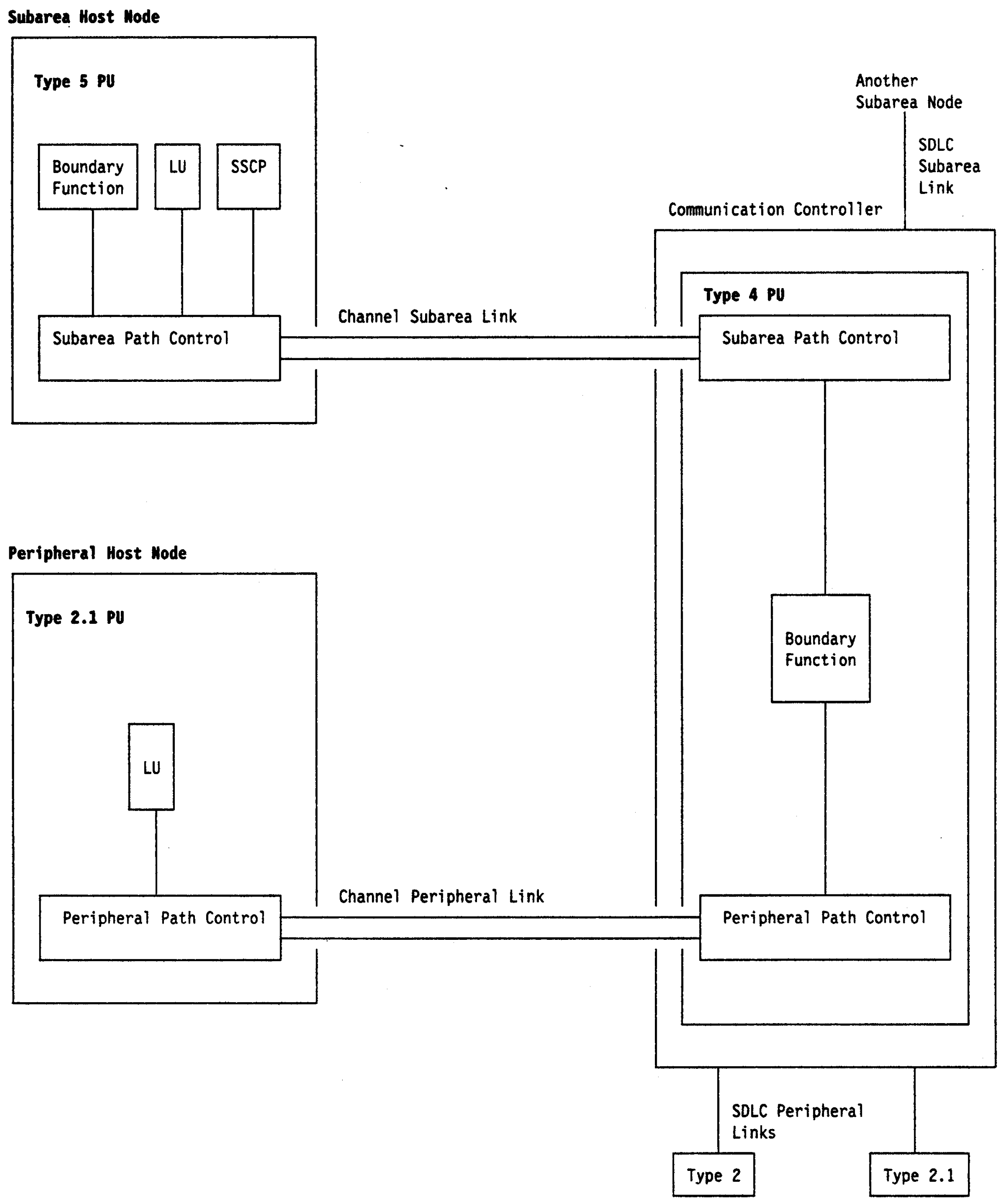


Figure 16. Links and Path Controls

automatic logon and *simulated logon*. See also *session-initiation request*.

logon mode table. In VTAM, a set of entries for one or more logon modes. Each logon mode is identified by a logon mode name. Synonymous with *logmode table*.

logon-interpret routine. In VTAM, an installation exit routine, associated with an interpret table entry, that translates logon information. It may also verify the logon.

LU. Logical unit.

LU type. In SNA, the classification of an LU-LU session in terms of the specific subset of SNA protocols and options supported by the logical units (LUs) for that session, namely:

The mandatory and optional values allowed in the session activation request.

The usage of data stream controls, function management headers (FMHs), request unit (RU) parameters, and sense codes.

Presentation services protocols such as those associated with FMH usage.

LU types 0, 1, 2, 3, 4, 6.1, 6.2, and 7 are defined.

LU-LU session. In SNA, a session between two logical units (LUs) in an SNA network. It provides communication between two end users, or between an end user and an LU services component.

LU-LU session type. A deprecated term for *LU type*.

LU 6.2. Logical unit 6.2.

macroinstruction. (1) An instruction that when executed causes the execution of a predefined sequence of instructions in the same source language. (2) In assembler programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macro definition. The statements normally produced from the macro definition replace the macroinstruction in the program. See also *definition statement*.

maintenance services. In SNA, one of the types of network services in system services control points (SSCPs) and physical units (PUs). Maintenance services provide facilities for testing links and nodes and for collecting and recording error information. See also *configuration services*, *management services*, *network services*, and *session services*.

major node. In VTAM, a set of resources that can be activated and deactivated as a group. See *node* and *minor node*.

management services. In SNA, one of the types of network services in control points (CPs) and physical units (PUs). Management services are the services provided to assist in the management of SNA networks, such as problem management, performance and accounting management, configuration management and change management. See also *configuration services*, *maintenance services*, *network services*, and *session services*.

