Using NS 3000/iX Network Services

HP 3000 MPE/iX Computer Systems

Edition 4

Customer Order Number 36920-61000

36920-90008
E0594

Printed in: U.S.A. May 1994

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Contents

Contents

Contents

Contents

Figures

Tables

Preface

This document is for persons wishing to program or interactively use NS 3000/iX network services. The network services (NS) enable HP 3000 systems and other NS-compatible systems to communicate with each other and share resources.

Audience

This manual is intended for interactive users as well as programmers. In order for you to use NS 3000/iX, you should be familiar with some of the more common MPE/iX commands and intrinsics.

The descriptions of Network File Transfer (NFT) in this manual only cover transfers between HP 3000's using NS 3000/iX. For specifics on using NFT between a PC and an HP 3000, refer to the *User Guide for HP PC Network Services*.

Organization of This Manual

The Introduction in this manual presents an overview of NS 3000/iX, discussing the architecture of the network and introducing the network services. The remaining chapters provide detailed documentation for individual network services. In general, they give an overview of the particular service, explain all relevant commands and intrinsics, and illustrate these commands and intrinsics with examples.

Appendix A, "Migration From NS 3000/V to NS 3000/iX Network Services," describes the differences between NS 3000/iX and NS 3000/V. Appendix B, "Migration From DS/3000 to NS 3000/iX Network Services," describes the differences between NS 3000/iX and DS/3000. The appendixes also describe migration between the services.

Related Publications

For information on related publications, refer to the MPE/iX Documentation Guide.

As mentioned in Edition 3, MPE/iX is a superset of MPE XL. All programs written for MPE XL will run without change under MPE/iX. You can continue to use MPE XL system documentation, although it may not refer to features added to the operating system to support POSIX (for example, hierarchical directories). Finally, you may encounter references to MPE V, which is the operating system for HP 3000s not based on the PA-RISC architecture. MPE V software can be run on the PA-RISC HP 3000s (Series 900) in what is known as compatibility mode.

Note to HP 9000 users: beginning with HP-UX release 10.0, DSCOPY and NetIPC are no longer supported on HP-UX.

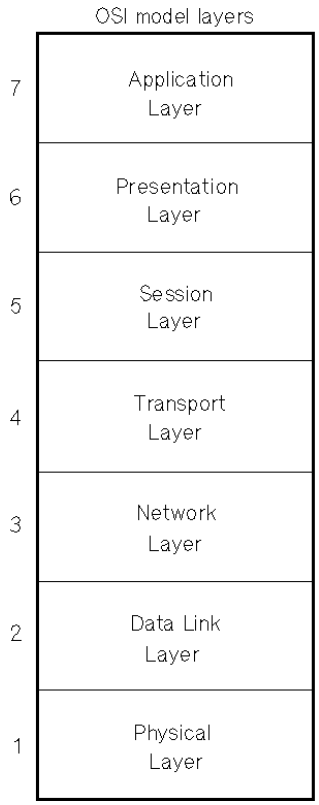
1 Introduction to NS 3000/iX

NS 3000/iX is the name of Hewlett-Packard's software services for linking multi-vendor computer equipment including MPE/iX based HP 3000 processors. The Network Services (NS) run in conjunction with Hewlett-Packard link products that consist of both hardware and software.

Network Architecture

A network architecture specifies the transmission tasks of distinct hardware and software modules or layers. The architecture of NS 3000/iX is based on the seven-layer OSI (Open Systems Interconnection) model (Figure 1-1) developed by the International Standards Organization (ISO). One of the purposes of having a layered architecture is to make the complexities of data communications transparent to the high-level user. Some familiarity with the tasks performed at different levels may be helpful.

Figure 1-1 OSI Model



The highest layer regulates user services, while the lowest layer regulates the actual transmission of bits from one node to another. Each computer system in the network is called a node. At each layer one or more protocols is responsible for carrying out the appropriate tasks. A protocol is a set of rules that specify software message format. From a logical point of view, the protocol entity at each level communicates with the corresponding protocol entity at the same level on another node. In reality, except for the physical transmission of data to another node, each protocol entity communicates with other protocols at the layer immediately above and below its own.

When a message is sent from one node to another in a network, it is first passed down through the architectural levels at the source node. That is, it is transferred from the control of one protocol entity to the control of the next. At one of the middle layers, the message is broken down into packets. At the lowest layer, the packets are sent across the physical communications link. The destination node collects the packets and passes them up to the higher protocol levels where they are reassembled into the original, complete message.

In NS 3000/iX, the Application Layer, at the top of the hierarchy, consists of user-level services such as Virtual Terminal (VT), Network File Transfer (NFT), Remote File Access(RFA), and Remote Database Access (RDBA). The next two layers, Presentation and Session, define functions that contribute to these high-level services, but there is no exact correspondence between NS 3000/iX features and these layers.

The Transport Layer protocols such as TCP, PXP, and UDP handle end-to-end communications between a source and a destination node, ensuring that a message from the source arrives at its destination in the proper form. The fragmentation of messages into packets occurs at this level.

The Network Layer protocols such as IP and X.25 perform an addressing function, making sure that the packets are acquired by the node to which they are addressed.

The Data Link Layer protocols such as IEEE 802.3, Ethernet, LAPB and X.25 govern the actual transmission of the packets over the communications link. At this level the packets are technically known as frames. The lowest layer, the Physical, provides electrical and mechanical specifications for the transmission of bits across the link.

Hewlett-Packard link products such as the ThinLAN 3000/iX Link correspond to the lower four layers of the Open Systems Interconnect (OSI) model (Figure 1-1), and the NS 3000/iX product corresponds to layer seven of the OSI model.

For more information on lower-level network functions, see the and the *NS 3000/iX Error Messages Reference Manual*.

Network Services

NS 3000/iX is the name of Hewlett-Packard's interactive and programmatic user-level, network services. All of these services are listed below and are fully documented in this manual:

- **Virtual Terminal (VT)** creates an interactive session for you on another system in the network, making your terminal appear as though it were directly connected to the other system. This service permits you to issue commands to the remote operating system, use subsystems such as editors and compilers within the remote environment, and run application programs that reside on the remote system. A feature called Reverse Virtual Terminal enables a local application program to communicate efficiently with remote terminals.
- **Remote File Access (RFA)** enables you to perform I/O operations to files and peripheral devices located on other nodes.
- **Remote Data Base Access (RDBA)** allows you to access and update TurboIMAGE data bases located on other nodes. TurboIMAGE is a Hewlett-Packard data base management system.
- **Network File Transfer (NFT)** allows you to transfer or copy files from one node to another, or within a single node, interactively or programmatically. For information on transfers between the PC and the HP 3000, refer to the *User Guide for HP PC Network Services*.
- **Remote Process Management (RPM)** enables a given process to create and terminate processes on other nodes. RPM is commonly used in conjunction with Network Interprocess Communication (NetIPC). NetIPC provides programmatic access to the Transmission Control Protocol (TCP), which is the Transport Layer protocol used by NS 3000/iX link products. For more information on NetIPC, refer to the *NetIPC 3000/iX Programmer's Reference Manual*.

These network services allow you to perform essential functions across a network or across gateways in an internetwork. In addition to a "virtual terminal" you have what amounts to virtual storage and virtual devices; you are not limited to the processing and storage capacities of your own system.

The NS 3000/iX network services including NFT (DSCOPY) and RFA do not support POSIX—the services cannot work with either bytestream files or files residing in HFS (hierarchical file system) directories. To transfer bytestream files across systems, use Hewlett-Packard's FTP/iX product.

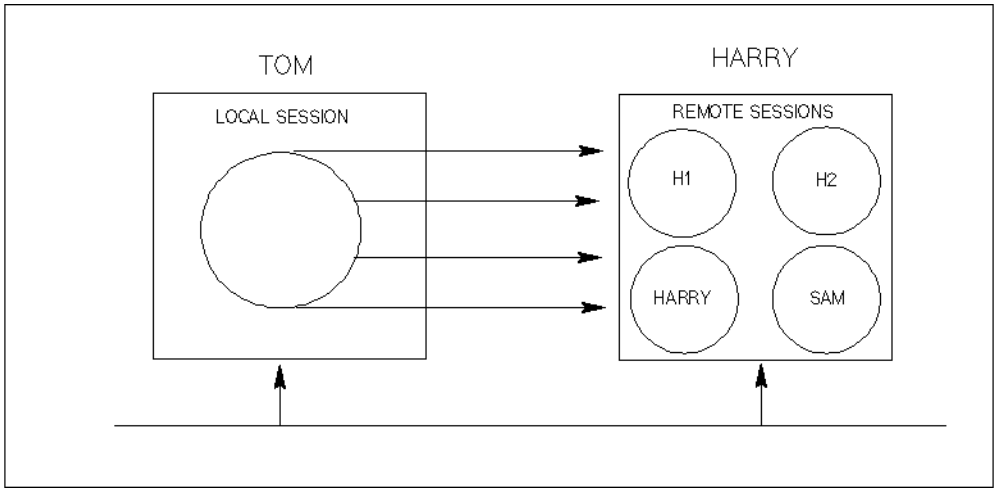
Network Names

When each computer system is configured as part of an NS 3000/iX network, it is assigned a unique *node name*. You use this node name to log on to each computer system. Node names can be in either NS node name syntax or ARPA domain name syntax as explained in the following sections. For people using NS node name syntax, the NS names work the same as in the versions prior to MPE/iX 4.0. Refer to the *HP 3000/iX Network Planning and Configuration Guide* for instructions on configuring node names (both NS and ARPA domain types) and for configuring aliases.

You can log on to a specific session within a node by employing a user-defined name known as an *environment ID*. A default *environment ID* is the node name itself. In order to designate a remote file or device, you must include its remote *environment ID* in an extended file designator, for example, `FILEX:ENV1`. You can maintain multiple remote sessions on a single node by specifying a new *environment ID* for each new session.

Figure 1-2 shows that a user on node TOM has four remote environments on node HARRY, one of which was given the default name HARRY.

Figure 1-2 Opening Several Remote NS 3000/iX Sessions



NS Node Name Syntax

A *node name* or an *environment ID* may optionally be qualified with a *domain* and *organization*. The *domain* and *organization* are arbitrary labels that the network manager specifies when configuring each node into the network. For example, in the name `EMPIRE.SJ.CA`, the *node name* is `EMPIRE`, the *domain* is `SJ`, and the *organization* is `CA`.

You can find full details on *node names*, *environment IDs*, and *remote file designations* in the chapters on "Virtual Terminal" and "Remote File Access" in this manual. For convenient reference, the syntax for *node names* and *environment IDs* is given here.

node[.domain[.organization]]

envname[.domain[.organization]]

If you do not qualify the *node* or *envname* in a user-level command or intrinsic, the configured *domain* and *organization* of the local node are assumed by default.

Each NS *node*, *envname*, *domain*, or *organization* specification can be up to 16 characters long, and can include alphanumeric characters, the underscore (_), and the hyphen (-). The first character of each *node*, *envname*, *domain*, or *organization* name must be alphabetic.

ARPA Domain Name Syntax

Following is the syntax for using ARPA domain names within **DSLINE** and **REMOTE** commands:

```
label[.label][...]
```

The labels must follow the rules for ARPANET host name. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, hyphens (-), or underscores (_). Although underscores are not specified as part of ARPANET syntax, HP allows them for compatibility with the NS-style node names.

The first part of each name (the first label) must not exceed 50 characters, and the total length of a domain name may not exceed 50 characters. If you wish to use an ARPA domain name that is greater than 50 characters, you must use the domain name's alias. Refer to the *HP 3000/iX Network Planning and Configuration Guide* for instructions on configuring aliases. HP recommends using fully qualified names when using ARPA domain name syntax.

2 Virtual Terminal

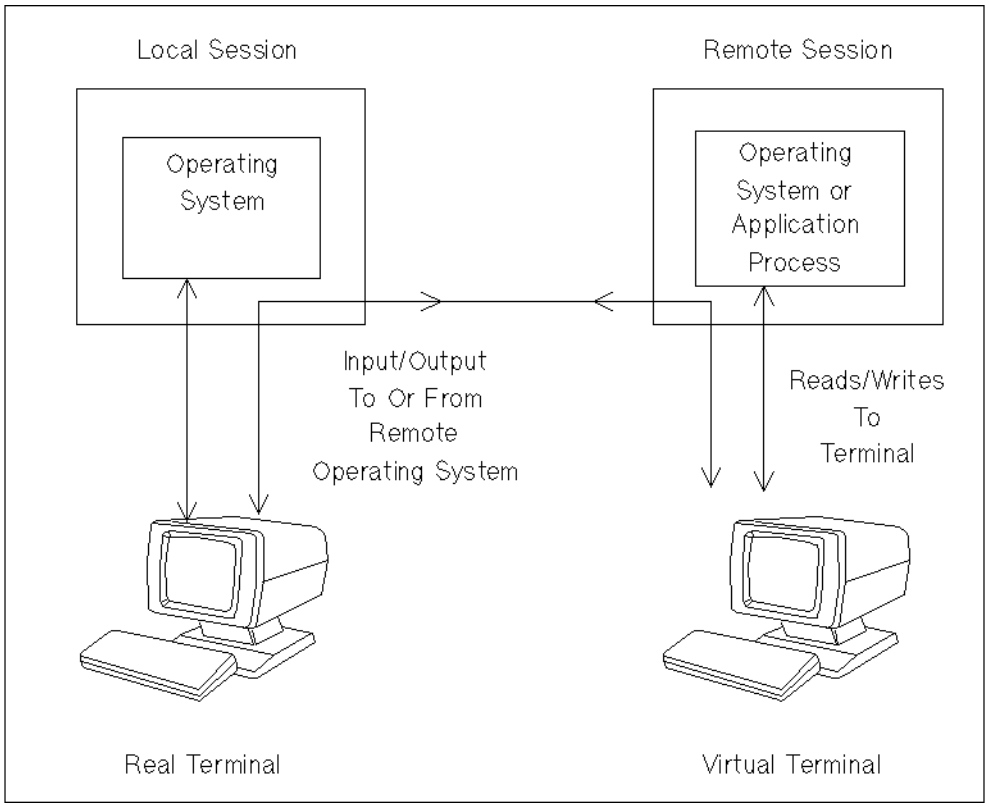
In order to issue interactive commands to a remote operating system or to a subsystem available on a remote computer, you must establish a session on the remote node. The Virtual Terminal service (VT) makes the fact that the session is remote almost entirely transparent. You enter commands and receive system/subsystem responses at your local terminal just as if your session were local. When you edit text in a remote editor subsystem, the text appears in the proper format on your local terminal screen. In reality, input and output to your local terminal pass through a “virtual” (as opposed to actual, physical) terminal configured on the remote system. Your remote commands are transmitted over network connections, sent to the virtual terminal, and subsequently executed on the remote system.

Using the Virtual Terminal service, you can take advantage of a remote system's processing capabilities. For example, if a program needs to be run on a remote node, you can use VT to access it, edit the program, and then compile, load, and run it directly on the remote node.

The Reverse Virtual Terminal service enables an application process within a node to communicate with a real terminal that is on its network or internetwork. The application’s home node sets up a virtual terminal for each real terminal that the application needs access to. Information sent from a terminal to the application process (or vice versa) passes through the appropriate virtual terminal. With Reverse VT, the application process can accept input from all nodes, though individual sessions are not established on each node.

Figure 2-1 is a schematic illustration of the Virtual Terminal (and Reverse Virtual Terminal) service. By using the NS 3000/iX Virtual Terminal service on a network, you can log on to any session-accepting node in the network from your own local node. All systems are transparently accessible to each other.

Figure 2-1 Virtual Terminal Service



In addition to your local session, you can also create multiple remote sessions on your own local node or on remote nodes. Optionally, you can configure your own remote prompts, so that you can identify each remote environment by its prompt.

In order to create a remote session, you can use either `REMOTE` *`:nodename`* followed by a logon, or, you can use `DSLINE` *nodename* followed by a `REMOTE HELLO` *user.acct*. After using either the **`DSLINE`** command or the **`REMOTE HELLO`** command, you then use the **`REMOTE`** command in order to be able to use commands in the remote environment. The following pages will explain how to use these commands.

For recommended programming practices for VT-connected devices, refer to the *Asynchronous Serial Communications Programmer's Reference Manual*.

DSLINE Command

Defines the attributes of a remote environment.

Syntax

```
                [envID              ]
DSLINE          [[envID=]nodename][;SERVICES][;DSLINE option]...
                [#Lenvnum           ]
```

Use

Available	In Session?	Yes
	In Job?	Yes
	In Break?	Yes
	Programmatically?	Yes
Breakable?		No
Capabilities?		None

Parameters

envID An environment ID—that is, a character string representing a specific session on a remote node. For NS names, the environment ID itself may optionally be qualified as follows: *envname*[.*domain*[.*organization*]]. Each portion of the string may have a maximum of 16 alphanumeric characters (including underscores and hyphens), of which the first must be alphabetic. The default *domain* and *organization* names are those specified for your local node when it is configured. For ARPA domain names, the environment ID has the syntax `label[.label[...]`. The labels must follow the syntax for ARPANET host names. The *envID* for ARPA domain names will not be fully qualified. Refer to ARPA Domain Name Syntax in Chapter 1, "Introduction to NS 3000/iX," for more details.

For some *dslineoptions*, an *envID* can be a generic environment ID representing a set of environments. A generic environment ID can include the MPE wild card characters @, #, and ?. @ stands for zero or more alphanumeric characters, # for one numeric character, and ? for one alphanumeric character. The attributes specified for a generic environment ID will be used as defaults for all matching environments, including environments defined later from the same local session,

unless overridden in a later **DSLINE** command. These defaults can be reset by the RESET option, which is described below.

Default: the specified *nodename* (which then becomes an environment ID) or, if *nodename* is omitted as well, the environment specified by the last **DSLINE** or **REMOTE** command.

nodename When you are using NS names, the *nodename* is the name assigned to the remote node when it is configured into the NS 3000/iX network. This name may optionally be qualified in the format *node*[.*domain*[.*organization*]]. The default domain and organization are those of the local node. Each portion of this string may have a maximum of 16 alphanumeric characters (including underscores and hyphens), of which the first must be alphabetic. When you are using ARPA domain names, the nodename has the syntax `label[.label[...]`. The labels must follow the syntax for ARPANET host names. The *envID* for ARPA domain names will not be fully qualified. Refer to ARPA Domain Name Syntax in Chapter 1, "Introduction to NS 3000/iX," for more details.

An environment ID may be equated with this node name, or the node name (if used alone) may become its own environment ID. In either case, the environment ID then represents a specific remote session on this node. Default: the environment specified by the last **DSLINE** or **REMOTE** command.

envnum An environment number representing a specific session on a remote node. This is the number of the environment in the message printed out following a prior **DSLINE** command.

SERVICES Lists the status of the services on the node (local or remote) indicated by the environment ID.

DSLINEoption One of the options described in the following paragraphs.

QUIET Specifies that no logon message be displayed when you log on to the remote environment, no logoff message be displayed when you log off, and no environment messages be displayed when a **DSLINE** command is executed.

COMP or NOCOMP (default is NOCOMP)
: Enables or disables data compression to the remote environment. The compression only affects NFT and RFA transmissions. If data compression is in effect, sequences of repeated characters (such as blanks) are translated into more compact form before transfer. They are decompressed after arrival at their destination. For a discussion of RFA compression, refer to "RFA Compression" in Chapter 3, "Remote File Access." This option may also be used with the **DSCOPY** command as explained in

CLOSE
: Deletes the environment ID(s) associated with the remote environment(s). *This option must be used without any other option*.

RESET
: Clears all information associated with a generic environment ID. *This option must be used without any other option*.

SHOW
: Requests that the attributes of a remote environment (individual or generic) be displayed.

PROMPT= *promptstring*
: Specifies a prompt for the remote environment. This can be used to distinguish one remote environment from another. The prompt string can be 1 to 8 characters long, optionally surrounded by quotation marks. All characters are allowed, but if the string contains a semicolon the string must be in quotes: for example, "MY;NODE". You can also use quotation marks to include a blank at the end of a prompt string. Default: the first seven letters of the (unqualified) environment name (or the whole environment name if shorter) terminated by #. If PROMPT= is specified without a prompt string, the prompt becomes the normal local prompt from the remote operating system for instance, a colon (:).

  If the remote system is an MPE/iX based system, then you can also specify its prompt by using the **SETVAR HPPROMPT** command as explained in the *MPE/iX Commands Reference Manual*. If you specify that the prompt be anything other than a colon, then that specification will override any prompt created by the DSLINE option described here. The SETVAR HPPROMPT will be temporarily overridden if you leave the remote session and then return to it—after a **REMOTE** command, the prompt will remind you which system you are on—after you type a carriage return, the

prompt will return to the one you set with SETVAR HPPROMPT. For an example, refer to the discussion under the **REMOTE** command.

LOGON=
logonsequence

Specifies a logon sequence for the remote system, which can be used by NFT and RPM in order to create a temporary session on the remote node. (See the NFT and RPM chapters of this manual for a discussion of when this logon will take effect.) The logon sequence must include all necessary passwords. It must be delimited by quotation marks if it contains characters which might cause it to be parsed incorrectly by the remote system.

TRACE=
traceoptions

Enables or disables tracing to the remote environment. You can trace the messages sent by any network service (VT, NFT, etc.) between your local session and the remote environment.

The trace records the actual message traffic for each intrinsic call or interactive request, including both network service headers and user-supplied data. You can also trace Transport Layer protocol activity supporting this Network Service traffic.

Other levels of tracing are available through Network InterProcess Communication and Node Management Services. The specific *traceoptions* parameters are:

{[[ON][,*service*][,*file*][,*recs*][,*maxdata*],TRANS]}

{[[OFF][,*service*]]}

The *service* parameter can be: VT, RFA, NFT, RPM, or ALL. Through these choices, you can activate or deactivate tracing for one or all Network Services. The other parameters have the following meanings:

- *file*: The name of a new or existing MPE/iX file in which the trace is to be stored. If this parameter is omitted, the trace information is sent to a default file named TRxxxxxx, where TR is followed by the six leading characters of the remote environment ID.
- *recs*: The number of records allotted to a new trace file. Default: 1024.
- *maxdata*: The maximum amount of data to be traced on an individual send or receive request, a value from 0 to 8000 bytes. Default: 2000 bytes.
- TRANS: Requests tracing of Transport Layer protocol activity, specifically headers and port messages.

For further information on tracing, see the *NS 3000/iX Operations and Maintenance Reference Manual*.

NOTE The following *dslineoptions* are obsolete and are ignored in all cases (if used, a warning message is printed): EXCLUSIVE, FROMADDR=, FROMADR=, LINEBUF=, LOCID=, NOQUEUE, OPEN, PHNUM=, QUEUE, REMID, SELECT=, TOADDR=, and TOADR=.

Description

The **DSLINE** command defines the attributes of a remote environment. These attributes are used when you log on to the remote node via a **REMOTE HELLO** command or when NFT or RPM creates a temporary remote session with the logon sequence specified in the DSLINE LOGON option. In order to establish a remote environment, you must either equate an environment ID with the actual node name, or else use the node name by itself, in which case the node name becomes the environment ID. The environment ID then represents a specific session on the remote node. You can use different environment IDs to represent different sessions on the same node.

Subsequent **DSLINE** commands can use an individual or generic environment ID or an environment number to identify the remote environment(s). If you omit the *nodename*, *envID*, and *envnum*, the default is the last environment referenced by a **DSLINE** or **REMOTE** command. (If this command uses a generic environment ID, the new default environment becomes the last individual environment listed in the environment message then displayed. See the examples that follow.)

After a **DSLINE** command has been executed, a message is printed that identifies all the affected environments. This message includes, for each environment, the environment number assigned (in order of environments defined), the fully qualified environment ID, and the fully qualified node name (if different from the environment ID). If the command specifies attributes for a generic environment ID, the generic environment ID is listed separately in the returned message, identified by the words GENERIC ENVIRONMENT. (Generic environments are given a separate number in the sequence of environments, but this number is not listed.) If the command uses a generic environment ID but does not specify attributes, a separate generic environment is not listed. The reason for this is that no new environment (with new default attributes) is being defined.

Examples

Starting with MPE/iX release 4.0, NS services support ARPA domain node names as well as NS node names. There are some changes in selecting environments using generic EnvIDs. These changes will not affect people who use only NS node names.

Pattern	Before Rel 4.0	After Rel 4.0
@	Selects all EnvIDs with default domain & organization.	Select all NS EnvIDs with default domain and organization and all ARPA domain EnvIDs.
@.@	Selects all EnvIDs with default organization.	Select all NS EnvIDs with default organization and all the ARPA domain EnvIDs with at least one ".".
@.@.@	Select all EnvIDs	Select all NS EnvIDs and all the ARPA domain EnvIDs with at least two ".".
@.@.@.@	None	All the ARPA domain EnvIDs with at least three ".".

Examples

```
:DSLINE SYS4                                          **NS node name with default
ENVIRONMENT 2: SYS4.DETROIT.MYCO=SYS4.DETROIT.MYCO          EnvID, domain & Org**

:DSLINE X1=SYS4                                       **2nd environment referencing
ENVIRONMENT 5: X1.DETROIT.MYCO=SYS4.DETROIT.MYCO            same remote node **

:DSLINE ROBERT                                        **DSLINE using ARPA domain
ENVIRONMENT 7: ROBERT=ROBERT.CUP.MYCO.COM               node name. Note: EnvID
                                                          is not fully qualified **

:DSLINE X2=ROBERT                                     **2nd environment referencing
ENVIRONMENT 9: X2=ROBERT.CUP.MYCO.COM                       same remote ARPA name**

:DSLINE BOB                                           **using alias to ARPA name
ENVIRONMENT 11: BOB=BOB(ROBERT.CUP.MYCO.COM)            "ROBERT". First 50 chars of
                                                         ARPA domain name
                                                        will be displayed in "()" **

:DSLINE X3=BOB                                        **2nd environment referencing
ENVIRONMENT 13: X3=BOB(ROBERT.CUP.MYCO.COM)             same alias name **

:DSLINE SYS3.CHICAGO                                  **using NS node with different
ENVIRONMENT 15: SYS3.CHICAGO.MYCO=SYS3.CHICAGO.MYCO     domain (CHICAGO)**

:DSLINE X@;PROMPT="TESTENV"                           **change the prompt for all
ENVIRONMENT 5: X1.DETROIT.MYCO=SYS4.DETROIT.MYCO         EnvIDs starts with X
ENVIRONMENT 9: X2=ROBERT.CUP.MYCO.COM                    using genericID X@ **
ENVIRONMENT 13: X3=BOB(ROBERT.CUP.MYCO.COM)
GENERIC ENVIRONMENT X@

:DSLINE                                              **lists default environment **
ENVIRONMENT 15: SYS3.CHICAGO.MYCO=SYS3.CHICAGO.MYCO

:DSLINE @          **lists all the ARPA domain environments (strict pattern match
                     for ARPA domain EnvIDs) and all the NS environments with
                      default domain (DETROIT) and organization (MYCO) **
```

```
ENVIRONMENT 2: SYS4.DETROIT.MYCO=SYS4.DETROIT.MYCO
ENVIRONMENT 5: X1.DETROIT.MYCO=SYS4.DETROIT.MYCO
ENVIRONMENT 7: ROBERT=ROBERT.CUP.MYCO.COM
ENVIRONMENT 9: X2=ROBERT.CUP.MYCO.COM
ENVIRONMENT 11: BOB=BOB(ROBERT.CUP.MYCO.COM)
ENVIRONMENT 13: X3=BOB(ROBERT.CUP.MYCO.COM
GENERIC ENVIRONMENT X@

:DSLINE @.@.@      **lists all NS environments and ARPA domain environments with
                      at least two “.” **

ENVIRONMENT 2: SYS4.DETROIT.MYCO=SYS4.DETROIT.MYCO
ENVIRONMENT 5: X1.DETROIT.MYCO=SYS4.DETROIT.MYCO
ENVIRONMENT 15: SYS3.CHICAGO.MYCO=SYS3.CHICAGO.MYCO
GENERIC ENVIRONMENT X@

:DSLINE X@;SHOW     **displays all the EnvIDs that start with X **

ENVIRONMENT #  : 5
ENVIRONMENT ID : X1.DETROIT.MYCO
NODE NAME      : SYS4.DETROIT.MYCO
LOGON          :
LOGGED ON      : NO
PROMPT         : TESTENV
ESTABLISHED BY : DSLINE
SERVICES       :
OPTIONS        :

ENVIRONMENT #  : 9
ENVIRONMENT ID : X2
NODE NAME      : ROBERT.CUP.MYCO.COM
LOGON          :
LOGGED ON      : NO
PROMPT         : TESTENV
ESTABLISHED BY : DSLINE
SERVICES       :
OPTIONS        :

ENVIRONMENT #  : 13
ENVIRONMENT ID : X3
NODE NAME      : BOB
LOGON          :
LOGGED ON      : NO
PROMPT         : TESTENV
ESTABLISHED BY : DSLINE
SERVICES       :
OPTIONS        :

ENVIRONMENT ID : X@
NODE NAME      :
LOGON          :
LOGGED ON      : NO
PROMPT         : TESTENV
ESTABLISHED BY : DSLINE
SERVICES       :
OPTIONS        :

:DSLINE @;CLOSE    **closes all ARPA domain environments and all NS environments
                     with default domain and organization (DETROIT & MYCO)**

ENVIRONMENT 2: SYS4.DETROIT.MYCO=SYS4.DETROIT.MYCO
ENVIRONMENT 5: X1.DETROIT.MYCO=SYS4.DETROIT.MYCO
ENVIRONMENT 7: ROBERT=ROBERT.CUP.MYCO.COM
ENVIRONMENT 9: X2=ROBERT.CUP.MYCO.COM
ENVIRONMENT 11: BOB=BOB(ROBERT.CUP.MYCO.COM)
ENVIRONMENT 13: X3=BOB(ROBERT.CUP.MYCO.COM)

:DSLINE BOB.TEST.TEST=BOB   **using user defined EnvID with alias ARPA domain **

ENVIRONMENT 2: BOB.TEST.TEST=BOB(ROBERT.CUP.MYCO.COM)
```

```
:DSLINE @.@.@                      **lists all the NS environments and
                                      ARPA domain EnvIDs with at least two “.”s **

ENVIRONMENT 2: BOB.TEST.TEST=BOB(ROBERT.CUP.MYCO.COM)
ENVIRONMENT 15: SYS3.CHICAGO.MYCO=SYS3.CHICAGO.MYCO
GENERIC ENVIRONMENT X@

:DSLINE @.@.@;CLOSE                  **closes all the NS envs and ARPA
                                       domain EnvIDs with at least two “.”s. **

ENVIRONMENT 2: BOB.TEST.TEST=BOB(ROBERT.CUP.MYCO.COM)
ENVIRONMENT 15: SYS3.CHICAGO.MYCO=SYS3.CHICAGO.MYCO

DSLINE X@;RESET                      **resets defaults for generic
GENERIC ENVIRONMENT X@                  environment ID **
```

REMOTE HELLO Command

Creates a session on a remote node.

Syntax

```
DSLINE  [:envID                ]          [          {envID               }]
        [:[envID=]nodename][ logon [;DSLINE={[envID=}nodename}]
        [envnum                ]          [          {#Lenvnum            }]
```

Use

Available	In Session?	Yes
	In Job?	Yes
	In Break?	Yes
	Programmatically?	Yes
Breakable?		No
Capabilities?		None

Parameters

envID An environment ID—that is, a character string representing a specific session on a remote node. For NS names, the environment name itself may optionally be qualified: *envname*[.*domain*[.*organization*]]. Each portion of the string may have a maximum of 16 alphanumeric characters (including underscores and hyphens), of which the first must be alphabetic. The default *domain* and *organization* names are those specified for your local node when it was configured as part of its NS 3000/iX network. For ARPA domain names, the environment ID has the syntax `label[.label[...]`. The labels must follow the syntax for ARPANET host names. Refer to ARPA Domain Name Syntax in Chapter 1, “Introduction to NS 3000/iX,” for more details.

If the *envID* is not equated with a node name, it must refer to a previously defined environment. If it is equated to a node name, it then represents a session on that node. If the *nodename* is used by itself, it then becomes its own environment ID, representing a particular session on that node. If *envID*, *nodename*, and *envnum* are all omitted in the beginning of the command line (before `HELLO`), the environment information must be given in the `;DSLINE=` option at the end of the command line or the default environment will be

assumed. The default environment for a *REMOTE* command is the one most recently referenced in a **DSLINE** or **REMOTE** command.

nodename When you are using NS names, the *nodename* is the name assigned to the remote node when it is configured into the NS 3000/iX network. This name may optionally be qualified in the format *node*[.*domain*[.*organization*]]. The default *domain* and *organization* are those of the local node. Each portion of this string may have a maximum of 16 alphanumeric characters (including underscores and hyphens), of which the first must be alphabetic. When you are using ARPA domain names, the nodename has the syntax `label[.label[...]`. The labels must follow the syntax for ARPANET host names. Refer to ARPA Domain Name Syntax in Chapter 1, "Introduction to NS 3000/iX," for more details.

An environment ID may be equated with this node name, or the node name (if used alone) may become its own environment ID. In either case, the environment ID then represents a specific remote session on this node. Default: the environment specified by the last **DSLINE** or **REMOTE** command.

envnum The number of the environment assigned when the environment was defined. This is the environment number listed in the message that appears after a **DSLINE** command. Note that when *envnum* is specified immediately after `REMOTE`, *envnum* is specified without the `#L` prefix.

logon A valid logon sequence for the remote node, in the form `HELLO` *user.account*[,*group*]. For information on additional MPE logon parameters and options, please see the *MPE/iX Commands Reference Manual*.

`DSLINE=` Defines an environment if one is not specified immediately after `REMOTE`. The parameters are used in the same way as they are after `REMOTE` (or as they are used in a **DSLINE** command).

Description

The **REMOTE HELLO** command creates a session on a remote node. If the remote environment (session) has already been defined in a previous **DSLINE** (or **REMOTE**) command, the environment ID or number may be used by itself to designate the environment. Otherwise, an environment ID may be equated to an actual node name or the node name may be used by itself as its own environment ID. The environment information

may be given immediately after the REMOTE or after DSLINE=. If the environment information is given in both places, the specification following REMOTE takes precedence. If no environment information appears in either place, the default environment assumed is the one most recently referenced by a **REMOTE** or **DSLINE** command. If you issue a **REMOTE HELLO** command for an environment that you are already logged on to, the existing remote session will be terminated and a new one will be created (according to the new user information). This is similar to what happens when you issue a new **HELLO** command within a local session. If you want a second concurrent session on the same node, you can designate a new environment ID for this node, either in a **DSLINE** command or in a **REMOTE HELLO** command.

NOTE

Before you can create a remote session on another node in an NS 3000/iX network, your system operator must have issued the following two console commands: NETCONTROL to open the communications links to the remote node, and NSCONTROL to enable Virtual Terminal service. These commands must be issued before any network service can function between the local and remote nodes. See the *NS 3000/iX Operations and Maintenance Reference Manual*.

Examples of REMOTE HELLO

The following examples show you the different ways that you can use the **REMOTE HELLO** command. The input is typed at the MPE/iX prompt, and user input is bold for clarity.

1. Method 1 shows that the **REMOTE HELLO** command (without environment information) may appear after a **DSLINE** command that provides the environment information.

   Method 1

   ```
   :DSLINE SYS2
   ENVIRONMENT 1: SYS2.DCL.DETROIT
   :REMOTE HELLO NSUSER.NSACCT **default environment**
   HP3000 / MPE/iX A.01.00 FRI, MAY 1, 1994, 4:13 PM
   ```

2. Method 2 shows that the environment information may be included in the DSLINE= portion of the **REMOTE HELLO** command.

   Method 2

   ```
   :REMOTE HELLO NSUSER.NSACCT;DSLINE=E2=SYS2
   HP3000 / MPE/iX A.01.00 FRI, MAY 1, 1994, 4:14 PM
   ENVIRONMENT 2: E2.DCL.DETROIT=SYS2.DCL.DETROIT
   ```

3. Method 3 shows that the environment information may appear immediately after REMOTE on the same command line.

   Method 3

   ```
   :REMOTE:CHEKHOV
   CHEKHOV#HELLO ANTON.PLAYS
   HP3000 / MPE/iX A.01.00 FRI, MAY 1, 1994, 4:26 PM
   ```

   or

```
:REMOTE:CHEKHOV HELLO ANTON.PLAYS
HP3000 / MPE/iX A.01.00 FRI, MAY 1, 1994, 4:26 PM
ENVIRONMENT 3: CHEKHOV.DCL.DETROIT
:REMOTE           **allows use of environment**
```

4. The **REMOTE HELLO** command may be issued in two parts, with the HELLO portion following the remote prompt which is returned. The environment specification may appear in a previous **DSLINE** command, as in method 1, or immediately after the REMOTE, as in method 3. The DSLINE= option is not legal in this case, since the DSLINE= parameter must follow the HELLO in the **REMOTE** command line.

5. Methods 4a and 4b show two different ways of accessing the same node. The **DSLINE SYS5** command sets the attributes of node SYS5 for both Method 4a and Method 4b. The PROMPT=>> part of the command shows the effect of setting the prompt on the remote node. Method 4a shows that the **REMOTE** and **HELLO** commands can be on separate lines. Method 4b shows that the environment ID is set equal to the node name in order to create a new session. A fifth environment is created. Upon returning to the local system, a status line displays the environment information.

   Methods 4a and 4b

```
:DSLINE SYS5;PROMPT=>>    **DSLINE for Methods 4a and 4b**
ENVIRONMENT 4: SYS5.DCL.DETROIT
```

   Method 4a

```
:REMOTE **default environment**
>>HELLO NSUSER.NSACCT
HP3000 / MPE/iX A.01.00 FRI, MAY 1, 1994, 4:51 PM
```

   or

   Method 4b

```
:REMOTE:MARIE=SYS5 **creates a new session**
MARIE#HELLO NEWUSER.NEWACCT
HP3000 / MPE/iX G.OO.OO FRI, MAY 1, 1994, 4:54 PM
MARIE#:
ENVIRONMENT 5: MARIE.DCL.DETROIT=SYS5.DCL.DETROIT
```

Releasing a Remote Environment

The method that you use to release a remote environment depends upon the way it was initially created:

1. If the environment was defined by a **DSLINE** command, it can be released only by a corresponding **DSLINE;CLOSE** command.
2. If the environment was defined in a **REMOTE HELLO** command, it can be released by either a **REMOTE BYE** command or a **DSLINE;CLOSE** command.
3. If you terminate the local session (with a local BYE or ABORTJOB) without explicitly releasing the remote environment, (a) all open files and active processes on the remote node will be closed or terminated, and (b) an implicit DSLINE @.@.@;CLOSE will be executed. (You will not be asked whether you wish to abort the remote session.)

If you issue a **DSLINE;CLOSE** command (without a REMOTE BYE) after a session has been established on a remote MPE/iX system, and if the option is not in effect, you will be asked:

```
ABORT REMOTE SESSION ON envID?
```

A **Y[ES]** response will abort the remote session and terminate the connection. A **N[O]** answer will cancel the **DSLINE ;CLOSE** command and leave the remote session intact. DSLINE ;CLOSE may not be issued while remote files are open or while dependent RPM processes exist on the remote node. The appropriate FCLOSE, TERMINATE, or RPMKILL intrinsic must be executed first or an error message will appear. For information on Remote File Access, and Remote Process Management, see the corresponding chapters of this manual.

NOTE

A **REMOTE BYE** command may be viewed as a particular instance of a command to a remote system. You can issue a remote command through the Virtual Terminal service once a remote session has been established via a **REMOTE HELLO**. (See the explanation of the **REMOTE** command in this chapter.)

Examples

In the following examples, all commands that are shown are typed at the MPE/iX prompt.

Method 1

```
DSLINE MODELT
REMOTE HELLO HENRY.FORD
REMOTE BYE
DSLINE;CLOSE
```

or (user input is bold for clarity)

```
DSLINE MODELT
REMOTE HELLO HENRY.FORD
DSLINE;CLOSE
ABORT THE REMOTE SESSION ON MODELT.DCL.DETROIT?YES
SESSION ABORTED BY SYSTEM MANAGEMENT
```

Method 2

```
REMOTE HELLO MARIE.SYS5;DSLINE=RADIUM
REMOTE BYE
```

or

```
REMOTE:RADIUM HELLO MARIE.SYS5
REMOTE BYE
```

Method 3

```
:DSLINE MODELA
:REMOTE HELLO HENRY.FORD
:BYE <F14P10M>**implicit :DSLINE;CLOSE executed**
```

or

```
:REMOTE HELLO HENRY.FORD;DSLINE=MODELA
:BYE
```

or

```
:REMOTE:MODELA HELLO HENRY.FORD
:BYE
```

REMOTE Command

Allows commands to be executed in a remote environment.

Syntax

REMOTE [:[*envID*][*command*] [*envnum*]

Use

Available	In Session?	Yes
	In Job?	Yes
	In Break?	Yes
	Programmatically?	Yes
Breakable?		No
Capabilities?		None

Parameters

envID The environment ID representing an established session on the remote node. This environment ID may be an actual node name. If both *envID* and *envnum* are omitted, the default environment is the one most recently referenced in a **DSLINE** or **REMOTE** command.

envnum The number of the environment assigned when the environment was defined. This environment number is listed in the message returned after a **DSLINE** command.

command A command that is to be executed in the remote environment; for example an MPE/iX command to be executed on a remote HP 3000.

Description

After you have established a remote environment (session) on a node, the **REMOTE** command allows you to issue commands in that remote environment. Because you can have several remote environments on a node at the same time, you use the **REMOTE** command along with a unique environment ID to specify one of several remote environments. If you issue **REMOTE** commands without specifying an *envID* or *envnum*, the default environment (the one most recently invoked) will be used.

NOTE If an RPM-created process is already executing in a remote environment, you cannot issue a **REMOTE** command for that environment.