Medium Access Control (MAC). The sublayer of DLC that supports medium-dependent functions and uses the services of the physical layer to provide services to Logical Link Control (LLC). The MAC sublayer includes the medium access port.

medium access control (MAC) procedure. (TC97) In a local area network, the part of the protocol that governs access to the transmission medium independently of the physical characteristics of the medium, but takes into account the topological aspects of the network, in order to enable the exchange of data between data stations.

message. (1) (TC97) A group of characters and control bit sequences transferred as an entity. (2) In VTAM, the amount of function management data (FMD) transferred to VTAM by the application program with one SEND request.

migration. Installing a new version or release of a program when an earlier version or release is already in place.

minor node. In VTAM, a uniquely-defined resource within a major node. See *node* and *major node*.

module. * A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine.

monitor. In the IBM Token-Ring Network, the function required to initiate the transmission of a token on the ring and to provide soft-error recovery in case of lost tokens, circulating frames, or other difficulties. The capability is present in all ring stations.

multiple-domain network. In SNA, a network with more than one system services control point (SSCP). Contrast with *single-domain network*.

Multiple Virtual Storage (MVS). An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for System/370. It is a software operating system controlling the execution of programs.

Multiple Virtual Storage for Extended Architecture (MVS/XA). An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for Extended Architecture. Extended architecture allows 31-bit storage addressing. MVS/XA is a software operating system controlling the execution of programs.

MVS. Multiple Virtual Storage operating system.

MVS/XA. Multiple Virtual Storage for Extended Architecture operating system.

NAU. Network addressable unit.

NC. Network control.

NCCF. Network Communications Control Facility.

NCP. (1) Network Control Program (IBM licensed program). Its full name is Advanced Communications Function for the Network Control Program. Synonymous with *ACF/NCP*. (2) Network control program (general term).

negative response (NR). In SNA, a response indicating that a request did not arrive successfully or was not processed successfully by the receiver. Contrast with *positive response*. See *exception response*.

NetView. A system 370-based IBM licensed program used to monitor a network, manage it, and diagnose its problems.

NetView command list language. An interpretive language unique to the NetView program that is used to write command lists.

NetView-NetView task (NNT). The task under which a cross-domain NetView operator session runs. See *operator station task*.

NetView/PC. A PC-based IBM licensed program through which application programs can be used to monitor, manage, and diagnose problems in IBM Token-Ring networks, non-SNA communication devices, and voice networks.

network. (1) (TC97) An interconnected group of nodes. (2) In data processing, a user application network. See *path control network, public network, SNA network, subarea network, type 2.1 network*, and *user-application network*.

network address. In SNA, an address, consisting of subarea and element fields, that identifies a link, a link station, or a network addressable unit. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See *local address*. See also *network name*.

network addressable unit (NAU). In SNA, a logical unit, a physical unit, or a system services control point. It is the origin or the destination of information transmitted by the path control network. Each NAU has a network address that represents it to the path control network. See also *network name, network address*, and *path control network*.

Network Communications Control Facility (NCCF). An IBM licensed program that is a base for command processors that can monitor, control, automate, and improve the operations of a network. Its function is included and enhanced in NetView's command facility.

network control (NC). In SNA, an RU category used for requests and responses exchanged for such purposes as activating and deactivating explicit and virtual routes and sending load modules to adjacent peripheral nodes. See also *data flow control layer* and *session control*.

Network Control Program (NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Its full name is Advanced Communications Function for the Network Control Program.

network control program. A program, generated by the user from a library of IBM-supplied modules, that controls the operation of a communication controller.

network log. A file that contains all messages processed by the NetView program.

network management vector transport (NMVT). A management services request/response unit (RU) that flows over an active session between physical unit management services and control point management services (SSCP-PU session).

network name. (1) In SNA, the symbolic identifier by which end users refer to a network addressable unit (NAU), a link, or a link station. See also *network address*. (2) In a multiple-domain network, the name of the APPL statement defining a VTAM application program is its network name and it must be unique across domains. Contrast with *ACB name*. See *uninterpreted name*.

network operator. (1) A person or program responsible for controlling the operation of all or part of a network. (2) The person or program that controls all the domains in a multiple-domain network. Contrast with *domain operator*.

network product support (NPS). The function of the NetView program that provides operations control for the IBM 3710 Network Controller, 5860 family of modems, and the NCP; and configuration of 3710s and the 5860 family of modems. NPS provides operator commands to run diagnostics for link problem determination and to change product operating parameters.

network services (NS). In SNA, the services within network addressable units (NAUs) that control network operation through SSCP-SSCP, SSCP-PU, and SSCP-LU sessions. See *configuration services, maintenance services, management services*, and *session services*.

network services (NS) header. In SNA, a 3-byte field in a function management data (FMD) request/response unit (RU) flowing in an SSCP-LU, SSCP-PU, or SSCP-SSCP session. The network services header is used primarily to identify the network services category of the request unit (RU) (for example, configuration services, session services) and the particular request code within a category.

NMVT. Network management vector transport.

NNT. NetView-NetView task.

node. (1) In SNA, an endpoint of a link or junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities. See *boundary node*, *host node*, *peripheral node*, and *subarea node* (including illustration). (2) In VTAM, a point in a network defined by a symbolic name. See *major node* and *minor node*.

node name. In VTAM, the symbolic name assigned to a specific major or minor node during network definition.

node type. In SNA, a designation of a node according to the protocols it supports and the network addressable units (NAUs) that it can contain. Five types are defined: 1, 2.0, 2.1, 4, and 5. Type 1, type 2.0, and type 2.1 nodes are peripheral nodes; type 4 and type 5 nodes are subarea nodes. See also *type 2.1 node*.

no response. In SNA, a value in the form-of-response-requested field of the request header (RH) indicating that no response is to be returned to the request, whether or not the request is received and processed successfully. Contrast with *definite response* and *exception response*.

NPS. Network product support.

NS. Network services.

online. Stored in a computer and accessible from a terminal.

open. (1) In the IBM Token-Ring Network, to make an adapter ready for use. (2) A break in an electrical circuit.

operand. (1) (ISO) An entity on which an operation is performed. (2) * That which is operated upon. An operand is usually identified by an address part of an instruction. (3) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor. (4) An expression to whose value an operator is applied. See also *definition statement*, *keyword*, *keyword parameter*, and *parameter*.

operator. (1) In a language statement, the lexical entity that indicates the action to be performed on operands. (2) A person who operates a machine. See *network operator*. See also *definition statement*.

operator profile. In the NetView program, the resources and activities a network operator has control over. The statements defining these resources and activities are stored in a file that is activated when the operator logs on.

operator station task (OST). The NetView task that establishes and maintains the online session with the network operator. There is one operator station task for each network operator who logs on to the NetView program. See *NetView-NetView task*.

OST. Operator station task.

pacing. In SNA, a technique by which a receiving component controls the rate of transmission of a sending component to prevent overrun or congestion. See *session-level pacing*, *send pacing*, and *virtual route (VR) pacing*. See also *flow control*.

pacing response. In SNA, an indicator that signifies a receiving component's readiness to accept another pacing group; the indicator is carried in a response header (RH) for session-level pacing, and in a transmission header (TH) for virtual route pacing.

page. (1) The portion of a panel that is shown on a display surface at one time. (2) To move back and forth among the pages of a multiple-page panel. See also *scroll*. (3) (ISO) In a virtual storage system, a fixed-length block that has a virtual address and that can be transferred between real storage and auxiliary storage. (4) To transfer instructions, data, or both between real storage and external page or auxiliary storage.

panel. (1) A formatted display of information that appears on a terminal screen. See also *help panel* and *task panel*. Contrast with *screen*. (2) In computer graphics, a display image that defines the locations and characteristics of display fields on a display surface.

parameter. (1) (ISO) A variable that is given a constant value for a specified application and that may denote the application. (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (3) Data passed to a program or procedure by a user or another program, namely as an operand in a language statement, as an item in a menu, or as a shared data structure. See also *keyword*, *keyword parameter*, and *operand*.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

path. (1) In SNA, the series of path control network components (path control and data link control) that are traversed by the information exchanged between two network addressable units (NAUs). See also *explicit route (ER)*, *route extension*, and *virtual route (VR)*. (2) In VTAM when defining a switched major node, a potential dial-out port that can be used to reach that node. (3) In the NetView/PC program, a complete line in a configuration that contains all of the resources in the service point command service (SPCS) query link configuration request list.

path control (PC). The function that routes message units between network addressable units (NAUs) in the network and provides the paths between them. It converts the BIUs from transmission control (possibly segmenting them) into path information units (PIUs) and exchanges basic transmission units (BTUs) and one or more PIUs with data link control. Path control differs for peripheral nodes, which use local addresses for routing, and subarea nodes, which use network addresses for routing. See *peripheral path control* and *subarea path control*. See also *link*, *peripheral node*, and *subarea node*.

path control (PC) layer. In SNA, the layer that manages the sharing of link resources of the SNA network and routes basic information units (BIUs) through it. See also *BIU segment*, *blocking of PIUs*, *data link control layer*, and *transmission control layer*.

path control (PC) network. In SNA, the part of the SNA network that includes the data link control and path control layers. See *SNA network* and *user application network*. See also *boundary function*.

PC. (1) Path control. (2) Personal Computer. Its full name is the IBM Personal Computer.

peripheral host node. A node that provides an application program interface (API) for running application programs but does not provide SSCP functions and is not aware of the network configuration. The peripheral host node does not provide subarea node services. It has boundary function provided by its adjacent subarea. See *boundary node*, *host node*, *node*, *peripheral node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

peripheral LU. In SNA, a logical unit representing a peripheral node.

peripheral node. In SNA, a node that uses local addresses for routing and therefore is not affected by changes in network addresses. A peripheral node requires boundary-function assistance from an adjacent subarea node. A peripheral node is a physical unit (PU) type 1, 2.0, or 2.1 node connected to a subarea node with boundary function within a subarea. See *boundary node*, *host node*, *node*, *peripheral host node*, *subarea host node*, and *subarea node*. See also *boundary function* and *node type*.

peripheral path control. The function in a peripheral node that routes message units between units with local addresses and provides the paths between them. See *path control* and *subarea path control*. See also *boundary function*, *peripheral node*, and *subarea node*.

peripheral PU. In SNA, a physical unit representing a peripheral node.

Personal Computer (PC). The IBM Personal Computer line of products including the 5150 and subsequent models.

physical connection. In VTAM, a point-to-point connection or multipoint connection. Synonymous with *connection*.

physical unit (PU). In SNA, a type of network addressable unit (NAU). A physical unit (PU) manages and monitors the resources (such as attached links) of a node, as requested by a system services control point (SSCP) through an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links. See also *peripheral PU* and *subarea PU*.

physical unit (PU) services. In SNA, the components within a physical unit (PU) that provide configuration services and maintenance services for SSCP-PU sessions. See also *logical unit (LU) services*.