If a *command* parameter is included in the REMOTE command line, the remote operating system executes it and restores control to the local operating system. The local prompt reappears on your terminal screen. For example, at the MPE/iX prompt (which is shown for clarity), type the following command (user input is bold):

```
:REMOTE LISTF **default environment**
   .
   .**list of file names from
   .**remote environment appears**
: **local prompt reappears**
```

If no *command* parameter is specified, a remote prompt is issued and the remote operating system retains control. You can then send commands to the remote system by entering them at this prompt without a preliminary REMOTE. The remote prompt reappears after the execution of each command until you enter a colon (:) at the remote prompt. This restores the local MPE/iX prompt, at which you can issue a subsequent REMOTE command or a local command. Suppose that the remote prompt is ENV1#. At the local MPE/iX prompt (which is shown for clarity), type the following commands (user input is bold):

```
:REMOTE**default environment already set to ENV1#**
ENV1#LISTF**executed in the remote environment**
   .**list of file names from node ENV1 appears**
   .
ENV1#:**typing a colon (:) restores the local prompt**
:REMOTE BYE
```

You can configure your own remote prompt (1 to 8 characters) by using the DSLINE ;PROMPT= option. If you wish to receive the remote system's local prompt (for instance, a colon [:]), specify DSLINE ;PROMPT= without a prompt string. Following is an example of a user-specified prompt (the local and remote prompts are shown for clarity, and user input is underlined):

```
:DSLINE SPOTS;PROMPT=VPRES>**default environment**
:REMOTE                **now set to node SPOTS
MPE /iX-SPOTS:HELLO NSUSER.NSACCT
```

If you use the MPE/iX **SETVAR HPPROMPT** command to set a system prompt to anything other than a colon, then that prompt will override any prompt created by the **DSLINE ;PROMPT=** option. Details on use of the **SETVAR** command are in the **MPE/iX Commands Reference Manual**.

If you use SETVAR HPPROMPT, then leave the remote session and return to it, the remote system's prompt will be displayed until you press **[Return]**. In the following example, user input is underlined, and the local and remote MPE/iX prompts are shown for clarity:

```
:REMOTE REMENV HELLO USER.ACCT
:REMOTE
REMENV#SETVAR HPPROMPT "XYZ>>"
XYZ>>**new prompt replaces REMENV#**
XYZ>>:**colon (:) returns you to local session**
:REMOTE**to return to remote**
REMENV# [Return]**default prompt orients you**
XYZ>>>**returns to prompt set by SETVAR
       HPPROMPT**
```

Using DSLINE and REMOTE Within Programs

Both the **DSLINE** and **REMOTE** commands can be executed within a program using the COMMAND intrinsic. You can set up environment attributes, log on remote sessions, and issue remote commands from within a program. For the format of the COMMAND intrinsic, refer to the *MPE/iX Intrinsics Reference Manual*.

NS 3000/iX maintains separate default environments for each process within a job or session. A **DSLINE** or **REMOTE** command that explicitly specifies an environment ID must be issued from the process to set up its default environment. Subsequent **DSLINE** or **REMOTE** commands issued from the process may then use (or change) the process default environment. For example, suppose an application has two processes that set up remote sessions on different nodes. These processes could execute the following command sequences using the COMMAND intrinsic:

Process 1

```
DSLINE NODE1 {sets default environment for process 1}
REMOTE HELLO USER1.ACCT1 {uses default environment NODE1}
```

Process 2

```
DSLINE NODE2 {sets default environment for process 2}
REMOTE HELLO USER2.ACCT2 {uses default environment NODE2}
```

Because NS 3000/iX maintains separate default environments for each of these processes, they can execute concurrently with no problems of confusion between the default environments.

DSLINE commands for the same environment can be issued from several different processes within a job or session. Attributes defined by the **DSLINE** commands take effect for all processes within the job or session. For example, if a DSLINE from one process specifies the COMP option, the COMP attribute is set for all processes using the environment.

NS 3000/iX keeps track of each process that issues a **DSLINE** command for an environment. The environment will not be deleted until all the processes have issued **DSLINE;CLOSE** commands. If any process terminates without issuing the DSLINE;CLOSE, one will be implicitly performed for the environment.

Only environments defined by **DSLINE** or **REMOTE** commands within a process will be displayed by generic **DSLINE** commands. For example, if you issue a **DSLINE @;SHOW** command from a process, only those environments defined within the process will be displayed.

Several processes within a job or session can issue **REMOTE** commands to the same remote session. However, this practice is not advised, since the commands will be unpredictably interleaved depending on the order

of execution of the processes. Also, if one process is already executing a **REMOTE** command to a remote session, a **REMOTE** command to the same remote session from another process will fail.

Concurrent execution of **REMOTE** commands by multiple processes within the same job or session is not supported. This applies whether the processes are communicating with the same or different remote sessions. One process should complete execution of the **REMOTE** command and return to local mode before a second process attempts execution of a **REMOTE** command.

Reverse Virtual Terminal

The Reverse Virtual Terminal (Reverse VT) service allows an application program to receive information from and send information to terminals located on other systems. All the systems involved must be connected via NS 3000 connections (either NS 3000/V or NS 3000/iX). The Reverse VT service must be initiated from the system on which the application resides.

Two important points for Reverse VT are as follows:

- Reverse VT is supported for terminal type 10 only;
- The terminal must be *available* in order to be opened successfully by the application. That is, no one can be logged on and no other application can be accessing the terminal. Pressing **[Return]**, for example, will make a terminal unavailable (the system is waiting for a logon attempt) until the logon timer expires.

To gain access to a remote terminal via Reverse VT, you can specify the VTERM option in the **FILE** command, which designates the terminal as a remote device. Or the application program itself may include the VTERM option in the *device* parameter of the FOPEN intrinsic which opens the connection to the device. (For the syntax of the **FILE** command and the FOPEN intrinsic, when used to access remote files and devices, see the Remote File Access chapter of this manual.)

The format for the file equation is either:

```
FILE X=X:envID;DEV=#ldev;VTERM     **8-character environment ID**
```

or

```
FILE X=X;DEV=envID#ldev;VTERM     **8-character environment ID**
```

The format for the FOPEN device parameter is:

```
#ldev;VTERM [Return]      **must be terminated by ASCII value
                               for carriage return**
```

In the FOPEN call, the location of the device may be specified either in the *formaldesignator* parameter (X:*envID*) or in the *device* parameter (*envID#ldev*;VTERM **[Return]**). If the FOPEN call indicates the location of the file, you can specify the VTERM option in a file equation issued directly on the remote terminal's node:

```
ENV1#FILE X;DEV=# ldev;VTERM
```

The *ldev* parameter is either the device class name or the logical device number of the remote terminal. If you specify a device class name rather than the logical device number of a terminal, the first available terminal in the device class table will be used.

Because the VTERM option is specified, the application program communicates with the remote terminal by way of the Virtual Terminal rather than by way of Remote File Access. In both VT and RFA, the remote terminals function as non-session I/O devices. With RFA, you have to create a remote session on the node where the terminal resides. With Reverse Virtual Terminal, however, the application program does not establish a remote session on the node where the terminal resides. The terminal users cannot call subsystems or issue commands to the remote system. A privileged mode program can communicate with the terminal in nowait mode by setting bit 4 in the FOPEN *aoption* parameter. The application program will not recognize a **[BREAK]** or **[CTRL]-Y** issued from the remote terminal's side of the connection.

Using the Remote Subsystem from a Batch Job

While in a batch job, you can establish a remote session by using the **DSLINE** or **REMOTE HELLO** command. The job streamed may be similar to the following (local and remote prompts are shown for clarity):

```
JOB USER.ACCOUNT
:DSLINE REMOTE2
:REMOTE HELLO RUSER.RACCOUNT
:REMOTE
#FILE OUT;DEV=LP
#BUILD WORK;DISC=50
#RUN USERPROG
#PURGE WORK
#:
:REMOTE BYE
:DSLINE;CLOSE
:EOJ
```

NOTE The remote # prompt is optional. It allows you to more easily identify remote commands in a job stream.

Remember that once a remote session has been established, it interacts with the job in the same way as it would interact with a terminal. If the remote session detects an error, the error will be printed to $STDLIST. If the error generates a user prompt, the next record in the job file is read as the response (in the same manner as waiting for a character or **[Return]** on a terminal). That record is then lost to the job.

BREAK

Pressing **[BREAK]** while a remote command is being executed suspends execution of the command and returns control to the environment from which you issued the command. For example, if you enter a command from a remote prompt, and then press **[BREAK]**, the system will return your remote prompt. When an application program is suspended by **[BREAK]**, the system will return the prompt that was last issued before the program was run. For example (user input is bold, and the local and remote prompts are shown for clarity):

```
:REMOTE EDITOR
    .
    .
[BREAK]
:SHOWME **returns to local prompt**
```

or

```
:REMOTE
ENV1#EDITOR
    .

[BREAK]
:ENV1#SHOWME **returns to remote prompt**
```

NOTE The response of a remote system to a **[BREAK]** request may depend on the time required to transmit the **[BREAK]**.

3 Remote File Access

The Remote File Access service (RFA) allows you to access remote files and devices. By using RFA you can, among other things, create, open, read, write, and close a file that resides on a remote HP 3000. Since the remote "file" may be a peripheral device, you can, for example, read from a tape mounted on a remote system or print local data on a remote printer.The Remote File Access facility uses the same MPE/iX file system intrinsics as are employed on a local system. The intrinsics are sent to the remote environment and executed there. Your local program can call these intrinsics explicitly or it can use the input/output procedures specific to the language in which the program is written. You can also access a remote file or device interactively. You will find discussions of interactive and programmatic remote file access methods later in this chapter.

Limitations

Following are RFA limitations:

1. RFA does not permit nowait (asynchronous) I/O.
2. RFA only works with filenames in the traditional MPE namespace. It does not work with filenames in the HFS (POSIX-compliant) namespace. RFA does not recognize HFS directories or filenames that contain slashes as directory diameters. For example, it will not work properly with filenames such as "`/FILEa,`" or "`/usr/include/stdio.h.`"
3. RFA only works with traditional MPE record-oriented files; that is files with fixed (F), variable (V), or undefined (U) record types. It does not work with files that have the POSIX-compliant, bytestream (B) or directory (H) types.

RFA Compression

Remote File Access (RFA) allows for data compression. RFA data compression can provide faster data transfer, especially over a slow link and when the file being remotely accessed has large records with repeated characters.To invoke data compression for the RFA service, specify the keyword, COMP, in the **DSLINE** command. The COMP keyword compresses both NFT and RFA data associated to the remote environment specified in the **DSLINE** command. For example, issuing the command **DSLINE NODE1;COMP** allows for compression of all RFA and NFT data to the remote node called NODE1.

To terminate data compression to the specified environment, specify the keyword NOCOMP in the on of any new remote files opened as well as prevents data compression of any future NFT data to the environment specified in the **DSLINE** command. Files opened remotely while compression is in effect for the environment remain in compressed mode even after the **DSLINE;NOCOMP** command is issued.

The same algorithm is used for both RFA and NFT data compression. The same algorithm is also used with DS Services compression: each sequence of up to 64 consecutive repeated characters is compressed to two bytes, and repeated blanks are compressed to one byte.

Without RFA compression enabled, the RFA service can transfer up to 29,900 bytes of data at one time. However, due to overhead associated with managing a compressed data transfer, the RFA service with compression enabled can transfer up to a maximum of 29,000 bytes of data at one time between MPE/iX-based HP 3000s. For transfers between an MPE/iX-based system and an MPE-V based system the maximum amount of compressed data that can be transferred at one time is 14,500 bytes.

RFA compression only compresses data transmitted with the FREAD, FREADDIR, FREADBACKWARD, FWRITE, FWRITEDIR and FUDPATE intrinsics. Therefore, RFA compression is not supported with the Remote DataBase Access (RDBA) service.

RFA compression is currently only available on HP 3000 Series 900 systems with version B.00.05 or later of the NS 3000/iX Network Services. If you request RFA compression of a file residing on a system that does not support RFA compression (such as systems running NS 3000/V software released prior to version A.00.12), RFA completes the request without compressing the RFA data. No error or warning messages are issued in this situation.

FILE Command

Specifies a formal designator that may be used to represent a remote file or device in a subsequent command or intrinsic. (Also known as a file equation.)

Syntax

```
                [=*formaldesignator                                  ]
                [=$NEWPASS                                           ]
                [=$OLDPASS                                           ]
FILE formaldesignator
                [=$STDIN                                             ]
                [=$STDINX                                            ]
                [=$STDLIST                                           ]
                [=filereferenc[:nodespec]{,filedomain]               ]
[;DEV=[[envname]#][device][,outpri][,numcopies]]
[;VTERM]
[;ENV=envfile[:nodespec]]
[;option] . . .
```

Use

Available	In Session?	Yes
	In Job?	Yes
	In Break?	Yes
	Programmatically?	Yes
Breakable?		No
Capabilities?		None

Parameters

formaldesignator A name in the form *file*[.*group*[.*account*]][:*nodespec*] that can be used to identify the file in a subsequent command or intrinsic call. (For the meaning of *nodespec*, see the next parameter explanation. MPE/iX currently permits this extended formal designator, with a node specification following a colon, in the **FILE** and **RESET** commands, and in the FOPEN intrinsic.) If not equated to another file designator, the *formaldesignator* contains the actual name of a file. A **formaldesignator* (with the asterisk) is a "backreference" to a formal designator defined in a previous **FILE** command.

nodespec Either an environment ID (specified in a previous **DSLINE** or **REMOTE** command) or $BACK. This node specification may appear in the file's formal designator or as an extension of an actual file reference. If an environment ID appears in a file designation and in the DEV= option, the attempt to open the file (for example, via FOPEN) will result in an error.

	$BACK indicates that the file resides one "hop" back toward your local system. This is legal only if the **FILE** command is issued in a remote session created by a REMOTE HELLO. The $BACK specification is equivalent to DEV=# (without an environment name). In either case, the file is accessed through the existing session.
filereference	The actual name of the file in the form: *file*[/*lockword*][.*group*[.*account*]].
filedomain	The file domain: NEW or OLD or OLDTEMP.
envname	An unqualified environment ID. The maximum length is 8 alphanumeric characters. A previously defined environment ID is permitted in the DEV= option, but the domain and organization qualifiers are not permitted and the name may not be longer than 8 characters.
device	The logical device name or number of a device such as a disc, tape, printer, or terminal. Default: DDISC. If the DEV= option appears, it must be followed by at least one parameter (which can be just #).
outpri	The output priority requested for a spooled device file. This a value between 1 (lowest priority) and 13 (highest priority).
numcopies	The number of copies requested for a spooled output device file (maximum 127).
VTERM	Specifies that the Reverse Virtual Terminal service should be employed instead of Remote File Access. This option applies only if the designated device is a remote terminal. VTERM allows a local application program to perform I/O to remote terminals located on systems that support Reverse Virtual Terminal. (See "Reverse Virtual Terminal" in the chapter on "Virtual Terminal" and "Remote Terminal Access: VT vs. RFA" later in this chapter.)
envfile	A name representing a file containing laser printer environment information, which controls printing output formats. This name may be an actual file reference or a formal file designator (preceded by an asterisk).
option	Any valid option in the MPE/iX **FILE** command. For further information, see the *MPE/iX Commands Reference Manual*.

Description

For Remote File Access purposes, the **FILE** command can be used to specify a formal designator for a remote file or device. You can use this formal designator to reference the remote file in a subsequent command or intrinsic call. If an environment ID is used to indicate the location of the file, it must be specified in a **DSLINE** or **REMOTE** command before you can use RFA. $BACK or DEV=# indicates the node one "hop" closer to your local system when the **FILE** command has been issued in a remote session. (This may be the local system itself.)

Precautions When Using $BACK

When using the $BACK backreference with RFA, you need to check the fully qualified node names of the machines on each side of the file transfer. If the domain and organization names differ between the two machines, problems may arise with use of $BACK.

To prevent a problem when using $BACK for a transfer between two nodes whose domain and organization are different, configure the remote machine (using NMMGR; NM capability required), so that its network directory includes two entries:

1) *localnode*.*localdomain*.*localorganization*, and
2) *localnode*.*remotedomain*.*remoteorganization*.

See the example under "Interactive Access" later in this chapter.

RESET Command

Cancels file equations.

Syntax

```
RESET {formaldesignator }
      {@              }]
```

Use

Available	In Session?	Yes
	In Job?	Yes
	In Break?	Yes
	Programmatically?	Yes
Breakable?		No
Capabilities?		None

Parameter

formaldesignator A formal file name in the form:

file[.*group*[.*account*]][:*nodespec*], for which a **FILE** command has previously been issued. The nodespec portion is either an environment ID indicating the location of the file or $BACK. $BACK means that the file resides one "hop" back toward your local system (which may be the local system itself).

@ Signifies all formal file designators specified in all **FILE** commands previously issued in this session or job.

Description

The **RESET** command resets a formal file designator to its original meaning, cancelling any **FILE** command that has been issued for this formal designator earlier in this session or job.

Interactive Access

In order to access a remote file or device interactively, you must first issue a **FILE** command that specifies the remote location of the file. However, you cannot indicate the location directly in the MPE/iX command or subsystem command that accesses the file.

Example 1

Let's say that you wish to print a text file named DOCUMENT on a line printer connected to a remote HP 3000 computer. You are editing the text file on your local system. After defining an environment on the remote node, you can issue a local **FILE** command at the MPE/iX prompt that designates the line printer as a remote device and specifies the environment in which it exists. You must also log on to the remote node:

```
DSLINE NIKOLAI
FILE REMPRINT;DEV = NIKOLAI#LP
REMOTE HELLO NSUSER.NSACCT
```

You can then send your finished TDP file to the remote line printer as follows. The local MPE/iX and TDP prompts are shown, and user input is underlined for clarity.

```
:RUN TDP.PUB.SYS
/FINAL FROM DOCUMENT TO *REMPRINT
```

Example 2

```
DSLINE NIKOLAI
REMOTE HELLO USER.ACCT
FILE SOURCE=XYZ:NIKOLAI
PASCAL *SOURCE
```

You may want to compile a remote Pascal source file on your local system. This is how you can do it from the MPE/iX prompt:

Example 3

Let's assume that you have created a session on a remote node by typing at the MPE/iX prompt:

```
DSLINE NIKOLAI
REMOTE HELLO USER.ACCT
```

In order to access a file on your local node, you may use the $BACK specification. Here's an example of that method. The local and remote MPE/iX prompts are shown, and user input is underlined for clarity.

```
:REMOTE
NIKOLAI#FILE SOURCE=XYZ:$BACK
NIKOLAI#PASCAL *SOURCE
```

Example 4

A problem may occur when using $BACK for a transfer between two nodes whose domains and organizations are different. To eliminate a problem with the use of $BACK when a transfer is being made between

two nodes whose domains and organizations are different: configure the remote machine (using NMMGR; NM capability required), so that its network directory includes two entries:

1) *localnode.localdomain.localorganization*, and
2) *localnode.remotedomain.remoteorganization*.

For example, if you were to issue a DSLINE from node A.LAB.CND to node B.SJ.CA, you would have to add the following entries to the network directory of node B.SJ.CA:

```
A.LAB.CND
A.SJ.CA
```

RFA Programmatic Access

Once an environment has been established on the remote node, a local application program can access remote files by calling standard MPE/iX file system intrinsics (or by using the input/output procedures specific to the language in which the program is written). If a **FILE** command specifying a formal designator for a remote file or device has been issued, an **FOPEN** call in the local program can use this formal designator in its *formaldesignator* parameter. For example:

```
:FILE X=X:NODEB
.
.
>
FOPNE (X,...);
```

A language-specific I/O procedure can also reference the file by means of this *formaldesignator*.

In Pascal, the file name used in the program can include the *nodespec* as follows:

```
OPEN (X, 'X:NODEB');
```

In the Pascal example, the file equation is not needed. Most language-specific file open statements do not permit a nodespec and must use a preceding **FILE** command.

If a **FILE** command has not been issued for the remote file, you must specify the location of the file in the FOPEN call, either in the *formaldesignator* parameter or in the *device* parameter (not both). (Currently, only in Pascal can you use the extended formal designator, with location, in a non-intrinsic I/O procedure.) These are the two possibilities:

- *formaldesignator*: *file*[/*lockword*][.*group*[.*account*]][:*nodespec*], where *nodespec* is an environment ID or $BACK, as defined for a **FILE** command;
- *device*: [*envname*]#[*device*], where *envname* is an 8-character (or shorter) string as defined for a **FILE** command.

For example:

```
FOPEN (X.NODEB,...)
or
FOPEN (X,...,NODEB#,...);
```

You can also use a file equation to override the location indicated in your program (**FILE** command parameters override FOPEN parameters). For example, the following sequence opens a file on NODEC:

```
:FILE X:NODEB=X:NODEC
.
.
.
FOPEN 9X:NODEB,...):
```

You can call the MPE/iX FFILEINFO intrinsic to retrieve information about a remote file. In the FFILEINFO intrinsic, ITEM 61 returns the environment ID of a remote file's location — over an NS link. If the file is located on your local system, ITEM 61 returns a blank. The condition codes for the file system intrinsics retain their normal meanings. Network connection errors return a CCL condition code. If such an error occurs, you can call the MPE/iX FCHECK intrinsic to determine the source of the error. File System error codes apply to the remote file. You can also call the MPE/iX PRINTFILEINFO intrinsic to display the status of the remote file.

NOTE

To ensure that the formal designator representing a remote file is syntactically correct, you should always call the FPARSE intrinsic within your program. This intrinsic is documented later in this chapter. For further information on MPE/iX file system intrinsics, including FOPEN, FPARSE, IOWAIT, and IODONTWAIT, see the *MPE/iX Intrinsics Reference Manual*.

FPARSC Intrinsic

Parses a file designator and determines whether it is syntactically correct.

Syntax

FPARSE (*string*,***result***[,*items*][,***vectors***])

Parameters

string (input) **Byte array, by reference**. The file designator that is to be parsed. The *string* can be terminated by any non-alphanumeric character except a slash, period, colon, underscore, or dash (`/, ., :, _, -`).

result (output) **Array of two 16-bit integers, by reference**. A value indicating the result of the parse, returned in the first 16-bit word. (The second 16 bits are reserved.) If the value is positive, the file *string* is syntactically correct and the value indicates the type of file reference being made:

- `0` = regular formal designator
- `1` = back reference (first character is *)
- `2` = system file (first character is $)

A negative value represents the error code returned. The error messages are listed in the *NS 3000/iX Error Messages Reference Manual*.

items (input) Array of 16-bit integers, by reference. An array of item codes representing portions of the file *string*. The item codes (except for 0) may be included in any order; the same order is followed in the output *vectors* array. Item code 0 indicates the end of the *items* array.