PLU. Primary logical unit.

POI. Programmed operator interface.

positional operand. An operand in a language statement that has a fixed position. See also *definition statement*. Contrast with *keyword operand*.

positive response. A response indicating that a request was received and processed. Contrast with *negative response*.

POST. Power-on self test. A series of diagnostic tests that are run each time the computer's power is turned on.

PPT. Primary POI task.

primary half-session. In SNA, the half-session that sends the session activation request. See also *primary logical unit*. Contrast with *secondary half-session*.

primary logical unit (PLU). In SNA, the logical unit (LU) that contains the primary half-session for a particular LU-LU session. Each session must have a PLU and secondary logical unit (SLU). The PLU is the unit responsible for the bind and is the controlling LU for the session. A particular LU may contain both primary and secondary half-sessions for different active LU-LU sessions. Contrast with *secondary logical unit (SLU)*.

primary POI task (PPT). The NetView subtask that processes all unsolicited messages received from the VTAM program operator interface (POI) and delivers them to the controlling operator or to the command processor. The PPT also processes the initial command specified to execute when the NetView program is initialized and timer request commands scheduled to execute under the PPT.

problem determination. The process of identifying the source of a problem; for example, a program component, a machine failure, telecommunication facilities, user or contractor-installed programs or equipment, an environment failure such as a power loss, or a user error.

product-set identification (PSID). (1) In SNA, a technique for identifying the hardware and software products that implement a network component. (2) A management services common subvector that transports the information described in definition (1).

profile. In the Conversational Monitor System (CMS) or the group control system (GCS), the characteristics defined by a PROFILE EXEC file that executes automatically after the system is loaded into a virtual machine. See also *operator profile*.

programmed operator interface (POI). A VTAM function that allows programs to perform VTAM operator functions.

PSID. Product-set identification.

PU. Physical unit.

public network. A network established and operated by communication common carriers or telecommunication Administrations for the specific purpose of providing circuit-switched, packet switched, and leased-circuit services to the public. Contrast with *user-application network*.

PU-PU flow. In SNA, the exchange between physical units (PUs) of network control requests and responses.

receive pacing. In SNA, the pacing of message units that the component is receiving. See also *send pacing*.

RECFMS. Record formatted maintenance statistics.

RECMS. Record maintenance statistics.

record. (1) (ISO) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (2) (TC97) A set of data treated as a unit. (3) A set of one or more related data items grouped for processing. (4) In VTAM, the unit of data transmission for record mode. A record represents whatever amount of data the transmitting node chooses to send.

record formatted maintenance statistics (RECFMS). A statistical record built by an SNA controller and usually solicited by the host.

record maintenance statistics (RECMS). An SNA error event record built from an NCP or line error and sent unsolicited to the host.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks. For example, the 3710 Network Controller routines may be reentrant.

regular command. In the NetView program, any VTAM or NetView command that is not an immediate command and is processed by a regular command processor. Contrast with *immediate command*.

release. For VTAM, to relinquish control of resources (communication controllers or physical units). See also *resource takeover*. Contrast with *acquire (2)*.

remote. Concerning the peripheral parts of a network not centrally linked to the host processor and generally using telecommunication lines with public right-of-way.

remove. In the IBM Token-Ring Network, to take an attaching device off the ring.

request header (RH). In SNA, control information preceding a request unit (RU). See also *request/response header (RH)*.

request unit (RU). In SNA, a message unit that contains control information, end-user data, or both.

request/response header (RH). In SNA, control information, preceding a request/response unit (RU), that specifies the type of RU (request unit or response unit) and contains control information associated with that RU.

request/response unit (RU). In SNA, a generic term for a request unit or a response unit. See also *request unit (RU)* and *response unit*.

reset. On a virtual circuit, reinitialization of data flow control. At reset, all data in transit are eliminated.

resource. (1) Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs. (2) In the NetView program, any hardware or software that provides function to the network.

resource takeover. In VTAM, action initiated by a network operator to transfer control of resources from one domain to another. See also *acquire (2)* and *release*. See *takeover*.

response. A reply represented in the control field of a response frame. It advises the primary or combined station of the action taken by the secondary or other combined station to one or more commands. See also *command*.

response header (RH). In SNA, a header, optionally followed by a response unit (RU), that indicates whether the response is positive or negative and that may contain a pacing response. See also *negative response, pacing response*, and *positive response*.

response time. (1) The amount of time it takes after a user presses the enter key at the terminal until the reply appears at the terminal. (2) For response time monitoring, the time from the activation of a transaction until a response is received, according to the response time definition coded in the performance class.

response unit (RU). In SNA, a message unit that acknowledges a request unit; it may contain prefix information received in a request unit. If positive, the response unit may contain additional information (such as session parameters in response to Bind Session), or if negative, contains sense data defining the exception condition.

Restructured Extended Executor (REXX). An interpretive language used to write command lists.

return code. * A code [returned from a program] used to influence the execution of succeeding instructions.

REXX. Restructured Extended Executor.

RH. Request/response header.

route. See *explicit route* and *virtual route*.

route extension (REX). In SNA, the path control network components, including a peripheral link, that make up the portion of a path between a subarea node and a network addressable unit (NAU) in an adjacent peripheral node. See also *path, explicit route (ER)* and *virtual route (VR)*.

routing. The assignment of the path by which a message will reach its destination.