The item codes have the following definitions:

```
0 = indicateds end of this array
1 = file name
2 = lockwood
3 = group name
4 = account name
5 = environment ID
```

Note that an initial $ is considered part of the file name portion, while * is not.

vectors (output) Array of 16-bit integers, by reference. Gives offset and length information that indicates the parse of the file *string*. Each element of the array is a 16-bit integer; each pair of elements corresponds to a portion of the *string*, in the same order as the *items* in the *items* array. The first 16-bit integer of each pair is the byte offset of the item (from the start of the *string* to the start of the item), and the second 16-bit integer is the item's length in bytes. A zero value (in both 16-bit integers) means that the item is not present in the file *string*.

For the last element of the vectors array, which corresponds to item code 0 in the *items* array, the second 16-bit integer is the total length of the file *string*. The first 16-bit integer of this element is zero unless the `result` parameter indicates that the file is a system file; in that case the value is the file's default designator type as defined for the FOPEN *foptions* parameter

```
0 = non-system filename
1 = $STDLIST
2 = $NEWPASS
3 = $OLDPASS
4 = $STDIN
5 = $STDINX
6 = $NULL
```

In case of a syntax error in the file designator *string*, the first word of the first 16 bits of the *vectors* array returns the byte offset of the invalid item in the *string*. The second 16 bits will be zero.

Description

The FPARSE intrinsic parses a file designator and indicates whether or not the string is syntactically correct. You can employ this intrinsic to check a formal designator representing a remote file before attempting to open the file via FOPEN. The optional *items* and *vectors* arrays enable you to acquire parsing information for the file designator, namely, the length of each item and its position in the *string*. The possible *items* are file name, lockword, group name, account name, and environment ID. The environment ID is treated as a single item; it is not parsed into environment name, domain, and organization.

Following are examples of the *items* (input) and *vectors* (output) arrays. Remember that the order of entries in the *vectors* array corresponds to the (arbitrary) order of *items* in the *items* array. Also, the last pair of entries of the *vectors* array has a different meaning from that of the other pairs: the second 16 bits gives the total length of the file *string*, and the first 16 bits gives a system file code when applicable.

In the first example the file string is:
`FILENAME/LOCKWORD.GROUP.ACCOUNT:CASH.ACCTNG.FINANCE`.

Example 1

items Array (item code)	Vectors Array (offset, length)
1	0, 8
5	32, 19
3	18, 5
4	24, 7
2	9, 8
0	0.51

In the second example, the file string is `*FILENAME:CASH`.

Example 2

items Array (item code)	Vectors Array (offset, length)
1	1, 8
2	0, 0
3	0, 0
4	0, 0
5	10, 4
0	0.15

In the third example, the file string is `$OLDPASS`.

Example 3

items Array (item code)	Vectors Array (offset, length)
1	0, 8
2	0, 0
3	0, 0
4	0, 0
5	0, 0
0	3, 8

Example RFA Program

What follows is a Pascal program that writes some test data to a remote file, reads the data back from the remote file, and sends the received data to a remote printer to be printed. An environment has been established on the remote node. Prior **FILE** commands have been issued for the formal file designators representing the remote disk file and remote printer. These file equations specify the remote location of the files.

```
$standard_level 'hp3000'$
$uslinit$
{*********************************}
{Note: issue file equations such as}
{     :file remfile=remfile:rodan}
{     and}
(     :file remprint;dev=rodan#lp}
{before running this program}
{*********************************}
program rfaprog(remfile,remprint,output);
type
          teststring   =   packed array [1..72] of char;
          smallint     =   -32768..32767;
var
          remfile      :   text;
          remprint     :   text;
          i            :   integer;
          test1        :   teststring;
          test2        :   teststring;
begin {program rfaprog}
{open remote disc file}
writeln('Opening remote disc file');
rewrite (remfile);
{write test data to remote file}
test1: = 'Remote File Access test';
for i: = 1 to 9 do
writeln (remfile,test1);
{open remote line printer as devicefile}
writeln('Opening remote LP file');
reset (remfile);
{read each record from remote file, then print each record on remote printer}
rewrite (remprint);
```

```
for i := 1 to 9
          begin
          readln (remfile,test2);
          writeln (remprint,test2);
{pick up listing on remote line printer and check output}
          end;
end {program rfaprog}.  {remote files/devices closed automatically}
```

Remote Terminal Access: VT vs. RFA

Both VT and RFA can be used to access remote terminals. In either case, a **FILE** command or FOPEN call must indicate that the file in question is actually a remote terminal. If you specify the VTERM option in the **FILE** command or the device parameter of the FOPEN call, the terminal will be accessed through Reverse VT rather than RFA. In both cases the remote terminal functions as a non-session I/O device. If the remote terminal is accessed through Reverse VT, you do not establish a session on the system to which the terminal is attached.

RFA/RDBA Automatic logon

A REMOTE HELLO is no longer necessary before using RFA. This is useful if your system does not support the Virtual Terminal service.

Overview

RFA requires a remote session when accessing or creating remote files to provide the file system security of the remote user. This session is referred to as an "environment." An environment is identified by either a DSLINE *environment ID* that refers to a VT session, or by a logon string that automatically creates a remote session. RFA will use an existing session established by VT before a session is automatically logged on. If a session already exists, and a logon string is specified, the logon string will be ignored. The user is responsible for supplying RFA with an environment to operate within.

How to Use the Automatic Logon Feature

RFA automatically creates its session by using the logon sequence specified in the LOGON option of the **DSLINE** command. At the MPE/iX prompt, type:

```
DSLINE nodename; LOGON=user.acct,group
```

If a remote session does not exist, and a DSLINE LOGON string has been specified, then this logon string will be used to automatically log on a remote session for an RFA FOPEN. Whenever RFA logs on a remote session, it will also log it off after all files in that session have been closed.

Remote Hello After RFA Automatic Logon

One may still establish a remote VT session using a common DSLINE environment after RFA has automatically logged on its own remote session under the same DSLINE environment. VT commands will not be executed under a session created automatically by RFA. A session created automatically by RFA will be recognized by VT. All RFA will be performed under RFA's session and all VT will be performed independently under the VT session. Since these two sessions are independent, they may be established under different user/account names.

RFA/RDBA Autologon Example

Suppose you were to issue the command **DSLINE NODE1;LOGON=** *user1.acct1*, and then open a remote file on NODE1. The FOPEN logs on a remote session on NODE1 under *user1.acct1*.

With the remote file still open, issue a `REMOTE:NODE1 HELLO` *user2.acct2* to log on a VT session. The remote session for VT will be completely independent of the remote session for RFA even though both sessions are operating under the same `DSLINE` environment. All VT commands will be executed under *user2.acct2*. All RFA intrinsics (including new `FOPEN`s) will be under *user1.acct1*. This will continue until the last remote file on the `NODE1` environment is closed.

When the last file is closed, RFA's remote session for *user1.acct1* will automatically be logged off. The VT session for *user2.acct2* will remain intact. At this point any following `FOPEN`s on `NODE1` will be performed under the existing VT session *user2.acct2*, not *user1.acct1*. This is because an existing session is used before a new session is automatically logged on.

To open files under *user1.acct1* again, the VT session on `NODE1` would have to be logged off so that subsequent `FOPEN` to `NODE1` would use RFA's Automatic Logon feature.

System Compatibility

The RFA Automatic Logon feature will not work unless both systems support it. If one attempts to automatically log on to a system that does not support RFA Automatic Logon, the `FOPEN` call will return `FSERR 227`.

4 Remote Database Access

Remote Database Access (RDBA) is a Network Service in which you use TurboIMAGE/3000 intrinsics and utilities to access and update TurboIMAGE data bases located on remote HP 3000s. TurboIMAGE is a Hewlett-Packard database management system. TurboIMAGE intrinsics are sent to the remote node and executed in the remote environment. The database must reside on an HP 3000 (MPE V or MPE/iX based) since other IMAGE products are not fully compatible with TurboIMAGE/iX. The database must also be located entirely on a single node and cannot be distributed over several nodes.

RDBA Access Methods

There are three ways to open a remote TurboIMAGE database within a program:

- Identify the database as a remote file in a prior **FILE** command; for example, FILE *dbname=dbname:envID*;
- Use the COMMAND intrinsic to include the **FILE** command information in your program
- Create a database-access file that supplies the command, a **DSLINE** command, and a **REMOTE HELLO** command.

When the file specified in a **FILE** command is a remote database, the syntax (for the first two methods) is the same as it is for a remote file; you can specify the location of the database in the formal or actual designator or in the DEV= option. When using the third method, creating a database-access file, the **FILE** command may specify a remote database location only in the DEV= option — not in the formal or actual designator. (For details see the description of the **FILE** command in the "Remote File Access" chapter of this manual.)

With all three access methods you must call the DBOPEN intrinsic within the program to open the remote database. In the first two cases you call DBOPEN with the database root file name supplied in the **FILE** command. In the third case you use the database-access file name.

In the first case a user needs to know the location of the database. The application program that accesses the database does not need to have this information. The second case embeds the information in the application and frees the user from the responsibility of knowing the location. The third case insulates both the user and the application program from the information.

The COMMAND intrinsic allows a program to execute MPE/iX commands. The first parameter of the intrinsic is a string of characters giving a specific command. You can issue a series of COMMAND calls within your program to establish a remote session (**DSLINE** and **REMOTE HELLO** commands) and identify the remote database (**FILE** command). You can then call DBOPEN to open the database. When you have called DBCLOSE to close the remote database, you must call COMMAND again to execute **REMOTE BYE** and **DSLINE;CLOSE** commands. See the *MPE/iX Intrinsics Reference Manual* for a detailed explanation of the COMMAND intrinsic.

A database-access file contains a **FILE** command, a **DSLINE** command, and one or more remote logon commands. When you call DBOPEN with the name of this file, the remote session is established and the remote database is opened. You can then call other TurboIMAGE intrinsics to perform the desired operations upon the database.

The database-access file has the following general format:

```
FILE dbname;DEV=envId#
DSLINE envid
locuser.locacct=HELLO remuser.remacct
```

More than one local/remote logon sequence equation may be included in the file. When you call DBOPEN with the database-access file name, a remote session is established for the remote user that has been "equated" with your local logon name. (An @ sign in the remote user, account, or group name position on the right side of an equation is automatically replaced by the corresponding local name on the left. An @ sign in a local name position on the left is replaced by the name you actually used when you logged on.)

Under the database-access file method, the remote session is released automatically when the database is closed (with or without an explicit DBCLOSE call).

Before you can reference a database-access file in a program, you have to ACTIVATE it by means of the DBUTIL utility program. At the MPE/iX prompt, type the following command (user input is bold):

```
RUN DBUTIL.PUB.SYS
>>ACTIVATE dbaccessfilename
     VERIFICATION FOLLOWS:
          FILE COMMAND:     LOOKS GOOD
          DSLINE COMMAND:   LOOKS GOOD
          HELLO COMMAND:    LOOKS GOOD
     ACTIVATED
>>EXIT
```

DBUTIL checks to see that the file has a file code of zero, is an unnumbered, ASCII file, has a record length not greater than 128 characters, and contains at least three records. In order to edit the database-access file or prevent programs from referencing it, you must issue a **DEACTIVATE** command within the DBUTIL program.

One of the benefits of the database-access file method is that it restricts who may use the database and thereby enhances security. (The contents of the database-access file itself can be hidden from the user who simply runs the application program.) Under this method, however, you can access only one database in any one remote session. Under the other two methods, you can access more than one database by means of multiple **FILE** commands.

You can use QUERY, Hewlett-Packard's interactive database inquiry facility, to retrieve information from a remote database. You could run QUERY.PUB.SYS locally, specifying either the database name itself or the appropriate database-access file name. If you use the actual database name, you must have previously established a session on the remote node and issued a **FILE** command for the remote database.

It is more efficient to run QUERY directly in the remote environment. At the MPE/iX prompt, type the following commands (user input bold):

```
DSLINE LEO
REMOTE HELLO NSUSER.NSACCT
REMOTE RUN QUERY.PUB.SYS
>DATA-BASE=databasename>
```

For more information on TurboIMAGE and QUERY, see the *TurboIMAGE Data Base Management System Reference Manual*, especially the sections entitled “Using a Remote Database;” and also see the *QUERY/3000 Reference Manual*.

5 Network File Transfer

Network File Transfer (NFT) is the network service that copies disk files. The files may be copied to the same computer or from one computer in a network to another. You can use Network File Transfer to transfer a file between any two systems in an NS 3000/iX network, even if both of those systems are remote from your own. You can also use it for purely local transfers on a single HP 3000.

This chapter explains how to use the interactive command **DSCOPY** to interactively invoke NFT, and the use of intrinsics that enable NFT to be invoked from within a program. The first part of this chapter explains how to use NFT to interactively copy files. The second part, "Programmatic NFT", describes how to use intrinsics to invoke NFT from within a program.

Limitations

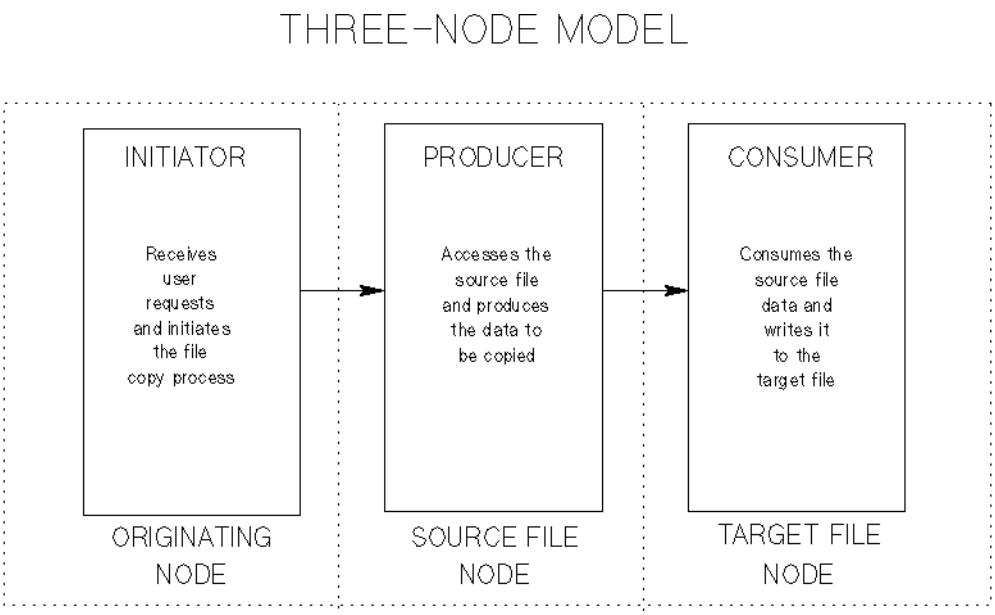
Following are NFT limitations:

1. NFT only works with filenames in the traditional MPE namespace. It does not work with filenames in the HFS (POSIX-compliant) namespace. NFT does not recognize HFS directories or filenames that contain slashes as directory delimiters. For example, it will not work properly with filenames such as "./FILEa," or "/usr/include/stdio.h".
2. NFT only works with traditional MPE record-oriented files; that is files with fixed (F), variable (V), or undefined (U) record types. It does not work with files that have the POSIX-compliant, bytestream (B) or directory (H) types.

Three-Node Model

NFT transfers files according to the model shown in Figure 5-1. There are three logical participants in the file transfer activity: initiator, producer, and consumer. This model is called the three-node model. According to the three-node model, the initiator, located on the system where the transfer originated, receives the request and initiates the transfer. The producer, located on the same node as the source file, accesses that file and "produces" the data that is to be transferred. The consumer, residing on the same node as the target file, "consumes" the data and writes it into the target file. All three participants are logically distinct. All three participants can be on separate nodes; the transfer request does not have to originate from either the source or the target node. It is also possible for any two or all three participants to reside on the same node.

Figure 5-1 Three-Node Model



This method, coupled with the ability to include a logon command string as part of the **DSCOPY** command, provides considerable flexibility. Because the initiation of the transfer request is independent of the producing and consuming functions, you don't have to explicitly log on to a remote source or target node. If you supply an appropriate logon sequence in the transfer request or in a prior **DSLINE** command, NFT will create a session on a remote source or target node if one does not exist already. If all the systems involved (as many as three) can establish NS-level connections with each other, you can transfer a file between any two of these nodes, and you can initiate the transfer from any of them.

File Copying Formats

NFT uses two file copying formats: Transparent Format and Interchange Format.

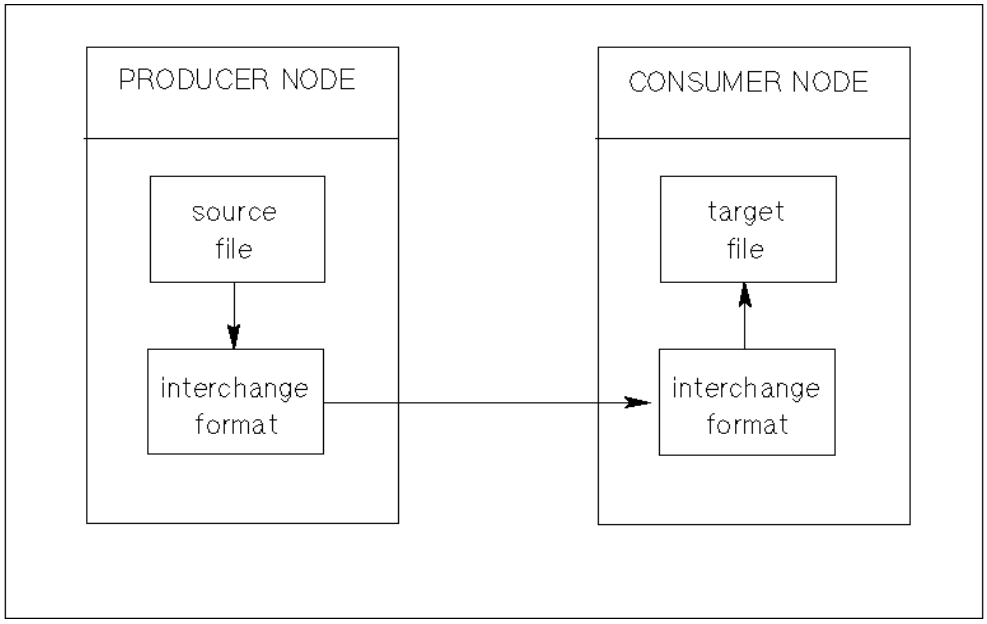
Transparent Format

When files are copied from a source file node that is the same type of computer as the target file node (for example, if they are both HP 3000s or HP 9000s), the files are copied using a format called Transparent Format. Transparent Format does not alter a file's attributes, but simply copies the file. It should be used when you want a low-overhead, maximum-speed file copy process between systems of the same type.

Interchange Format

When two computers are of different types (for example, one is an HP 9000 running a release prior to HP-UX 10.0 and one is an HP 3000), files copied from one to the other must be converted to Interchange Format (Figure 5-2). Interchange Format consists of a set of attributes that describe a file in a standard way so that it can be understood by any NS system. Interchange Format is invoked by default whenever you use NFT to copy a file residing on one type of system to a system of another type. You can also use a **DSCOPY** command option (INT) to explicitly specify that a file be converted to Interchange Format. In addition, several options automatically invoke Interchange Format. These options are described in the DSCOPY syntax description later in this chapter.

Figure 5-2 Interchange Format



When a file is copied using Interchange Format, it is translated into Interchange Format at the source system before it is copied to the target system. At the target system, it is mapped from Interchange Format into the target system's file format. Interchange Format's standard file attributes enable the target computer to map the source file into a target file with attributes that match the source file's as closely as possible.

You can use the options that invoke Interchange Format to give a target file a different set of attributes from those that characterized the source file from which it was copied, even if the files are being transferred between computer systems of the same type. For example, by copying a file composed of variable length records and using the `FIX` option, you can create a file containing the same information, but formatted into fixed-length records. Other options (described in detail later in this chapter) can be used to create duplicate files that differ from their source files in record size, length, type of data and other file characteristics.

Data Interpretation

Although the purpose of Interchange Format is to create an accessible target file on different kinds of systems, it does not ensure that the target file will be usable. This is because Interchange Format changes a file's attributes only; it does not perform data interpretation. Interchange Format can create an unusable target file if the target system has a different representation for the data present in the source file.

For example, if a file that contains floating point numbers is copied to a different kind of computer, there is no guarantee that the target node will be able to read the data as floating point. Consequently, the usability of your target files must be determined by the applications that use them.

DSCOPY

Transfers or copies a disc file from one node to another (or within a single node).

Syntax

DSCOPY

[*sourcefile* [*sfileloc*] [to [*targetfile*] [*tfileloc*][;*opt*]]]
[+[*sfileloc*] [to [*tfileloc*][;*opt*]. . .]
[+*opt*[;*opt*]...

Use

Available	In Session?	Yes
	In Job?	Yes
	In Break?	No
	Programmatically?	No
Breakable?		No
Capabilities?		None

Parameters

sourcefile The name of the file to be transferred, optionally including qualifiers.

Specify HP 3000 source files as follows:

filename[/*lockword*][.*groupname*.[*accountname*]]

The *sourcefile* may be a formal designator defined in a prior file equation. If the *sourcefile* is a formal designator defined in a prior file equation, then the *sourcefile* must be preceded by an asterisk.

The *sourcefile* parameter may also specify a generic file set, via MPE/iX "wildcard" characters "@", "#", or "?". See "Multiple Transfer" later in this chapter for more information on generic file sets.

sfileloc A node specification for the source file, in the form:

[*delim* [*location*]] [**[**[*logon*]]**]**

where *delim* is either a colon (:) or a comma (,).

The *location* parameter is either a node name or a previously defined environment ID. Logon must be a valid logon sequence for the node in question, including all necessary passwords. If specified, the logon is used to create a temporary remote session on the node. Note that brackets are required around the logon sequence.

Here are some syntactically correct examples of HP 3000 source file location specifications:

```
:ENV1
,NODEA
,NODEA [NSUSER/PASSWD.NSACCT]
:NODEA
,ENV1[NSUSER.NSACCT]
:
:ENV1 []
```

If *delim* and *location* are omitted, then the default is the global source file location specification or, if there is no global specification currently in effect, the local node name. (For an explanation of global specifications, see "Using Global Specifications" later in this chapter.) If *delim* appears without a *location*, the local node name is used whether or not there is a global specification.

If you specify an individual or global *logon*, it will be used to create a new session even if a session already exists on the node in question (in the specified remote environment). If the *logon* parameter and its surrounding brackets are omitted, the default is the global *logon* sequence. Or, if there is no global logon currently in effect, and there is no current remote session, the default is the logon sequence specified in a prior **DSLINE** command for this remote environment (in the LOGON= option). If you include the brackets, but omit the *logon*, then a global *logon* specification is ignored and the DSLINE logon specification (or existing session if there is one) is used. The order of priority (from high to low) is: logon specified here; global logon; existing session; DSLINE logon.

In short, if you want to use an existing session or a DSLINE logon, you should not include a logon in the transfer specification. You should also clear any global logon in effect or use empty brackets to cause the global logon to be ignored. If you want a new (temporary) session to be created for the transfer, regardless of whether a session already exists, you should include a valid logon in the transfer specification or in a global specification.

`to` Either the word TO or a semicolon (;). If *sfileloc* ends with a colon (:, not followed by a node name or environment ID), you must use the semicolon (;) form.

targetfile The name of the file into which the source file will be copied. For files being copied to an HP 3000 system, specify the target file as follows:

filename[/*lockword*][.*groupname*.[*accountname*]]

The *targetfile* may also be a formal file designator defined in a prior file equation. If the *targetfile* is referenced by a formal designator, the *targetfile* must be preceded by an asterisk.

A file equation can also redirect the *targetfile* to the temporary domain by specifying the disposition directive "`;TEMP`" in the file equation. The default disposition for the targetfile is the permanent domain.

NOTE If a file equation is used, it cannot reference the "`;DEL`" option in the disposition directive, or reference either the "`;OLD`", "`;OLDTEMP`", or "`;NEW`" file domain options in the file equation. Use of these options will result in an error.

A new HP 3000 target file is given a lockword if you supply one in the *targetfile* parameter. For more information on using lockwords, refer to the paragraph on lockwords in the section called "Using DSCOPY" later in this chapter.

If the source file uses special MPE/iX "wildcard" characters to specify a generic file set, then the *targetfile* parameter must be either omitted or included in the following form:

`@[.`*group*`[.`*account*`]]`

Refer to "Multiple Transfer" later in this chapter for details on using wildcard characters.

You can specify a KSAM file pair in the *targetfile* parameter by enclosing the pair of names (separated by a comma) in quotation marks.

Default: If the file name is omitted, the corresponding *sourcefile* name and lockword is assumed. If group and/or account names are omitted, corresponding portions of the target node session are assumed. If the specified target file name is the name of an already existing file, `DSCOPY` asks you if you want to purge the existing file. If you respond by typing `N`, `DSCOPY` prompts you for an alternative target file name.

tfileloc A node specification for the target file, with the same syntax and defaults as the *sfileloc*.

opt Any of the options described below can be used to transfer files between HP 3000s on the same LAN.

NOTE The descriptions of the options listed below assume that the transfers are between two HP 3000s (producer and consumer nodes) only. To learn what defaults, options, and syntax to use to transfer files between PCs and HP 3000s, refer to the *User Guide for HP PC Network Services*.

There is no limit to the number of options that you can specify for a single DSCOPY request. Each option must be separated by a semicolon. Some options conflict with each other. In Table 5-1, options grouped together conflict. If you specify conflicting options in the same invocation of the **DSCOPY** command, *only the last option specified will take precedence*. For example, if you specify the ASC and BIN options in the same file transfer request, only the option specified rightmost in the command line will take effect.

Table 5-1 Conflicting DSCOPY Options

APP OVER REP	ASC BIN
FIX VAR	SEQ DIR

APP (append) Appends the source file to an existing target file. An error condition will occur if there is not enough file space allocated to hold both the original and appended files, source file type attributes do not match target file attributes, or the specified target file does not exist. If not enough space exists to hold both files, as much of the source file as will fit will be appended to the target file before an error message is issued.

ASC (ASCII) Specifies that records contain ASCII characters and that ASCII spaces (octal 020000 000040) will be used as padding for fixed-length records. transfers files using Interchange Format. If the STRIP option is specified when the ASC option is in effect, DSCOPY will remove extra spaces from the ends of records.

If you do not specify this option, and if the HP 3000 source file is ASCII, then the HP 3000 target file will be ASCII.

`BIN` (binary)

Specifies that records contain binary information and that ASCII null characters (all zeros) will be used as padding when fixed-length records are created.

`BIN` transfers files using Interchange Format. `BIN` can be used with the `STRIP` option to remove null padding from the ends of records.

If you do not specify this option, and if the HP 3000 source file is binary, then the HP 3000 target file will be binary.

`CHECKPT=`
checkpt spec

`CHECKPT` (along with `RESTART`) is used to recover file transfers that have prematurely aborted due to a transient problem such as a link failure.

checkpt spec = [*checkpoint interval*]
[,[*restart ID file*]
[,[*restart record*]]]

The c*heckpoint interval* is the time in seconds between handshake sequences from producer to consumer and back. This is the maximum time that will be lost if a transfer failure occurs and a restart becomes necessary. The default *checkpoint interval* is 300 seconds (5 minutes).

The *restart ID file* is the file in your local group and account where the *restart ID* will be written by NFT. The file must have fixed-length records. If you specify a *restart ID file* that does not already exist, NFT will create it. The *restart ID* is an ASCII number that uniquely identifies the file transfer. It must be passed back to NFT with the `RESTART` keyword to initiate a restart. If you do not specify a *restart ID file*, then the *restart ID* will only be written to `$STDLIST` and will not be written to a file.

The *restart record* is the record in the *restart ID file* to which the *restart ID* will be written. Note that this record will be overwritten by NFT. The *restart ID* will be the first 1 or 2 characters in the record. The default is 0, the first record in the file.

`CLEAR` — Clears all global specifications previously issued in the current interactive `DSCOPY` session.

If you do not specify this option, global specifications will remain in effect until a conflicting option is specified globally or `CLEAR` is specified. A previously issued global specification can also be overridden by a conflicting option for a single file transfer.

`COMP` (compress) — Compresses contents of target file during transmission. Compression minimizes the space required to represent sequences of repeated characters. The data are decompressed before they are written to the target file. Note: `COMP` cannot be used for local transfers, that is, to a target file on the same node as the source file.

If you do not specify this option, NFT will check to see if `COMP` has been specified in a previous **`DSLINE`** command for the target environment. If not, copied files will not be compressed.

`DIR` (direct) — Specifies that the target file will be organized to allow direct access. Each record in the source file will have its logical record number sent along with it so that target file records can be accessed in the correct sequence. `DIR` causes files to be copied using Interchange Format.

NOTE Do not use `DIR` if the HP 3000 target file is to be variable (either because `VAR` is in effect or because the source file records are variable length and the `FIX` option is not in effect); this will cause an error to occur.

If you do not specify this option, and if the source file is a direct access file (such as an RIO file), then the target file will be a direct access file.

`FCODE=` *sourcefilecode*
: Gives the file code for the source file. In order to copy a privileged file (for example, a TurboIMAGE data base), you need to supply the appropriate negative file code in this option. The resulting target file will be given the same file code. If a privileged file is being copied, System Manager or Privileged Mode capability is required for the producer environment unless the logon used for the producer environment is the same as the logon used when the file was created.

`FIX` (fixed)
: Specifies that the target file will be composed of fixed-length records. Record size can be specified using the `RSIZE` option. If `RSIZE` is not specified, target file records are the same length as the maximum length record in the source file. If the source file is binary or the BIN option is in effect, variable length records that are less than the target record length will be padded with ASCII nulls (zeros). If the source file is ASCII or the `ASC` option is in effect, variable length records that are less than the target record length will be padded with ASCII spaces (octal `020000 000040`). `FIX` causes files to be copied using Interchange Format.

  If you do not specify this option, and if the source file contains fixed-length records, then the target files will contain fixed-length records.

`FSIZE=` *filesize*
: Specifies the amount of space, in records, to allocate for the target file. If the target file is to have variable length records, filesize is the number of maximum size records to allocate. (Record size or maximum record size

	can be specified using the RSIZE option.) FSIZE causes files to be copied using Interchange Format. If you do not specify this option, then the target file length will be the same as the source file length.
INT (Interchange)	Causes file or files to be copied using Interchange Format. Files are automatically copied using Interchange Format if the producer and consumer nodes are different kinds of computer systems or if the APP, ASC, BIN, DIR, FIX, FSIZE, RSIZE, SEQ, STRIP, or VAR options are specified. If you do not specify this option, files will be copied using Transparent Format.
MOVE	Causes the source file to be purged after a successful transfer. If you do not specify this option, source files remain intact at the source node.
OVER (overwrite)	Causes a copy of the source file to overwrite an existing target file, beginning with the first record. If the source file is larger than the existing target file, NFT will copy as much of the source file as will fit in the existing file space and will then return an error. If the source file is smaller than the target file, then the contents of the existing file that extend beyond the end of the copied source file will remain in the target file. If the target file does not exist, a new file will be created. The attributes of the source and target files must match, or DSCOPY will return an error message. If you do not specify this option, existing target files will not be overwritten.
QUIET	Suppresses all display output about file transfers except error messages.

If you do not specify this option, DSCOPY will display information about the success of the file transfer, such as source and target file names and target file lengths.

REP (replace)

Causes an existing target file to be purged and replaced by a copy of the source file. If the target file does not exist, a new file will be created.

If you do not specify this option, existing target files will not be replaced.

RESTART=
restart spec

When used after a prior CHECKPT option, RESTART initiates the restart of a transfer. You must be logged on to the same environment where you first specified CHECKPT. You can only use RESTART if both the source and target nodes are HP 3000s that support CHECKPT/RESTART. If this option is not specified, a restart will not be attempted.

Restart spec = *restart ID* or
restart ID file
[,[*restart record*]]

The *restart ID* is an ASCII number that uniquely identifies a checkpointed transfer. The *restart ID* is returned to you when you first specify CHECKPT.

The *restart ID file* is a file in your local group and account where the *restart ID* was written by NFT when you specified CHECKPT.

The *restart record* is the record in the *restart ID file* where the *restart ID* was written when you specified CHECKPT. The default is 0, the first record in the file.

RSIZE=
recordsize

Specifies the length of target file records (*recordsize*) in bytes. If the target file is to consist of fixed-length records, *recordsize* is the size of each record. If the target file is to consist of variable length records, *recordsize*

specifies the maximum size record allowed in the file. If DSCOPY must truncate records to adhere to the specified *recordsize*, it will issue a warning. RSIZE causes files to be copied using Interchange Format.

If you do not specify this option, then target-file records will be the same length as source-file records.

SDEV=*source_device*
: Names the disc device on the source node from which the source file should be obtained.

  If you do not specify this option, the disc device will be given the device class name DISC in the source system's device I/O configuration file.

SEQ (sequential)
: Causes the target file to be organized to allow sequential access. Records in the source file will be sent to the target node contiguously. SEQ causes files to be copied using Interchange Format.

  If you do not specify this option, and if the source file is a sequential access file, then the target file will be a sequential access file.

SHOW
: Displays global specifications currently in effect (not including specifications specified after SHOW on the same line).

STRIP
: Removes padding (extra bytes used to create records of all the same length) from the ends of records when creating the target file.

  You can use STRIP to create variable length records from fixed-length records. If the source file is a fixed ASCII file, spaces are removed. If the source file is a fixed binary file, nulls are removed. STRIP is invalid when used with a source file that has variable records. If you do not specify this option, then any padding that exists in the source file is copied as is to the target file.

TDEV=
target_device
: Names the disc device on the target node to which the target file should be written. You can specify alternate disc devices for a KSAM file pair in the TDEV= option by enclosing a pair of device names, separated by a comma, in quotation marks. If you do not specify this option, then the disc device will be given the device class name DISC in the target system's device I/O configuration file.

VAR (variable)
: Specifies that target file records will be of variable length. You can specify the maximum record size allowed in the target file with the RSIZE option. VAR causes files to be copied using Interchange Format.

NOTE Do not use VAR in conjunction with the DIR option if the target node is an HP 3000; this will cause an error to occur. If you do not specify this option, and the source file records are of variable length, then the target file records will be of variable length.

+opt
: Indicates that the following *opt* specifications are global. All specifications except file names may be made global. These remain in effect until a new, conflicting global specification is issued or the CLEAR option is used. Individual, non-global specifications override global specs for one transfer only. Global specifications are cleared when the DSCOPY subsystem terminates.

NOTE If you specify an invalid or unsupported keyword as a global option, NFT ignores any options that follow.

Summary of DSCOPY Options

Table 5-2 summarizes the available DSCOPY options. Refer to the syntax description earlier in this chapter for a full description of the effects of each option listed.

Table 5-2 DSCOPY Options Summary

Option Name	Description
APP *	Appends source file to existing file specified as target file.
ASC *	Causes target file to contain ASCII data.
BIN *	Causes target file to contain binary data.
CHECKPT = *checkpoint spec*	Used (along with RESTART to recover file transfers that have aborted.
CLEAR	Removes previously specified global specifications.
COMP	Compresses contents of target file during transmission.
DIR *	Causes target file to be a direct access file.
FCODE = *sourcefilecode*	Specifies file code needed to open source file; gives same file code to target file.
FIX *	Causes target file to contain fixed length records.
FSIZE = *filesize**	Specifies size of target file in records.
INT *	Causes files to be copied using Interchange Format.
MOVE	Causes source file to be purged after file transfer.
OVER	Causes copied file to overwrite file that is specified as target file.
QUIET	Suppresses all display output except error messages.
REP	Causes source file to replace specified target file. Previously existing file of same name as target file is purged.
RESTART = *restartspec*	Initiates the restart of a transfer that was previously checkpointed.
RSIZE = *recordsize* *	Specifies length (in bytes) of target file records.
SDEV = *source_device*	Specifies disc device on which source file resides.
SEQ *	Causes target to be organized to allow sequential access.
SHOW	Displays global specifications in effect.

Option Name	Description
`STRIP *`	Removes padding from records in target file.
`TDEV =` *target_device*	Specifies disc device to which target file will be written.
`VAR *`	Causes target file to be composed of variable length records

Note: * indicates option invokes Interchange Format

Using DSCOPY

The following notes describe the function of DSCOPY and explain its operation in special situations:

Required Access. Read and lock access is required for any file you want to copy with DSCOPY. If you do not have both read and lock access to the file, DSCOPY will issue a file security violation error message.

Interactive Use. When you enter a **DSCOPY** command, NFT becomes interactive and displays the prompt DSCOPY. Once the prompt appears, you can issue additional file transfer requests by specifying only source and target filenames and additional options; there is no need to exit the DSCOPY subsystem before requesting additional file transfers.

File Specification Defaults. At NS 3000/iX nodes, the logon and file name specifications are identical to those you usually use at your HP 3000. If a group name is included in the source file specification and the account name is omitted, NFT searches for the group in the logon account. If only a file name is specified (that is, group and account names are omitted) NFT uses the logon group and account.

Special Characters in Logons, File Names or Node Names. If a file name, node name, or logon contains characters that have special significance to either NFT or the source or target node operating systems, you can enclose the name or logon string in quotation marks (“ ”) so that the name or logon is accepted.

Continuation Lines. If a **DSCOPY** command is too long to fit on a single line, you can continue to type; DSCOPY will automatically continue the command on the following line. You can also type an ampersand (&) followed by **[Return]** at the end of a line; this causes the prompt `continue` to appear, after which you can type the rest of the command.

On-Line HELP. To obtain an on-line description of the **DSCOPY** command, type ? to the right of the prompt DSCOPY.

Terminating DSCOPY. To terminate the DSCOPY subsystem, type either (1) `//`**[Return]** or (2) **[CTRL]-Y** when a file transfer is not in progress. (Typing **[CTRL]-Y** when a file transfer is in progress interrupts the file transfer, but does not terminate the subsystem unless the transfer requests originate from a command file.) From a job stream (batch job), the characters `//` must be used to terminate DSCOPY.

Interrupting a File Transfer. To interrupt a file transfer that is in progress, type **[CTRL]-Y**. Refer to “Interrupting a File Transfer” later in this chapter for details.

Lockwords. If the sourcefile has a lockword, and the target file is implied, then the targetfile will get the same lockword as the sourcefile. If a targetfile already exists, and you want to give it a new lockword, you must specify the targetfile and its lockword.

Message Files. MPE/iX message files must be transferred in Transparent Format. They will not be copied if Interchange Format is used; thus they can be copied to HP 3000 systems only.

KSAM Files. KSAM file pairs (data files and corresponding key files) can be transferred together as pairs using Transparent Format. Files must be transferred one at a time if Interchange Format is used. (Note: Because Interchange Format causes file characteristics to be lost, KSAM files might be unusable if copied using Interchange Mode.) Using Transparent Format, KSAM file pairs can be specified as source or target files in two ways, as follows:

- Specify only the name of the data file. In this case, NFT will create both data and target key files; the key file copy will be the same name as the data file copy with the letter "K" appended to it.
- Specify both data and key files, using the following syntax:

  "*datafilename,keyfilename*"

The names of the data and key files must be enclosed in quotes.

If either of the target file names specified in a KSAM file transfer already exists, DSCOPY will report an error. To avoid this, you can use the REP option so that the existing file is replaced by the contents of the source file. However, specifying REP will cause to purge the existing KSAM file on the target node before the new file replaces it. If a KSAM file (either data or key file) were open when a system failure occurred, attempts to copy the file with DSCOPY will normally be unsuccessful. However, if the same file or files is copied in a generic transfer (using wildcard characters), no error message will appear, but the target file will be unusable. Refer to the *KSAM/3000 Reference Manual* for recovery procedures that can be used to make the KSAM files (both source and target) usable.

Requesting Transfers from Files. DSCOPY requests can be issued from command files as well as being issued directly from the keyboard. A command file is an unnumbered text file that contains one or more file transfer requests. Each request in the file must consist of source and target file specifications as well as any desired options. Do not include the word "DSCOPY" in the file.

To issue a DSCOPY request from within a file, you must first issue a file equation equating the formal designator DSCOPYI to the file containing the request.

For example, to have NFT read file transfer requests from the file FILE1, at the MPE/iX prompt, enter:

```
FILE DSCOPYI=FILE1
```

followed by

```
DSCOPY
```

Transfer requests are normally read from the file DSCOPYI, which by default is set to $STDIN(X)— the user's terminal in an interactive session.

NOTE

If any global specification is specified in the DSCOPY command line, DSCOPY will ignore the file equation for DSCOPYI and subsequently enter the DSCOPY subsystem.

Variable Length Records. If a file containing variable length records is copied to an HP 3000 using Interchange Format, the space allocated for the file will be 4 bytes less than the length of the source file rounded up to the nearest multiple of 256 bytes. On an HP 3000, direct access of variable length record files is not allowed. As a result, DSCOPY will return an error if you specify the DIR and VAR options in the same command or if the source file has variable length records and the DIR option is specified but the FIX option is not in effect.

RIO (Relative I/O) Files. DSCOPY will copy RIO files as direct files automatically; you do not need to specify the DIR option to enable direct access to the resulting target file. However, an RIO file will retain its RIO characteristic only if copied to another HP 3000.

Entering MPE/iX Commands. You can enter MPE/iX commands after the DSCOPY prompt by typing a colon (:) followed by the command and **[Return]**.

Job Streams. If the **DSCOPY** command is used in a job stream (batch job), other MPE/iX commands must not be inserted in the job stream between the **DSCOPY** command and the // that terminates the DSCOPY subsystem.

Multiple Transfer

Using special "wildcard" characters, you can tell NFT to transfer a generic set of HP 3000 files to another HP 3000. For MPE files (on the HP 3000) these wildcard characters are the same ones used within the MPE/iX file system:

@ — stands for zero or more alphanumeric characters;

— stands for one numeric character;

? — stands for one alphanumeric character.

When used with DSCOPY, wildcard characters can be used to specify HP 3000 *file names only*; they *cannot* be used to specify group or account names. The characters # and ? can be used to specify source file names only. The character @ can be used to specify both source and target file names, but can be used only once, with no other characters surrounding it, to indicate a set of target file names. If wildcards are used to transfer more than one file, and the destination file group and account is explicitly specified, the @ character must be used to specify target file names.