RPL exit routine. In VTAM, an application program exit routine whose address has been placed in the EXIT field of a request parameter list (RPL). VTAM invokes the routine to indicate that an asynchronous request has been completed. See *EXLST exit routine*.

RU. Request/response unit.

RU chain. In SNA, a set of related request/response units (RUs) that are consecutively transmitted on a particular normal or expedited data flow. The request RU chain is the unit of recovery: if one of the RUs in the chain cannot be processed, the entire chain is discarded. Each RU belongs to only one chain, which has a beginning and an end indicated by means of control bits in request/response headers within the RU chain. Each RU can be designated as first-in-chain (FIC), last-in-chain (LIC), middle-in-chain (MIC), or only-in-chain (OIC). Response units and expedited-flow request units are always sent as only-in-chain.

scope of commands. In the NetView program, the facility that provides the ability to assign different responsibilities to various operators.

screen. An illuminated display surface; for example, the display surface of a CRT or plasma panel. Contrast with *panel*.

scroll. To move all or part of the display image vertically to display data that cannot be observed within a single display image. See also *page (2)*.

secondary half-session. In SNA, the half-session that receives the session-activation request. See also *secondary logical unit (SLU)*. Contrast with *primary half-session*.

secondary logical unit (SLU). In SNA, the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. An LU may contain secondary and primary half-sessions for different active LU-LU sessions. Contrast with *primary logical unit (PLU)*.

secondary logical unit (SLU) key. A key-encrypting key used to protect a session cryptography key during its transmission to the secondary half-session.

send pacing. In SNA, pacing of message units that a component is sending. See also *receive pacing*.

sequence number. A number assigned to a particular frame or packet to control the transmission flow and receipt of data.

service point (SP). An entry point that supports applications that provide network management for resources not under the direct control of itself as an entry point. Each resource is either under the direct control of another entry point or not under the direct control of any entry point. A service point accessing these resources is not required to use SNA sessions (unlike a focal point). A service point is needed when entry point support is not yet available for some network management function.

service point command service (SPCS). An extension of the command facility in the NetView program that allows the host processor to communicate with a service point by using the communication network management (CNM) interface.

session. In SNA, a logical connection between two network addressable units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. Each session is uniquely identified in a transmission header (TH) by a pair of network addresses, identifying the origin and destination NAUs of any transmissions exchanged during the session. See *half-session, LU-LU session, SSCP-LU* *session*, *SSCP-PU session*, and *SSCP-SSCP session*. See also *LU-LU session type* and *PU-PU flow*.

session control (SC). In SNA, (1) One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification. (2) A request unit (RU) category used for requests and responses exchanged between the session control components of a session and for session activation and deactivation requests and responses.

session-initiation request. In SNA, an Initiate or logon request from a logical unit (LU) to a control point (CP) that an LU-LU session be activated.

session-level pacing. In SNA, a flow control technique that permits a receiver to control the data transfer rate (the rate at which it receives request units) on the normal flow. It is used to prevent overloading a receiver with unprocessed requests when the sender can generate requests faster than the receiver can process them. See also *pacing* and *virtual route pacing*.

session monitor. The component of the NetView program that collects and correlates session-related data and provides online access to this information.

session services. In SNA, one of the types of network services in the control point (CP) and in the logical unit (LU). These services provide facilities for an LU or a network operator to request that the SSCP initiate or terminate sessions between logical units. See *configuration services*, *maintenance services*, and *management services*.

shared. Pertaining to the availability of a resource to more than one use at the same time.

simulated logon. A session-initiation request generated when a VTAM application program issues a SIMLOGON macroinstruction. The request specifies a logical unit (LU) with which the application program wants a session in which the requesting application program will act as the primary logical unit (PLU).

single-domain network. In SNA, a network with one system services control point (SSCP). Contrast with *multiple-domain network*.

SLU. Secondary logical unit.

SMF. System management facility.

SNA. Systems Network Architecture.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network addressable units (NAUs), boundary function components, and the path control network.

SP. Service point.

SPCS. Service point command service.

span. In the NetView program, a user-defined group of network resources within a single domain. Each major or minor node is defined as belonging to one or more spans. See also *span of control*.

span of control. The total network resources over which a particular network operator has control. All the network resources listed in spans associated through profile definition with a particular network operator are within that operator's span of control.

SSCP. System services control point.

SSCP-LU session. In SNA, a session between a system services control point (SSCP) and a logical unit (LU); the session enables the LU to request the SSCP to help initiate LU-LU sessions.

SSCP-PU session. In SNA, a session between a system services control point (SSCP) and a physical unit (PU); SSCP-PU sessions allow SSCPs to send requests to and receive status information from individual nodes in order to control the network configuration.

SSCP-SSCP session. In SNA, a session between the system services control point (SSCP) in one domain and the SSCP in another domain. An SSCP-SSCP session is used to initiate and terminate cross-domain LU-LU sessions.