For example, the source file designation E@.PUB.SYS can be used to copy all files in the PUB group of the SYS account whose names begin with E. The source files will be copied to corresponding target files having the same file names as the source files, in the logon group and account. To transfer the same files to a group or account other than the logon group and account, use the @ character to specify the destination file set, as follows: *@.group.account*. For example, if while logged on to another account you decided to copy the files designated by E@.PUB.SYS to the TST group of the IND account, you could specify the destination file set as @.TST.IND. The resulting target files would have the same names as their corresponding source files but would be located in the TST group of the IND account. When a generic file set is copied, the producer and consumer "negotiate" the transfer of each file. Intermediate results are reported after each transfer. If an error occurs during one of these transfers, an error message is reported.

In an individual file transfer, if you name a target file that already exists, and you do not specify the replace or overwrite option, you will be prompted for further action. (For information on the replace and overwrite options, see the parameter explanations for the **DSCOPY** command.) In the case of a generic file transfer, however, you will not be prompted. Instead, the transfer attempt will produce an error, and the DSCOPY subsystem will attempt to transfer the next file in the set.

Using Global Specifications

Global specifications, indicated by a + preceding the specification, take effect for all subsequent transfers unless one of the following conditions occurs:

1. A new global specification that conflicts with the old one is given. For example, if REP is specified globally, it will override and cancel a prior OVER global specification currently in effect.
2. An item given in an individual, non-global transfer specification conflicts with a previous global specification. This item will override the global specification for this one transfer only.
3. The CLEAR option is used. This will clear all global specifications currently in effect (not including further specifications on the same line after the CLEAR).

4. The DSCOPY subsystem is terminated.

All specifications except file names may be made global in this manner. A new SDEV or TDEV specification, or "SDEV=" or "TDEV=" without a device name, clears a previous global source or target device. The only way to clear MOVE, COMP, or QUIET is to use CLEAR.

If all source and target parameters are omitted, or if the command begins with + (global), you will receive a subsystem prompt consisting of the string DSCOPY. (You can also issue global specifications within the subsystem.)

Interrupting a File Transfer

[BREAK] is disabled during a DSCOPY operation. To interrupt a file transfer in progress, type **[CTRL]-Y** instead.

The DSCOPY subsystem issues a prompt consisting of the word DSCOPY. If you press **[CTRL]-Y** (or **//[Return]**) in response to this prompt, the subsystem is terminated. If you enter **[CTRL]-Y** in response to a continuation prompt (issued when an ampersand on the previous line allows a command to be continued on the next line), the current command is not executed and the DSCOPY prompt is reissued. If **[CTRL]-Y** interrupts a file transfer, you are prompted for further instructions. The following commands are allowed:

1. **A[BORT]** — stops transfer and saves as permanent files new files created during the transfer. Files in the process of transferring that have been partially copied are saved as new target files.
2. **C[ANCEL]** — stops transfer. Target files that have already been completely copied during the transfer are saved on the target system. Files whose transfer was incomplete are not saved on the target system.
3. **P[ROGRESS]** — causes progress of transfer to be reported.
4. ? — gives a description of the commands listed above.

If an error or a CANCEL request interrupts a transfer, a newly created file will be purged unless the transfer is complete. If a generic file set is being transferred, those files that have been successfully copied are not purged. In the case of a KSAM file (actually two files, a data file and a key file), the new data file is purged unless its transfer is complete; the new key file is not purged even if its transfer is not complete. If an incomplete key file is created in this way, you should purge it or overwrite it in a subsequent transfer.

Event Recording

DSCOPY produces a listing of user requests and file transfer results (including error messages). This information is sent to a primary file and a secondary file, either of which (or both) may be disabled. The

primary file is $STDLIST (the terminal in the case of sessions, the system line printer in the case of streamed jobs). The secondary file is a file or device with the formal designator DSCOPYL.

The QUIET option suppresses all information regarding the success of file transfers except error messages. Primary output to $STDLIST is disabled if the *opt* parameter of the DSCOPY intrinsic is set to 0,1, or 2; otherwise, primary output is enabled. Secondary output is normally disabled since the secondary file DSCOPYL defaults to $NULL. It will be enabled if a file equation names DSCOPYL as the formal designator of an actual file or device. For example, assume the file is named OUTFILE. Type the following at the MPE/iX prompt:

```
FILE DSCOPYL=OUTFILE
```

If the file does not already exist, you can indicate TEMP or SAVE as the disposition of the file. Otherwise the file will be purged.

```
FILE DSCOPYL=OUTFILE,new;TEMP
```

In the second case NFT will create the file for you, but you must SAVE it to make it permanent.

The NFT facility also sets a number of Job Control Words (JCWs). JCWs are 16-bit values, identified by a name, which are maintained by MPE on a per-job or per-session basis. The JCW named DSCOPY indicates how many files were successfully transferred in the DSCOPY subsystem.

The JCW named NFTERR gives the NFT error code returned after an unsuccessful transfer in an NS 3000/iX network. If a warning has been issued, the “warn” bit of this JCW is set. Bit 2 of this JCW indicates which NS 3000 NFT error message set the error code belongs to: if on, it is the HP 3000-specific error message set; if off, it is the generic NFT error message set.

If a transfer error occurs in a job stream, DSCOPY continues with the next transfer request. However, the abort bit of the system JCW, named JCW, is set. As a result, the job will fail after the DSCOPY subsystem is exited unless a **CONTINUE** command has been specified.

Using Checkpoint and Restart with DSCOPY

If you specify CHECKPT in the **DSCOPY** command line, the file transfer will occur normally, but an additional handshake sequence will occur between the source and target HP 3000 systems at periodic intervals (optionally specified by you). If a failure occurs, it is then possible to restart the transfer from the point where the last handshake took place. A *restart ID* is a number that uniquely identifies a transfer. The *restart ID* is returned to you before a file transfer actually begins. For example, if a transfer were 44% complete when a handshake occurred, and the link were to fail at some time before the next handshake, the transfer could be restarted at a later time with at least 44% of the transfer already complete.

To restart an aborted transfer, specify the keyword RESTART in the **DSCOPY** command line along with the same **restart ID** that was returned to you when **CHECKPT** was specified. You *must* be logged on to the same local environment where CHECKPT was specified. NFT will then attempt to restart the transfer from the point where the last handshake sequence took place. The restarted transfer will continue to be checkpointed and may be restarted in the event of a subsequent failure.

In order to use CHECKPT/RESTART, the producer and consumer environments cannot be the same; that is, checkpointing will not be done for local transfers.

When you restart a transfer by using the RESTART option, you can also invoke the CHECKPT option in order to modify the *checkpoint interval*; all other options will be ignored.

NOTE You can access and change source and target files between checkpointing and restarting. If you change the source file, you might not get the exact file you want on the target side. In this case, you should probably restart the entire transfer again.

New Restart Files Created During Checkpointed Transfer

In order to store restart information that will survive a system failure, NFT creates files called restart files. One file is created in the group and account of each role being played by NFT, that is, initiator, producer, and consumer. If more than one role is being played by a single environment, the restart file will be shared. The name of the restart file will be NFTRxx, where xx is a number from 1 to 99. These files will be purged upon successful completion of the file transfer. If an

intermediate failure occurs, the files will not be purged. Therefore, transfers that fail and are not restarted to a successful completion will leave restart files unpurged; NFT will not purge unused restart files.

Similarly, for generic transfers a permanent file called GENSETx, where x is a number from 0 to 9, is created to hold the list of files to be transferred on the producer node. This file is purged when the generic transfer is complete. Likewise, a file called NFTSCRxx, where xx is a number from 0 to 99, is created on the consumer node when the REP option is specified. This file is used as a scratch file to hold the target file during the transfer. When the transfer is complete, the old target file is purged and the scratch file is renamed to the target file name. Again, if an intermediate failure occurs, these files will not be purged. If the transfer is not restarted to a successful completion, these files will remain unpurged; NFT will not purge unused generic or scratch files.

Using the DSCOPYI File for Checkpointing

Transfers initiated using the **DSCOPYI** command file can also use CHECKPT/RESTART. If a failure occurs during any of the transfers in the list, that transfer can be restarted in the normal way. When that transfer is complete, the next transfer in the list will take place as if no failure had ever occurred. When checkpointing from a DSCOPYI file note the following conditions. First, a separate *restart ID* will be returned for each transfer. Second, in order to restart a transfer from DSCOPYI, the same file equation that was specified for checkpointing must be in effect when the restart is attempted.

NOTE The RESTART keyword cannot be used from within a DSCOPYI file.

Using CHECKPT and RESTART in Shared Environments

Checkpointing is allowed when the producer or consumer environments were created using REMOTE HELLO. If a restart is necessary, however, NFT will always attempt a programmatic logon to the producer or consumer nodes. In other words, NFT will set up its own, temporary environment, equivalent to your using the square brackets to specify the remote logon. If a password is needed to access one or both of these environments, it will not be available to NFT, and the restart will fail. The only way to ensure that this problem will not occur is to use the programmatic logon feature of NFT when CHECKPT is specified. The password(s) will then be available to NFT, so that a subsequent restart will be able to logon to the remote node(s).

Files Not Allowed with CHECKPT and RESTART

CHECKPT/RESTART is not allowed with message or circular files. It is also not allowed with files of variable length records in interchange mode.

Troubleshooting After Using CHECKPT and RESTART

If a restart returns an error, some possible explanations might be:

1. One or more of the necessary files for restarting has been lost or corrupted, that is, NFTRxx, GENSETx, or NFTSCRxx, or the source or target files.
2. The transfer did not progress past the negotiation stage before it was aborted.
3. The error was not one from which a restart can be done; examples include a file lockword violation or an unknown node name. A restart can only be done if the transfer has progressed past the negotiation stage to the data transfer stage.
4. The circumstances that caused the failure have not cleared; that is, the remote system is still down, or the link has not yet been reestablished.
5. You are not using the same local logon as was used when checkpointing was specified.
6. The file equation for DSCOPYI is not the same as it was when checkpointing was specified, or the command file has been purged or corrupted.

HP 3000 to HP 3000 Copying Examples

Following are examples of how to use the **DSCOPY** command for copying files between HP 3000s.

Local to Local

DSCOPY can be used (from the MPE/iX prompt) to make a local copy of a local file. If no global location specifications are in effect, the following names will be interpreted as local files:

```
DSCOPY SFILE TO TFILE
```

The node name delimiter (colon or comma) used alone will override any global specification and indicate that the file is local. In this example a semicolon replaces TO, since TO would be misinterpreted as the source environment ID following the colon.

```
DSCOPY SFILE:;TFILE
```

The following (still local) example copies a file named INFO in the PUB group of the MKTG account into a (new) file of the same name in the user's logon group and account:

```
DSCOPY INFO.PUB.MKTG TO INFO
```

Remote to Local

In the next example we assume that a remote session has been established on REMNODE and that no global *tfileloc* specification is in effect. This command, typed at the MPE/iX prompt, requests a local copy of a remote file:

```
DSCOPY FILEA:REMNODE TO FILEB
```

Local to Remote

If a remote session has not already been established, and if there is no global or DSLINE logon for the remote environment, you must include a logon sequence in the transfer specification. The following is a local-to-remote transfer (typed from the MPE/iX prompt):

```
DSCOPY FILEY TO FILEZ:REMNODE[REMUSER.REMACCT]
```

Remote to Remote

In the next examples we'll assume that remote sessions have already been established. From your local system you can copy one remote file to another on the same remote node. At the MPE/iX prompt, type:

```
DSCOPY FILE17:REMNODE TO FILE18:REMNODE
```

You can also copy a file from one remote system to another:

```
DSCOPY FILE1:REMNODEA TO FILE2:REMNODEB
```

Multiple Transfer

This example shows how to use the @ character to copy a set of files. All files in the PUB group of the MKTG account whose last four characters are BACK are copied to the AAA group of the ENG account. The target files will have the same file names as the source files. At the MPE/iX prompt, type:

```
DSCOPY @BACK.PUB.MKTG TO @.AAA.ENG
```

Global Specifications

Finally, you can establish global transfer specifications by putting a + before the specification sequence. If you include the global specifications in the **DSCOPY** command line, you will be placed in the subsystem, from which you can issue further commands. For example, at the MPE/iX prompt type the command as follows (user input is bold for clarity):

```
DSCOPY + :REMNODEB TO :REMNODEA; MOVE; COMP
DSCOPY THISFILE TO THATFILE
```

After the first **DSCOPY** command establishes global specifications, the command at the subsystem prompt moves THISFILE on REMNODEB to THATFILE on REMNODEA, purging the original file. The data are compressed during the transfer. Assume that remote sessions have already been established.

CHECKPT and RESTART Examples

Following are examples of how to use CHECKPT and RESTART:

1. The following example shows checkpointing being initiated for an IMAGE dataset file. The checkpoint interval is the default (5 minutes). The *restart ID* will not be written to a file, but will be written to $STDLIST if QUIET has not been specified or if output has not been disabled.

```
:DSCOPYSFILE;TFILE:REMNODE[REMUSER.REMACCT];
CHECKPT=;FCODE=-401
```

2. The following example shows checkpointing being initiated for the remote producer case. You have specified a new checkpoint interval (60 seconds), a *restart ID file* (IDFILE), and a record in that file where the *restart ID* will be written (12)

```
DSCOPYSFILE:REMNODE[REMUSER.REMACCT];TFILE; CHECKPT=60,IDFILE,12
```

3. This example shows checkpointing being initiated in a generic transfer. You have specified a new checkpoint interval (600 seconds) and a *restart ID file* (IDFILE). The *restart ID* will be written to the first record in the file, by default.

```
:DSCOPY@.PUB;@:REMNODE[REMUSER.REMACCT];CHECKPT= 600,IDFILE
```

4. This example shows a restart being specified as a global option. All other global options are ignored. The *restart ID* is 4.

```
:DSCOPY+RESTART=4
```

5. This example shows a restart being specified as a non-global option. The *restart ID* is 2. Checkpointing is also specified in order to change the previous checkpoint interval to 100 seconds.:

```
:DSCOPY;;CHECKPT=100;RESTART=2
```

6. This example shows the restart option with the *restart ID file* specified. The *restart ID* will be obtained from the first record of this file.

```
:DSCOPY;;RESTART=IDFILE
```

Programmatic NFT

The following subsections describe intrinsics that can be used to perform file transfers from within programs. Two intrinsics are available: DSCOPY, which performs the same function as the interactive **DSCOPY** command, and DSCOPYMSG, which writes a message indicating the outcome of the transfer request.

DSCOPY Intrinsic

Transfers or copies a file from one node to another (or within a single node).

Syntax

DSCOPY (*opt*,*spec*,***result***)

Parameters

opt (input) 16-bit integer, by reference. Enables/disables the primary output (to $STDLIST) and determines whether to continue after first transfer failure. The following bits of the opt parameter are significant (all others are reserved):

bit 15	Used to determine whether DSCOPY should terminate after the first transfer failure during a multiple transfer. 0 = attempt all transfers, even after a failure 1 = terminate DSCOPY after first failure
bit 14	Allows addition of a command file name in *spec* array for multiple file transfer. 0 = disables use of command file 1 = enables use of command file
bit 13	Enables/disables primary output. 0 = primary output disabled 1 = primary output enabled

<+>Recommended values:

0	All transfers attempted. Primary output disabled.
1	DSCOPY terminates after first failure. Primary output disabled.
2	All transfers attempted. Use of command file spec enabled. Primary output disabled.

3	DSCOPY terminates after first failure. Use of command file spec enabled. Primary output disabled.
4	All transfers attempted. Use of command file spec disabled. Primary output enabled.
5	DSCOPY terminates after first failure. Use of command file spec disabled. Primary output enabled.
6	All transfers attempted. Use of command file spec enabled. Primary output enabled.
7	DSCOPY terminates after first failure. Use of command file spec enabled. Primary output enabled.

HP recommends that you use the aforementioned values. Note that although other values can be used, they must be between 0 and 14 (decimal), inclusive.

spec (input) **Logical array, by reference.** In the case of a single or generic transfer request, this parameter should contain the DSCOPY transfer specification in the same syntax as the **DSCOPY** command parameters. The text should be ASCII characters terminated by a one-byte binary zero (that is, the ASCII null character). If this parameter contains the terminating zero (null character) only, the copy request(s) will be read from a file with the formal designator DSCOPYI (whose default is $STDIN, the session terminal). This is a way, in addition to using "wildcard" characters, of specifying multiple transfer requests. If DSCOPYI represents an actual file, it must be unnumbered and its lines must not end with nulls (zeros).

The spec data type differs slightly from language to language. See "Programming Language Considerations" below for data type definitions of specific languages.

result (output) **Two-element array of 16-bit integers, by reference**. Indicates the outcome of the intrinsic call. The first word of the array indicates whether or not the transfer was successful. A zero value signifies success; a nonzero value indicates an NFT error. If the number is positive, indicating an unsuccessful transfer over an NS 3000/iX link, bit 2 (where bit 0 is the high-order bit) indicates which NS 3000 error set the error belongs to: the HP 3000-specific error set (on) or the generic NFT

error set (off). The lower-order bits give the actual NFT error number in one or the other error set. Thus there are three NFT error sets. The result parameter containing these error numbers is interpreted correctly by the DSCOPYMSG intrinsic. Refer to the *NS 3000/iX Error Messages Reference Manual* for these error messages.

The second word of the array represents the number of files that were successfully copied.

Description

The DSCOPY intrinsic copies one file into another, performing exactly the same operations that the **DSCOPY** command performs. The source and target files do not have to be on the same node, and the program that calls the intrinsic does not have to be located on the same node as either of the files.

The *opt* parameter determines: (1) whether or not primary output is enabled, (2) whether to return after all transfers in a series have been attempted or after the first unsuccessful transfer, and 3) whether the file transfer will take place from DSCOPYI or from a designated command file.

If a single or generic transaction is involved, the *spec* parameter can contain the full text of the transfer specification, including all parameters and options, terminated by an ASCII null character. A null character (numeric zero) alone indicates that the transfer requests are to be read from the DSCOPYI file.

The returned *result* parameter indicates whether or not the transfer was successful. (All the parameters are required; DSCOPY is not option-variable.)

This intrinsic does not return condition codes. Split stack calls are not allowed.

NOTE

BREAK is disabled during a DSCOPY intrinsic operation. After DSCOPY completes, BREAK is re-enabled. If BREAK is programmatically disabled before the DSCOPY call, you must programmatically disable the BREAK again after the DSCOPY completes.

Programmatic examples may be found at the end of this chapter.

DSCOPYMSG intrinsic

Writes a message that corresponds to the result code returned by a DSCOPY intrinsic call.

Syntax

```
DSCOPYMSG    (result,fnum,r)
```

Parameters

result (input) **Two-element array of 16-bit integers, by reference.** The result is returned by the DSCOPY intrinsic. If the value in the first word is zero, then the file transfer was successful. Otherwise, the value indicates the error that occurred. (See the explanation of the *result* parameter in the DSCOPY intrinsic.)

fnum (input) **16-bit integer, by reference.** A file number indicating where the message associated with *result* should be written. If the value is zero, the message is printed on $STDLIST. If *fnum* contains a file number, the message is written to this file.

r (output) **16-bit integer, by reference.** The result returned by this DSCOPYMSG call. If this value is zero, the call was successful. Otherwise, the value indicates the error that occurred.

Description

The DSCOPYMSG intrinsic writes a message that corresponds to the result code returned by a DSCOPY intrinsic call. All the parameters are required; DSCOPYMSG is not option-variable. This intrinsic does not return condition codes. Split stack calls are not allowed.

Programming Language Considerations

The DSCOPY and DSCOPYMSG intrinsics are SPL procedures that may be called by programs written in other languages. Following are appropriate data types and calling sequences for the different languages available. (Other data types are sometimes possible.)

SPL

In SPL, *opt*, *fnum*, and *r* may be integers; *spec* must be a logical array; and *result* may be a logical array. The calling sequences are:

```
DSCOPY (OPT, SPEC, RESULT);
DSCOPYMSG (RESULT, FNUM, R);<D>
```

COBOL

In COBOL, *opt*, *fnum*, and *r* may be numeric data items; *spec* may be an alphanumeric data item; and *result* may be a numeric array. The calling sequences are:

```
CALL INTRINSIC "DSCOPY" USING OPT, SPEC, RESULT.
CALL INTRINSIC "DSCOPYMSG" USING RESULT, FNUM, R.
```

FORTRAN

In FORTRAN, *opt*, *fnum*, and *r* may be 16-bit integers; *spec* may be a character array; and *result* may be an array of 16-bit integers. The calling sequences are:

```
CALL DSCOPY (OPT, SPEC, RESULT)
CALL DSCOPYMSG (RESULT, FNUM, R)
```

BASIC

In BASIC, the intrinsics have a different name. In addition, only certain kinds of parameter names are permitted, as illustrated in the following calling sequences:

```
CALL BDSCOPY (O, S$, R)
CALL BDSCOPYMSG (R, F, R0)
```

Here O, F, and R0 may be integers; S$ is a string; and R may be an array of integers.

Pascal

In Pascal, *opt*, *fnum*, and *r* may be 16-bit integers; *spec* may be a packed array of characters or a string (a legal type in HP Standard Pascal); and *result* may be an array of 16-bit integers. The calling sequences are:

```
DSCOPY (OPT, SPEC, RESULT);
DSCOPYMSG (RESULT, FNUM, R);
```

NOTE In Pascal, if the spec parameter is represented as a character array or string, the numeric zero which terminates it should be represented by the ASCII null character. If spec is a mixed-type structure, the zero can be a numeric (one-byte) zero.

Programmatic NFT Examples

The following programs, in COBOL and Pascal, illustrate single and multiple file transfers via the DSCOPY intrinsic. They also call the DSCOPYMSG intrinsic to print an error message if necessary.

The multiple-file-transfer examples use transfer specifications that are read from a file with the formal designator DSCOPYI.

In the COBOL version of the multiple-file transfer, we assume that this file is the default $STDIN, namely the user's terminal. A second and alternative way of doing the COBOL multiple-file transfer would be to create an actual unnumbered file ("copyfile") that contains **DSCOPY** commands (for instance, SFILEA TO TFILEA). You would then have to create a file equation that equates DSCOPYI with the copyfile you have created.

COBOL: Single Transfer

In this application, the *opt* parameter is set to zero (0). All transfers will be attempted. Primary output is disabled. The command file spec for multiple transfers cannot be used. The *spec* parameter contains the full text of the transfer specification, including all parameters and options, and is terminated by an ASCII null character.

```
001000$CONTROL USLINIT
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. SINGLETRANSFER.
001300 REMARKS. THIS PROGRAM TRANSFERS A FILE TO A REMOTE NODE;
001400          IT CALLS THE DSCOPY AND DSCOPYMSG INTRINSICS.
001500 ENVIRONMENT DIVISION.
001600 CONFIGURATION SECTION.
001700 SOURCE-COMPUTER. HP3000
001800 OBJECT-COMPUTER. HP3000
001900 DATA DIVISION.
002000 WORKING-STORAGE SECTION.
002100 01     OPT PIC S9(4) COMP VALUE 0.
002200 01     SPEC.
002300 02     ASCIIPART PIC X(40) VALUE
002400        "NFTTEST TO NFTTARG:SOMENODE[NSUSER.NSACCT]".
002500 02     TERMINATOR PIC S9(4) COMP VALUE 0.
002600 01     RESULT.
002700 02     RESULTS PIC S9(4) COMP OCCURS 2 TIMES.
002800 01     FNUM PIC S9(4) COMP VALUE 0.
002900 01     R PIC S9(4) COMP VALUE 0.
```

```
003000 PROCEDURE DIVISION.
003100 BEGIN.
003200 CALL "DSCOPY" USING OPT, SPEC, RESULT.
003300 IF RESULTS(1) > 0 CALL "DSCOPYMSG" USING RESULT, FNUM, R.}
003400 STOP RUN.
```

COBOL: Multiple Transfer

In this application, the *opt* parameter is set to one (1). DSCOPY terminates after first failure. Primary output is disabled. The command file spec for multiple transfers cannot be used. The *spec* parameter contains a null character (numeric zero) indicating that transfer requests are to be read from the DSCOPYI file. The "COPYFILE" must already exist. You must issue the file equation "FILE DSCOPYI=COPYFILE" prior to execution of the program.

```
001000$CONTROL USLINIT
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. MULTTRANSFER.
001300 REMARKS. THIS PROGRAM ACCEPTS INTERACTIVE TRANSFER REQUESTS;
001400 IT CALLS THE DSCOPY AND DSCOPYMSG INTRINSICS.
001500 ENVIRONMENT DIVISION.
001600 CONFIGURATION SECTION.
001700 SOURCE-COMPUTER. HP3000
001800 OBJECT-COMPUTER. HP3000
001900 DATA DIVISION.
002000 WORKING-STORAGE SECTION.
002100 01     OPT PIC S9(4) COMP VALUE 1.
002200 01     SPEC.
002300 02     TERMINATOR PIC S9(4) COMP VALUE 0.
002400 01     RESULT.
002500 02     RESULTS PIC S9(4) COMP OCCURS 2 TIMES.
002600 01     FNUM PIC S9(4) COMP VALUE 0.
002700 01     R PIC S9(4) COMP VALUE 0.
002800 PROCEDURE DIVISION.
002900 BEGIN.
003000     CALL "DSCOPY" USING OPT, SPEC, RESULT.
003100     IF RESULTS(1) > 0 CALL "DSCOPYMSG" USING RESULT, FNUM, R.
003200     STOP RUN.
```

Pascal: Single Transfer

In this application, the *opt* parameter is set to four (4). All transfers will be attempted. Primary output is enabled. The command file spec for multiple transfers cannot be used. The *spec* parameter contains the full text of the transfer specification, including all parameters and options, and is terminated by an ASCII null character.

```
$standard_level 'hp3000', uslinit$
program pcopy (input,output);
type
          small_int = -32768..32767;
const
          null = chr(0); {ASCII null char}
@COMPUTERTXT = var
          opt      : small_int;
          fnum     : small_int;
          r        : small_int;
          spec     : string [80];
          result   : array [1..2] of small_int;
procedure DSCOPY; intrinsic;
procedure DSCOPYMSG; intrinsic;
begin {program pcopy}
opt := 4;  All transfers attempted, output enabled, command file disabled}
fnum := 0;
{copy local file NFTTEST to file NFTTARG on node SOMENODE}
spec := 'NFTTEST TO NFTTARG:SOMENODE[NSUSER.NSACCT]' + null; {string terminated
by ASCII null char}
DSCOPY (opt, spec, result);
if result[1] > 0 then DSCOPYMSG (result, fnum, r)
end.
```

Pascal: Multiple Transfer

In this application, the *opt* parameter is set to two (2). All transfers will be attempted. Primary output is disabled. The command file spec for multiple transfers is enabled. The *spec* parameter contains the "COPYFILE" name terminated by an ASCII null character. The "COPYFILE" must exist prior to execution of the program.

```
$standard_level 'hp3000', uslinit$
program pcopy2 (copyfile);
type
          small_int = -32768..32767;
const
          null = chr(0);  {ASCII null char}
var
          copyfile  : text;
          opt       : small_int;
          fnum      : small_int;
          r         : small_int;
          spec      : string [11];
          result    : array [1..2] of small_int;
procedure DSCOPY; intrinsic;
@COMPUTERTXT = procedure DSCOPYMSG; intrinsic;
begin {program pcopy2}
opt := 2;  {output disabled; attempt all transfers; command file enabled}
fnum := 0;
spec := '(copyfile)' + null;
rewrite (copyfile);
writeln (copyfile, '+ ; :SOMENODE [NSUSER.NSACCT]'); {global spec}
writeln (copyfile, 'SOURCE1 TO TARGET1');
writeln (copyfile, 'SOURCE2 TO TARGET2');
writeln (copyfile, 'SOURCE3 TO TARGET3');
close (copyfile);
DSCOPY (opt, spec, result);
if result[1] >> 0 then DSCOPYMSG (result, fnum, r)
end.
```

6 Intrinsics for Node and Environment Status

Following are descriptions of the two intrinsics described in this chapter — NSINFO and NSSTATUS:

NSINFO returns information about NS environments and your local node.

NSSTATUS returns information about services, servers, and NS users on local or remote nodes.

NSINFO Intrinsic

This intrinsic is used to programmatically obtain information about NS environments that have been created in your session. These are environments created by either a DSLINE *envID*, or a **REMOTE HELLO...;DSLINE=*envID*** command. This intrinsic also allows you to obtain some information about your local node (such as the local node name).

Syntax

```
                  BA              I
NSINFO          ( [envID], [envIDlength],

                  I       I
                [envnum], status

                 (IV)     (BA)
                [,itemnum1, item1]
                [,itemnum2, item2]
                [,itemnum3, item3]
                [,itemnum4, item4]
                [,itemnum5, item5])

                                where: BA    = Byte array
                                        I    = Integer
                                       IV    = Integer value
```

Parameters

envID

(input/output) **52-byte character array.** Specifies environment with matching environment identifier. See the discussion for an explanation of the use of *envID*, *envIDlength*, and *envnum*.

envIDlength

(input/output) **16-bit integer, by reference.** Length of environment ID in bytes. See the discussion for an explanation of the use of *envID*, *envIDlength*, and *envnum*.

envnum
(input/output) **16-bit integer, by reference.** Specifies environment by environment number. See the discussion for an explanation of the use of *envID*, *envIDlength*, and *envnum*.

status (output) **16-bit integer, by reference.** If the intrinsic call was successful, `0` is returned. Otherwise, an environment error number is returned.

NOTE

The following parameters appear in pairs to identify an item of data. One to five item pairs may be specified in the call. For brevity, all five item pairs will be described as *itemnum i* and *item i*, where *i* is 1, 2, ... 5.

inemnum i (input) **16-bit integer, by value.** Item number identifying the *item i* service to be performed.

item i (output) **Data type varies. By reference.** Value of the item specified by the corresponding *itemnum i*; the data type of the item value depends on the item itself.

Valid *itemnums* and the corresponding *items* are:

`1.` *node length* **16-bit integer.** Length of the following node name item, in bytes. This item must precede item 2.

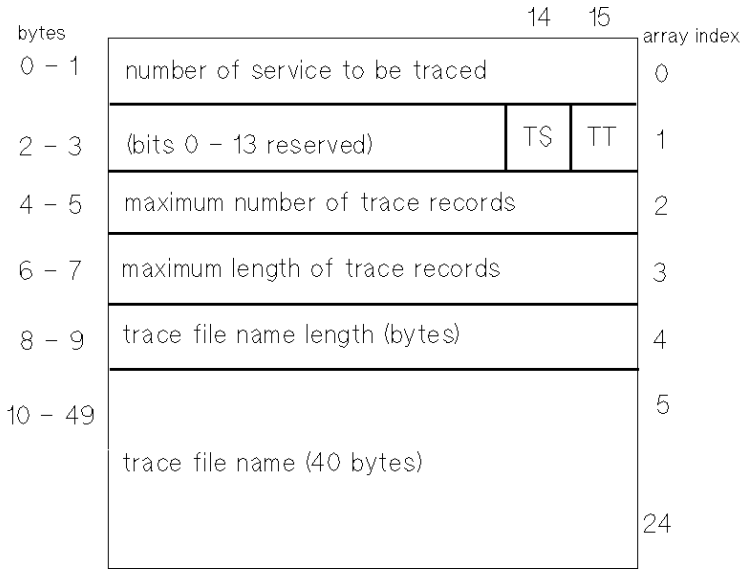
`2.` *node* **52-byte character array.** Name of the node where the environment is located. This item must be preceded by item 1.

`3.` *logon length* **16-bit integer.** Length of the following logon length item, in bytes. This item must precede item 4.

`4.` *logon* **54-byte character array.** Logon string (for example, *user.acct,group*) used to log on environment session. This item must be preceded by item 3.

`5.` *trace information* **Array of 25 16-bit integers.** Information about tracing of one service's NetIPC messages to the remote environment. In an `NSINFO` call using this item, the first word of the array must be set to the service for which the trace information is to be retrieved. The remainder of the fields will be filled in with the data by `NSINFO`. The array has the format shown in Figure 6-1.

Figure 6-1 NSINFO Trace Information Data Structure



Service number Number identifying what service this trace information on record pertains to.

`0` = All services

`1` = VT

`2` = NFT

`3` = RFA

`4` = RDBA

`5` = RPM

TS If the service is being traced, this value is `1`; otherwise, this value is `0`.

TT If the transport is being traced, its value is `1`; otherwise, its value is `0`.

`6.` *establishment event* **16-bit integer.** Event (that is, command) that set up the environment. Values are:

`0` = Environment created by a **DSLINE** command

`1` = Environment created by a **REMOTE** command.

`7.` *compression* **16-bit logical.** If true (odd), data compression will be used when data is transmitted on the service's connection. Currently, NFT and RFA allow data compression. All other services will ignore the compression setting.

8. *NFT service* **16-bit logical.** If true (odd), a process in this job or session has requested NFT service on *some* environment. This item is independent of the environment specified by *envID* or *envnum*.

9. *prompt length* **16-bit integer.** Length of the following prompt string, in bytes. This item must precede item 10.

10. *prompt* **8-byte character array.** String to be used as the REMOTE mode prompt. Item 9 must precede this item.

11. *VT service* **16-bit logical.** If true (odd), then VT service has been established on the environment. If false (even), there is no VT service for the environment.

12. *RFA service* **16-bit logical.** If true (odd), then RFA service has been established on the remote environment. If false (even), there is no RFA service for the environment.

13. *node envnums length* **16-bit integer.** Contains the number of *envnums* entered into the array supplied by item 14. If 0 is returned, then no environments are defined on the node specified in *envID*. This item must precede item 14.

14. *node envnums* **Array of 100 16-bit integers.** Will be filled with the *envnums* of environments defined on the node whose name is specified in *envID*. Must be preceded by item 13.

16. *logged on* **16-bit logical.** If true (odd), a remote session has been logged on for the environment. If false (even), there is no remote session for the environment.

18. *local node length* **16-bit integer**. The length, in bytes, of the name of the local node. This item is independent of the environment specified by *envID* or *envnum*. It must precede item 19. The NS transport must have been started for this to succeed

19. *local node name* **52-byte character array.** The name of the local node. This item must be preceded by item 18. The NS transport must have been started for this to succeed.

21. *service use count* **16-bit integer.** Count of the number of services currently sharing the environment.

25. *RPM son* **16-byte integer.** If true (odd), the remote environment contains an RPM son process. If false (even), the RPM service is not active for the environment.

26. *list environments length* — **16-bit integer.** Contains the number of *envnums* entered into the array supplied in item 27. If 0 is returned then no environments matched generic environment ID. This item must precede item 27.

27. *list environments* — **Array of 100 16-bit integers.** Will be filled with *envnums* whose *envID* matches the generic environment ID input as the *envID* parameter. Must be preceded by item 26.

36. *local node ARPA domain name length* — **16-bit integer.** The length, in bytes, of the local node's ARPA domain name. This item is independent of the environment specified by *envID* or *envnum*. It must precede item 37. The NS transport must have been started for this to succeed.

37. *local node ARPA domain name* — **255-byte character array.** The name of the local node's ARPA domain name. The NS transport must have been started for this to succeed.

38. *IP address* — **32-bit integer.** The IP address (4 bytes) of the node in the given environment. Currently, this option works only for $BACK environments. The value zero will be returned for all other environments.

Description

The NSINFO intrinsic is used to obtain information associated with environments defined in a job or session. There are two different ways that this intrinsic may be called. In the first method, *envID/envnum* define the specific environment or generic environment for which information is desired. In the second method, *envID* specifies the node name for which node-specific information is desired.

Selection of the environment may be done by environment ID string or by environment number. The presence and value of the *envID*, *envIDlength*, and *envnum* parameters determine which mode is used. Also, these parameters are both input and output; the selected environment's ID or number can be retrieved through them. The rules for using the *envID*, *envIDlength*, and *envnum* are:

- If *envID* and *envIDlength* are specified, and *envIDlength* is greater than 0, the environment will be determined by the environment ID in *envID*. If *envnum* is specified and its value is 0, the environment number will be determined by the environment specified in *envID*.
- If *envID* and *envIDlength* are omitted, or if *envIDlength* is 0, the environment will be selected by the environment number in *envnum*. If *envID* and *envIDlength* are specified, and *envIDlength* is 0, they will be set to the environment ID string and its length.
- If *envID* and *envIDlength* are omitted or *envIDlength* is 0, and *envnum* is 0, the default environment for the calling process will be selected. The default environment number will be returned in *envnum*, and its environment ID string and length will be returned in *envID* and *envIDlength* respectively, if specified. Note: The default environment is defined per process not per job or session.
- Some services interpret *envID* as either a node name or a generic environment ID (such as items 26, 27 and 13, 14). In calls that use an item number such as this, *envID* and *envIDlength* are required parameters.
- For services that are not dependent on any environment specification (such as items 8 or 18, 19), *envID*, *envIDlength*, and *envnum* are not required parameters.

Items that flag a certain condition, such as item 16 (logged on status) or 12 (RFA service established), will return either an odd (TRUE) or an even (FALSE) value.

This intrinsic may not be called in split stack mode.

Errors

Table 6-1 lists the errors that are returned in status upon completion of an NSINFO call. If an error is returned, all data returned from the call should be disregarded.

Table 6-1 NSINFO Errors

Error Number	Meaning
0	No error. Successful return from NSINFO.
1	Required parameter missing. Either the *envID* and *envIDlength* or the *envnum;status* must be specified.
2	Invalid *itemnum* value.
3	*item* parameter without a corresponding *itemnum* parameter.
4	*itemnum* parameter without a corresponding *item* parameter.
5	Invalid *envID* syntax. The correct syntax is *envID.domain.organization*.
10	Invalid *service* in *trace information* array. Must be between 0 and 6.
20	No environment entry exists for the selected environment.
23	The call selected the default environment but none was set by a prior **:DSLINE** or **:REMOTE** command.
24	The value in *envnum* does not correspond to an existing environment.
31	Total length of a *node name* or *envID* is invalid. Must be 1 to 50 characters.
32	First part (*node* or *envID*) of a *node name* or *envID* does not begin with an alphabetic character.
33	First part (*node* or *envID*) of a *node name* or *envID* is longer than 16 characters
34	First part (*node* or *envID*) of a *node name* or *envID* contains a non-alphanumeric character.
35	The *domain* part of a *node name* or *envID* is missing.
36	The *domain* part of a *node name* or *envID* does not begin with an alphabetic character.
37	The *domain* part of a *node name* or *envID* is longer than 16 characters.
38	The *domain* part of a *node name* or *envID* contains a non-alphanumeric character.

Error Number	Meaning
39	The *organization* part in a *node name* or *envID* is missing.
40	The *organization* part in a *node name* or *envID* does not begin with an alphabetic character.
41	The *organization* part in a *node name* or *envID* is longer than 16 characters.
42	The *organization* part in a *node name* or *envID* contains a non-alphanumeric character
44	NS transport not started. To start NS transport execute the **NETCONTROL START** command.
45	Unknown node
48	A specified item requires another item
96	DB is not pointing to the caller's stack
97	An item parameter was passed by reference to NSINFO which was outside of the caller's legal stack space.
98	Insufficient stack space for execution of procedure.

NSSTATUS Intrinsic

The NSSTATUS intrinsic returns information about services, servers, and NS users on local or remote nodes. This information is equivalent to the data displayed by NSCONTROL STATUS.

Syntax

```
                BA       IV          BA     IV    I
NSSTATUS  ([name], [namelength], {nodename], {nodelenth], status
                (IV)     (BA)
          [,itemnum1, item1]
          [,itemnum2, item2]
          [,itemnum3, item3]
          [,itemnum4, item4]
          [,itemnum5, item5]);

                                where: BA    = Byte array
                                        I    = Integer
                                       IV    = Integer value
```

Parameters

name (Input) **26-byte character array, by reference.** Service name, server name, or user name, depending on which items are specified. May be omitted for certain items.

namelength (Input) **16-bit integer, by value.** Length of *name* in bytes. May be omitted for certain items.

nodename (Input) **52-byte character array, by reference.** Name of a remote node from which information will be obtained. If omitted, NSSTATUS information will be obtained from the local node.

nodelength (Input) **16-bit integer, by value.** Length of *nodename* in bytes. If omitted, NSSTATUS information will be obtained from the local node.

status (Output) **Required 16-bit integer, by reference.** 0 is returned if call is successful; otherwise an error number is returned.

itemnum i (Input) 16-bit integer, by value. Item number identifying the item of data to be retrieved.

item i (Output) **Data type varies.** By reference. Array in which data identified by *itemnum i* will be returned. Required parameter if *itemnum i* present.

Itemnum/item parameters must appear in pairs. Up to five items of information can be retrieved by specifying one or more *itemnum/item* pairs.

Data Items

1. *service list* **2000-byte character array (output).** List of all services installed on indicated node. Format for the list is given in the data structure shown in Figure 6-2. NM capability is not required.

Service List Fields

Number of Services The number of *Service List Entries* included.

Service List Entry One for each configured server. Format for the list entry is given in the data structure shown in Figure 6-3.

Service List Entries

Service Name The name of the service; a port name for a local service or a socket name for a remote service.

Server Type Name The name of the server created for service requests to this service

ST If `0`, the service has not been started. If `1`, the service has been started.

RM If `0`, the service is local. If `1`, the service is remote.