SSP. System Support Programs (IBM licensed program). Its full name is Advanced Communications Function for System Support Programs. Synonymous with *ACF/SSP*.

start option. In VTAM, a user-specified or IBM-supplied option that determines certain conditions that are to exist during the time a VTAM system is operating. Start options can be predefined or specified when VTAM is started.

statement. A language syntactic unit consisting of an operator, or other statement identifier, followed by one or more operands. See *definition statement*.

station. (1) One of the input or output points of a network that uses communication facilities; for example, the telephone set in the telephone system or the point where the business machine interfaces with the channel on a leased private line. (2) One or more computers, terminals, or devices at a particular location.

subarea. A portion of the SNA network consisting of a subarea node, any attached peripheral nodes, and their associated resources. Within a subarea node, all network addressable units, links, and adjacent link stations (in attached peripheral or subarea nodes) that are addressable within the subarea share a common subarea address and have distinct element addresses.

subarea host node. A host node that provides both subarea function and an application program interface (API) for running application programs. It provides system services control point (SSCP) functions, subarea node services, and is aware of the network configuration. See *boundary node*, *communication management configuration host node*, *data host node*, *host node*, *node*, *peripheral node*, and *subarea node*. See also *boundary function* and *node type*.

subarea node. In SNA, a node that uses network addresses for routing and whose routing tables are therefore affected by changes in the configuration of the network. Subarea nodes can provide gateway function, and boundary function support for peripheral nodes. Type 4 and type 5 nodes are subarea nodes. See *boundary node*, *host node*, *node*, *peripheral node*, and *subarea host node*. See also *boundary function* and *node type*.

subarea path control. The function in a subarea node that routes message units between network addressable units (NAUs) and provides the paths between them. See *path control* and *peripheral path control*. See also *boundary function*, *peripheral node*, and *subarea node*.

subarea PU. In SNA, a physical unit (PU) in a subarea node.

subsystem. A secondary or subordinate system, usually capable of operating independent of, or asynchronously with, a controlling system.

subvector. A subcomponent of the MAC major vector.

supervisor. The part of a control program that coordinates the use of resources and maintains the flow of processing unit operations.

system management facility (SMF). A standard feature of MVS that collects and records a variety of system and job-related information.

system services control point (SSCP). In SNA, a central location point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network. Multiple SSCPs, cooperating as peers, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its domain.

system services control point (SSCP) domain. The system services control point and the physical units (PUs), logical units (LUs), links, link stations and all the resources that the SSCP has the ability to control by means of activation requests and deactivation requests.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

System Support Programs (SSP). An IBM licensed program, made up of a collection of utilities and small programs, that supports the operation of the NCP.

TAF. Terminal access facility.

takeover. The process by which the failing active subsystem is released from its extended recovery facility (XRF) sessions with terminal users and replaced by an alternate subsystem. See *resource takeover*.

task. A basic unit of work to be accomplished by a computer. The task is usually specified to a control program in a multiprogramming or multiprocessing environment.

task panel. Online display from which you communicate with the program in order to accomplish the program's function, either by selecting an option provided on the panel or by entering an explicit command. See *help panel*.

telecommunication line. Any physical medium such as a wire or microwave beam, that is used to transmit data. Synonymous with *transmission line*.

terminal. A device that is capable of sending and receiving information over a link; it is usually equipped with a keyboard and some kind of display, such as a screen or a printer.

terminal access facility (TAF). In the NetView program, a facility that allows a network operator to control a number of subsystems. In a full-screen or operator control session, operators can control any combination of such subsystems simultaneously.

TERMINATE. In SNA, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure to end one or more designated LU-LU sessions.

time-out. (1) (ISO) An event that occurs at the end of a predetermined period of time that began at the occurrence of another specified event. (2) A time interval allotted for certain operations to occur; for example, response to polling or addressing before system operation is interrupted and must be restarted.

time sharing option (TSO). An optional configuration of the operating system that provides conversational time sharing from remote stations.

token. A sequence of bits passed from one device to another along the token ring. When the token has data appended to it, it becomes a frame.

transmission control (TC) layer. In SNA, the layer within a half-session that synchronizes and paces session-level data traffic, checks session sequence numbers of requests, and enciphers and deciphers end-user data. Transmission control has two components: the connection point manager and session control. See also *half-session*.

transmission line. Synonym for *telecommunication line*.

TSO. Time sharing option.

type 2.1 node (T2.1 node). A node that can attach to an SNA network as a peripheral node using the same protocols as type 2.0 nodes. Type 2.1 nodes can be directly attached to one another using peer-to-peer protocols. See *end node*, *node*, and *subarea node*. See also *node type*.

type 2.1 node (T2.1 node) control point domain. The CP, its logical units (LUs), links, link stations, and all resources that it activates and deactivates.

unformatted. In VTAM, pertaining to commands (such as LOGON or LOGOFF) entered by an end user and sent by a logical unit in character form. The character-coded command must be in the syntax defined in the user's unformatted system services definition table. Synonymous with *character-coded*. Contrast with *field-formatted*.

unformatted system services (USS). In SNA products, a system services control point (SSCP) facility that translates a character-coded request, such as a logon or logoff request into a field-formatted request for processing by formatted system services and translates field-formatted replies and responses into character-coded requests for processing by a logical unit. Contrast with *formatted system services*. See also *converted command*.

uninterpreted name. In SNA, a character string that a system services control point (SSCP) is able to convert into the network name of a logical unit (LU). Typically, an uninterpreted name is used in a logon or Initiate request from a secondary logical unit (SLU) to identify the primary logical unit (PLU) with which the session is requested.

upstream. In the direction of data flow from the end user to the host. Contrast with *downstream*.

user. Anyone who requires the services of a computing system.

user-application network. A configuration of data processing products, such as processors, controllers, and terminals, established and operated by users for the purpose of data processing or information exchange, which may use services offered by communication common carriers or telecommunication Administrations. Contrast with *public network*.

user exit. A point in an IBM-supplied program at which a user routine may be given control.

user exit routine. A user-written routine that receives control at predefined user exit points. User exit routines can be written in assembler or a high-level language (HLL).

USS. Unformatted system services.

value. (1) (TC97) A specific occurrence of an attribute, for example, "blue" for the attribute "color." (2) A quantity assigned to a constant, a variable, a parameter, or a symbol.

variable. In the NetView program, a character string beginning with & that is coded in a command list and is assigned a value during execution of the command list.

vector. The MAC frame information field.

verb. (1) In SNA, the general name for a transaction program's request for communication services. (2) In VTAM, a programming language element in the logical unit (LU) 6.2 application program interface (API) that causes an LU 6.2 function to be performed.

Virtual Machine (VM). A licensed program whose full name is the Virtual Machine/System Product (VM/SP). It is a software operating system that manages the resources of a real processor to provide virtual machines to end users. As a time-sharing system control program, it consists of the virtual machine control program (CP), the conversational monitor system (CMS), the group control system (GCS), and the interactive problem control system (IPCS).

virtual route (VR). In SNA, a logical connection (1) between two subarea nodes that is physically realized as a particular explicit route, or (2) that is contained wholly within a subarea node for intranode sessions. A virtual route between distinct subarea nodes imposes a transmission priority on the underlying explicit route, provides flow control through virtual-route pacing, and provides data integrity through sequence numbering of path information units (PIUs). See also *explicit route (ER)*, *path*, and *route extension*.

virtual route (VR) pacing. In SNA, a flow control technique used by the virtual route control component of path control at each end of a virtual route to control the rate at which path information units (PIUs) flow over the virtual route. VR pacing can be adjusted according to traffic congestion in any of the nodes along the route. See also *pacing* and *session-level pacing*.

virtual route selection exit routine. In VTAM, an optional installation exit routine that modifies the list of virtual routes associated with a particular class of service before a route is selected for a requested LU-LU session.

virtual storage. (ISO) The notion of storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

Virtual Storage Extended (VSE). An IBM licensed program whose full name is the Virtual Storage Extended/Advanced Function. It is a software operating system controlling the execution of programs.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

vital product data. Product identification information such as machine type, model number, and serial number for hardware products. For software products, vital product data can be version and release level.

VM. Virtual Machine operating system. Its full name is Virtual Machine/System Product. Synonymous with *VM/SP*.

VM SNA console support (VSCS). A VTAM component for the VM environment that provides Systems Network Architecture (SNA) support. It allows SNA terminals to be virtual machine consoles.

VM/SP. Virtual Machine/System Product operating system. Synonym for *VM*.

VR. Virtual route.

VSAM. Virtual Storage Access Method.

VSCS. VM SNA console support.

VSE. Virtual Storage Extended operating system. Synonymous with *VSE/AF*.

VSE/AF. Virtual Storage Extended/Advanced Function operating system. Synonym for *VSE*.

VTAM. Virtual Telecommunications Access Method (IBM licensed program). Its full name is Advanced Communications Function for the Virtual Telecommunications Access Method. Synonymous with *ACF/VTAM*.

VTAM operator command. A command used to monitor or control a VTAM domain. See also *definition statement*.

wrap. In general, to go from the maximum to the minimum in computer storage. For example, the continuation of an operation from the maximum value in storage to the first minimal value.

Bibliography

NetView Publications

Learning About NetView: Operator Training (SK2T-0292) is an interactive PC-based operator training package that teaches SNA and basic network management concepts to new and inexperienced NetView operators. This training package uses graphics, animation and interactive NetView product simulations in a series of lessons to teach the basics of NetView operation.

NetView Installation and Administration Guide (SC31-6018) helps system programmers install and prepare the NetView program for operation. It is arranged in a simplified, step-by-step style and is meant to be used in conjunction with the sample network documented in *Network Program Products Samples*.

NetView Administration Reference (SC31-6014) is for system programmers and network operators who need a complete understanding of the NetView resource definition statements. This book lists each statement in alphabetical order giving its purpose and location.

NetView Tuning Guide (SC31-6079)[1] describes methods for controlling and improving the performance of the NetView Release 3 program. It is designed for system programmers who need to understand how NetView tuning values are determined and optimized.

NetView Customization Guide (SC31-6016) is designed for system programmers and others who want to customize the NetView program to reflect their network's needs or operating procedures. This book focuses on the different application programming interfaces that can be customized and explains how to modify NetView help panels and problem determination displays.

NetView Customization: Using PL/I and C (SC31-6037) describes the ways system programmers can tailor the NetView program to satisfy unique requirements or operating procedures. It discusses the uses and advantages of user-written programs (exit routines, command processors, and subtasks). It also provides instructions in designing, writing, and installing user-written programs in PL/I and C.

NetView Customization: Using Assembler (SC31-6078) describes the ways system programmers can tailor the NetView program to satisfy unique requirements or operating procedures. It discusses the uses and advantages of user-written programs (exit routines, command processors, and subtasks). It also provides instructions in designing, writing, and installing user-written programs in Assembler.

NetView Customization: Writing Command Lists (SC31-6015) explains how to simplify network operator tasks by using command lists. It provides step-by-step instructions for writing simple and advanced command lists and for migrating from NCCF message automation to NetView message automation.

NetView Operation Primer (SC31-6020) provides a basic description of the network management task for new network operators. Topics include starting and stopping a network, controlling resources, monitoring a network, and gathering the necessary data to report a problem.

NetView Operation (SC31-6019) provides system programmers and experienced network operators a comprehensive explanation of network management using the NetView program. Topics include detailed command explanation and panel flows, as well as information on how the various components interact with each other.

NetView Command Summary (SX75-0026) is a reference card that provides network operators with the format of all the commands and the commonly used NetView command lists. The commands are listed in alphabetical order by component.

NetView Problem Determination and Diagnosis (LY43-0001) aids system programmers in identifying a NetView problem, classifying it, and describing it to an IBM Support Center.

NetView Problem Determination Supplement for Management Services Major Vectors 0001 and 0025 (LD21-0023) describes major vectors 0001 and 0025 for system programmers and network operators involved in problem determination or diagnosis. The supplement may be used for the generic alert option and other problem determination tasks.

NetView Resource Alerts Reference (SC31-6024) lists the messages sent by NetView-supported hardware and software resources. It helps system programmers analyze the messages into their component parts: action codes, event types, message text, and qualifiers. The book is a reference for those who need more information than online help provides.

1 When available.

NetView Storage Estimates (SK2T-1988) is an interactive PC-based tool that helps the user estimate storage requirements for NetView. This tool can be used for planning, installation, and tuning purposes. It is intended for network planners, system programmers, and IBM service personnel.

Console Automation Using NetView: Planning (SC31-6058) describes an approach to automate the way a system handles messages and responses to alerts. It includes information you should know before beginning such automation, as well as sample plans and proposals you might find useful in promoting your automation concept. This book includes planning information for MVS, VM, and VSE users of the NetView program.

NetView/PC Publications

NetView/PC Planning, Installation, and Customization (SC31-6002) provides planning, installation, and customization information on NetView/PC and explains the communication requirements upstream to the host and downstream to supported devices. Information relating to the required PC environment and host products that support NetView/PC is also provided. It also discusses topics that are of general interest when you are ordering your equipment.

NetView/PC Application Program Interface/Communications Services Reference (SC31-6004) is a reference for OS/2 programmers who use the API/CS and for system programmers who write command processors to run under NetView. The API/CS provides a means for vendor and other external applications to use the communication services of NetView/PC.

NetView/PC Operation (SC31-6003) describes how to operate the program and diagnose problems in NetView/PC.

NetView/PC Quick Reference (SX75-0016) describes all of the functions of the F-keys throughout the NetView/PC program.

Other Network Program Products Publications

For more information about the books listed in this section, see *Bibliography and Master Index for NetView, NCP, and VTAM*.

Network Program Products General Information (GC30-3350)

Network Program Products Planning (SC30-3351)

Network Program Products Samples (SC30-3352)

Bibliography and Master Index for NetView, NCP, and VTAM (GC31-6081)[2]

VTAM Publications

The following list shows the books for VTAM V3R2. For information about the books for VTAM V3R1, V3R1.1, or V3R1.2, see any VTAM V3R2 book or the *Network Program Products Bibliography and Master Index*.

VTAM Installation and Resource Definition (SC23-0111)

VTAM Customization (LY30-5614)

VTAM Directory of Programming Interfaces for Customers (GC31-6403)

VTAM Operation (SC23-0113)

VTAM Messages and Codes (SC23-0114)

VTAM Programming (SC23-0115)

VTAM Programming for LU 6.2 (SC30-3400)

VTAM Diagnosis Guide (LY30-5601)

VTAM Data Areas for MVS (LY30-5592)

VTAM Data Areas for VM (LY30-5593)

VTAM Data Areas for VSE (LY30-5594)

VTAM Reference Summary (LY30-5600)

NCP, SSP, and EP Publications

The following list shows the related books for NCP V4 and NCP V5.

NCP, SSP, and EP Generation and Loading Guide (SC30-3348)

NCP, SSP, and Related Products Directory of Programming Interfaces for Customers (GC31-6202)

NCP Migration Guide (SC30-3252 for NCP V4 and SC30-3440 for NCP V5)

[2] When available.

NCP, SSP, and EP Resource Definition Guide (SC30-3349 for NCP V4 and SC30-3447 for NCP V5)

NCP, SSP, and EP Resource Definition Reference (SC30-3254 for NCP V4 and SC30-3448 for NCP V5)

NCP and EP Reference Summary and Data Areas (LY30-5570 for NCP V4 and LY30-5603 for NCP V5)

NCP Customization Guide (LY30-5571 for NCP V4 LY30-5606 for NCP V5)

NCP Customization Reference (LY30-5612 for NCP V4 and LY30-5607 for NCP V5)

SSP Customization (LY43-0021)

NCP, SSP, and EP Messages and Codes (SC30-3169)

NCP, SSP, and EP Diagnosis Guide (LY30-5591)

NCP and EP Reference (LY30-5569 for NCP V4 and LY30-5605 for NCP V5)

Related Publications

PL/I Programming Guide (SC26-4307)

PL/I Programming: Language Reference (SC26-4308)

PL/I Programming: Messages and Codes (SC26-4311)

C/370 User's Guide (SC09-1264)

SAA: Common Programming Interface C Reference (SC26-4353)

KnowledgeTool Application Development Guide (SH20-9262)

Index

Reader's Comment Form

NetView™
Customization: PL/I and C
Release 3

Publication No. SC31-6037-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments: ____________________

____

____

____

____

____

____

What is your occupation? ____________________

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC31-6037-0

Reader's Comment Form

Fold and tape   **Please Do Not Staple**   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 40   ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E15
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape   **Please Do Not Staple**   Fold and tape

IBM®

Reader's Comment Form

NetView™
Customization: PL/I and C
Release 3

Publication No. SC31-6037-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments: ____________________

____

____

____

____

____

____

What is your occupation? ____________________

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC31-6037-0

Reader's Comment Form

Fold and tape   **Please Do Not Staple**   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 40   ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E15
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape   **Please Do Not Staple**   Fold and tape

IBM®