Figure 6-2 Service List Data Structure

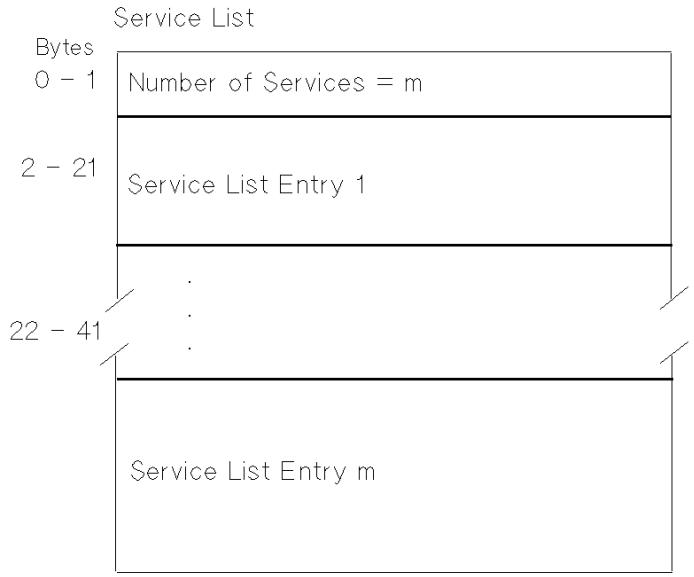
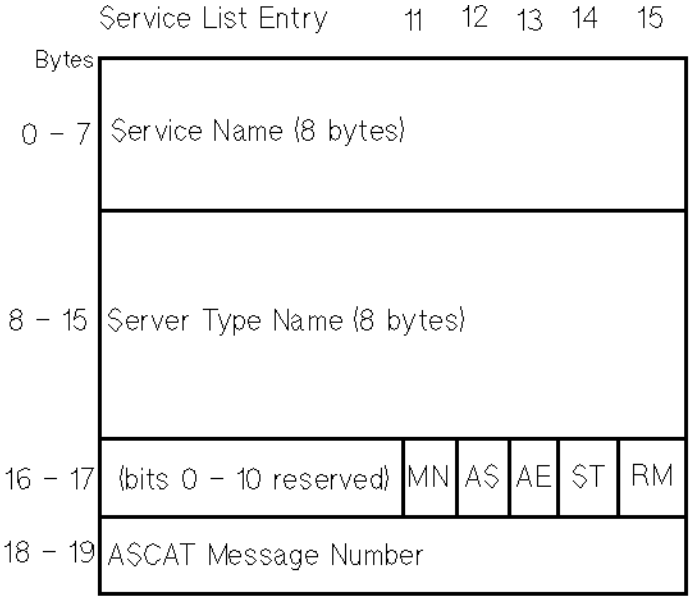


Figure 6-3 Service List Entry Data Structure



MN (1 bit): 0 = Service is local or remote
1 = Service is monitor

AS (1 bit): 0 = Autologon not supported by this service
1 = This service supports autologon

AE (1 bit): 0 = Autologon disabled
1 = Autologon enabled

ST (1 bit): 0 = Not started
1 = Started

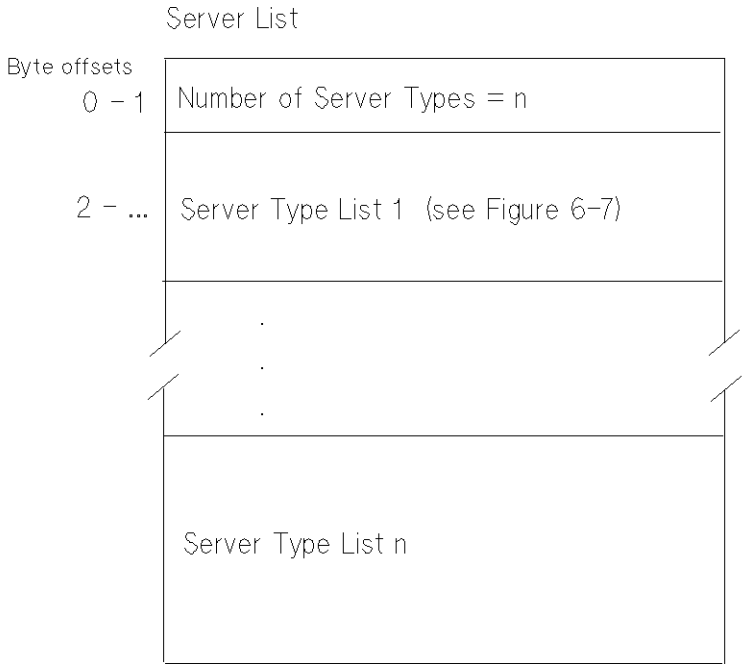
RM (1 bit): 0 = Local
1 = Remote

2. *server list* **2000-byte character array (output).** List of all servers installed on the indicated node. Format for the list is given in the data structures shown in Figure 6-4, Figure 6-7, and Figure 6-8.

Server List Fields

Number of Server Types	The number of Server Type Lists returned.
Server Type List	See the description of the fields under item 12.

Figure 6-4 Server List Data Structure



3. *user list* **2400-byte character array (output).** List of all users currently using NS on indicated node. Formats for the data structures of the user list and user list entries are shown in Figure 6-5 and Figure 6-6 respectively.

4. *service started* **16-bit integer.** Returns a value indicating one of the following conditions:

```
-1 = Named service not installed.
 0 = Named service not started.
 1 = Named service started.
```

NM capability is not required.

Figure 6-5 User List Data Structure

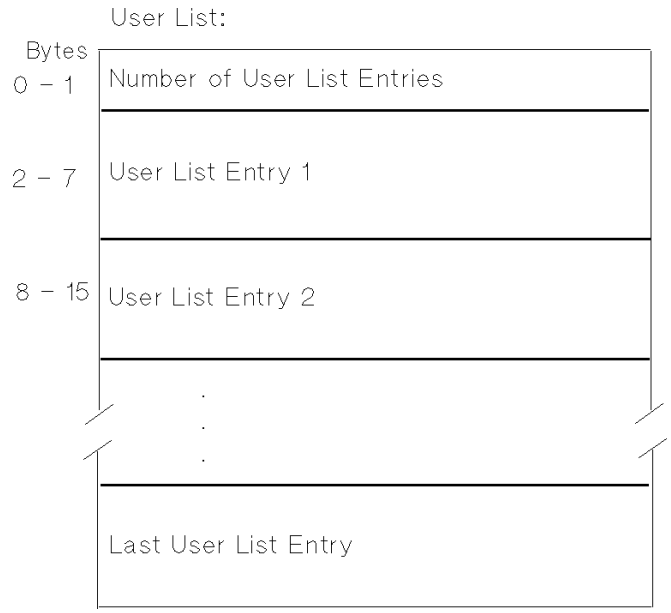
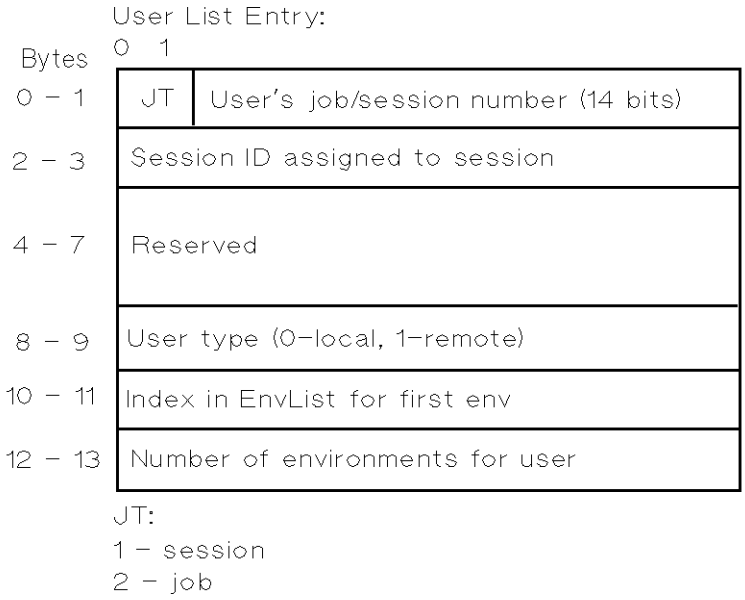


Figure 6-6 User List Entry Data Structure



5. *service local* **16-bit integer.** Returns a value indicating one of the following conditions:

```
-1 = Named service not installed.
 0 = Named service not remote.
 1 = Named service local.
 2 = Named service is monitor.
```

NM capability is not required.

6. *service server* **8-byte character array.** Name of server associated with named service. If the named server is not available, then eight ASCII spaces are returned. NM capability is not required.

7. *min servers* **16-bit integer**. Minimum number of servers for the named server type on the indicated node. A returned value of -1 implies that the named server does not exist.

8. *max servers* **16-bit integer.** Maximum number of servers of the named server type on the indicated node. A returned value of -1 implies that the named server is not available

9. *debug create* **16-bit integer.** Returns a value indicating one of the following conditions:

```
-1 = Named server not available.
 0 = Named server not created with debug option.
 1 = Named server is created with debug option.
```

10. *active servers* **16-bit integer.** Current number of active servers of the named server type on the indicated node. A returned value of -1 implies that the named server type is not available.

11. *reserved servers* **16-bit integer.** Current number of reserved servers for the named type on the indicated node. A returned value of -1 implies that the named server type is not available.

12. *server type list* **2000-byte character array (output).** List of servers of the named server type on the indicated node. The formats of the data structures are shown in Figure 6-7 and Figure 6-8.

Server Type List Fields

Server Name	The name of the server program file, without the group and account.
Minimum Servers	The minimum allowed number of servers for this type.
Maximum Servers	The maximum allowed number of servers for this type.

Create With Debug	If `0`, servers are not created with the debug option. If `1`, servers are created with a debug breakpoint.
Number of Servers	The number of active and reserved servers for this type.
Server Entry	One for each active and reserved server process. Format of the data structure of each server type entry is shown in Figure 6-8.

Server Entry Fields

Server PIN	The process ID number for the server.
Job Number	The number of the job or session into which the server has adopted. If the server is in the system environment, the job number is `0`.
Service	The service using the server (DSSERVER only). Values are: `0` — VTL (Normal VT Terminal Monitor) `1` — VT (Normal VT Application Monitor) `2` — RFA `4` — RPM `5` — VTR (Reverse VT Terminal Monitor) `6` — VTRL (Reverse VT Application Monitor)
AC	If `0`, the server is reserved. If `1`, the server is active

Figure 6-7 Server Type List Data Structure

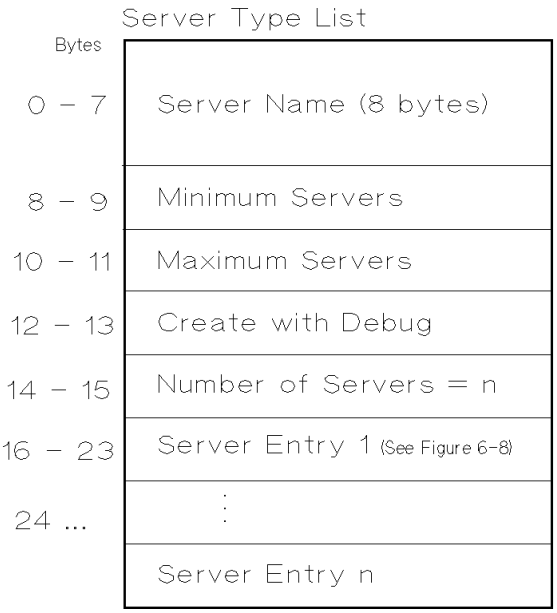
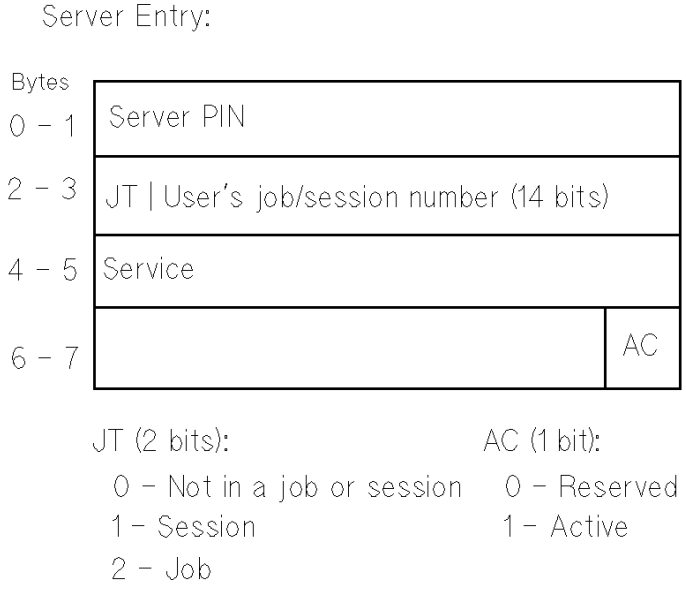


Figure 6-8 Server Entry Data Structure



13. *user.acct* **26-byte character array.** The user and account for the named session on the indicated node. If the named user is not present, then 26 ASCII spaces are returned.

14. *job number* **16-bit integer.** The job number for the named session on the indicated node. If the specified session is not present, then `-1` is returned.

15. *session ID* **16-bit integer.** The session ID assigned to the user on the indicated node. If the named user is not present, then `-1` is returned.

16. *user type* **16-bit integer.** Returns a value indicating one of the following conditions:

```
1 = Named user is not present.
0 = Named user is remote.
1 = Named user is local.
```

17. *num environments* **16-bit integer.** Number of active environments for the user on the indicated node. If the named user is not present, then `-1` is returned.

18. *environment list* **2400-byte character array (output).** List of environments for the user on the indicated node. Formats for the data structures of the environment list and environment list entries are shown in Figure 6-9 and Figure 6-10 respectively.

Figure 6-9 Env List Data Structure

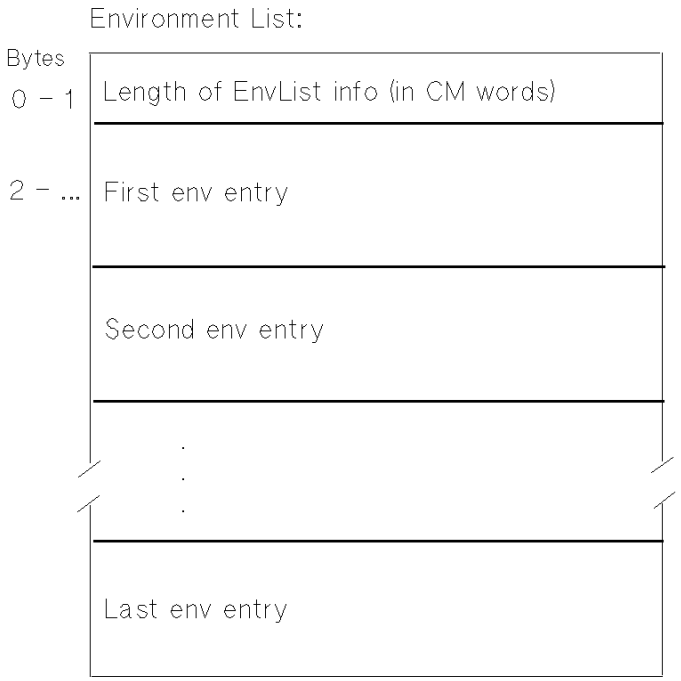
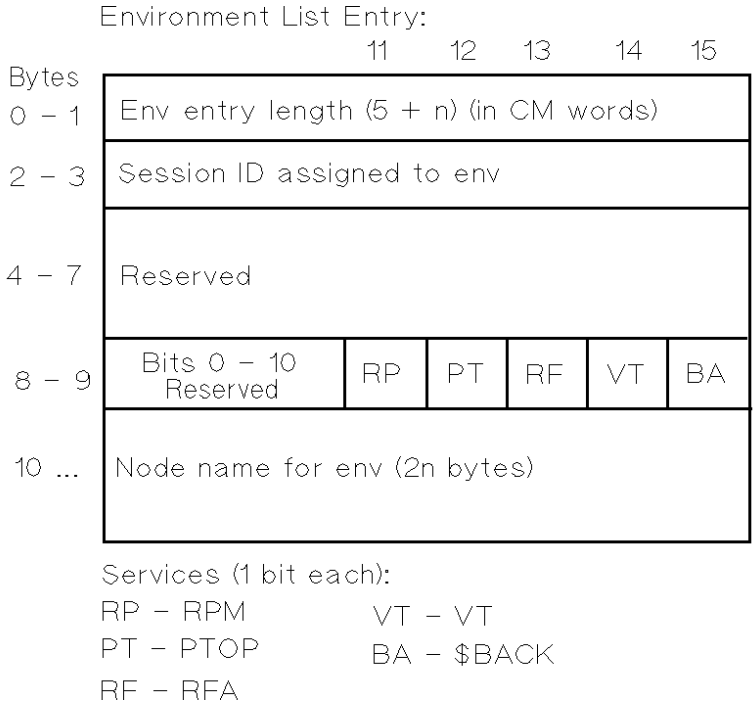


Figure 6-10 Env List Entry Data Structure



19. *autologon supported*

16-bit integer. Returns a value indicating one of the following conditions:

```
-1 = Named service not installed.
 0 = Named service does not support autologon.
 1 = Named service supports autologon.
```

29. *autologon enabled*

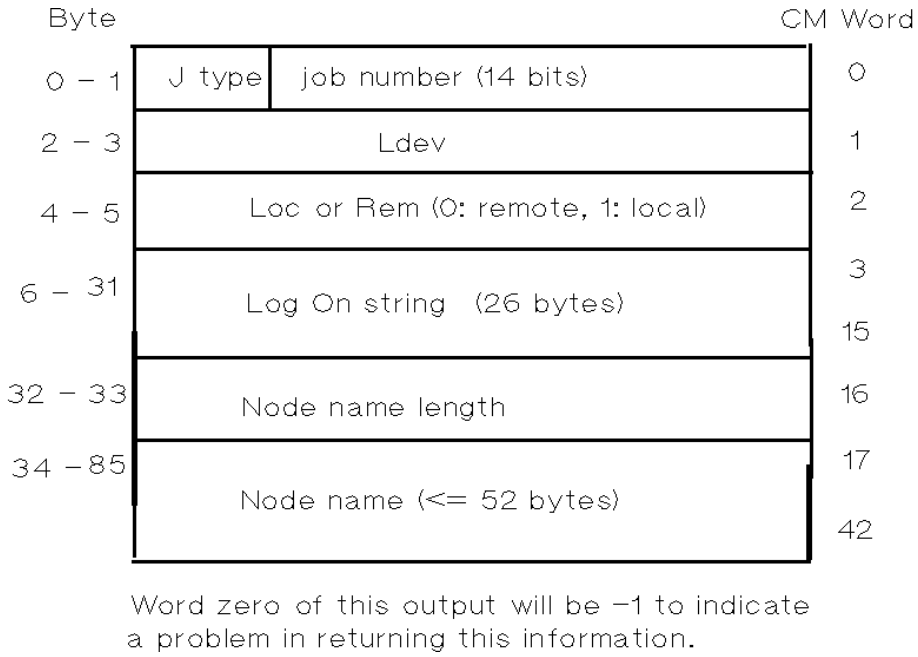
16-bit integer. Returns a value indicating one of the following conditions:

```
-1 = Named service not installed.
 0 = Autologon is off for the named service.
 1 = Autologoni is on for the named service.
```

21. *NS session's initiator information*

86-byte character array. The initiator's (job or session) information of the requested VT session on an indicated node. Format of the output array is shown in Figure 6-11.

Figure 6-11 Initiator's Information Format



Description

For items 1 through 3, the *name* and *namelength* parameters are ignored.

For items 4 through 6, the *name* and *namelength* parameters specify the service for which NSSTATUS information will be returned.

For items 7 through 12 the *name* and *namelength* parameters specify the server type.

For items 13 through 18, the *name* and *namelength* parameters specify the user for whom information will be returned. The format can be *user.acct* (specifying the user and account), or #J/Snnn (where J is the job number and S is the session number). Note: if there is more than one session for a *user.acct*, only information on the first session found will be returned.

The calling user must have Node Manager (NM) capability to retrieve all except item numbers 1, 4, 5, and 6. This intrinsic may not be called in split stack mode. The condition code remains unchanged.

Some of the data structure fields are described with the term "CM word." A CM word is a compatibility mode word. This is a 2-byte long, 2-byte aligned field. This corresponds to an SPL type LOGICAL or INTEGER or any 16-bit interger in other languages such as both CM and NM in PASCAL, FORTRAN, and COBOL.

NSSTATUS Intrinsic Examples

1. Determine if the VT service is started on the local node:

```
Name := 'VT';
NameLength:=2;
ItemNum:=4;
NSSTATUS (Name, NameLength, , , Status, ItemNum,
VtStarted);
```

   If the VT service is started, `VtStarted` will be set to `1`; otherwise it will be set to `-1` or `0`.

2. Determine if the VT service is started on the remote node `NODE1`:

```
Name := 'VT';
NameLength:=2;
Node Name:= 'NODE1'
NodeLength:=5;
ItemNum:=4;
NSSTATUS (Name, NameLength, NodeName, NodeLength,
Status, ItemNum, VtStarted0;
```

   If the VT service on `NODE1` is started, `VtStarted` will be set to `1`; otherwise it will be set to `-1` or `0`.

3. Determine if the NFT service is started and to find out how many active and reserved NFT servers currently exist on the local node:

```
Name := 'NFT';
NameLength:=3;
ItemNum4:=4;
ItemNum10:=10;
ItemNum11:=11;
NSSTATUS (Name, NameLength, , , ,Status,
     ItemNum4,NftStarted,
     ItemNum10, ActiveServers,
     ItemNum11, ReservedServers);
```

   This sets `NftStarted` to `1` if NFT is started, and returns the number of active and reserved NFT servers to `ActiveServers` and `ReservedServers`, respectively. This example uses the fact that NFT is the name of both the service and the server.

4. To obtain all the status information for services, servers, and users on the remote node `NODE1:`

```
NodeName := 'NODE1'
NSSTATUS (, , NodeName, NameLength, Status,
     ItemNum1,ServiceList,
     ItemNum2, ServerList,
     ItemNum3, UserList);
```

   This returns to the arrays *ServiceList*, *ServerList*, and *Userlist* the information formatted as defined in the data structures in this chapter.

5. Suppose #S1 (Manager.sys) on node A creates a remote session #S5 (using VT) on node B. The initiator of #S5 information can be obtained by running the following `NSSTATUS` intrinsic call on node B.

```
Name    := '#S5'     {or logon string of #S5}
NameLength :=3; ItemNum :=21;
```

```
NSSTATUS (Name, NameLength, , , Status, ItemNum,
InitiatorInfo);
```

If the intrinsic returns without an error then the following information will be in the `InitiatorInfo` record.

```
InitiatorInfo.JobType ---  1      { indicates Session}
InitiatorInfo.JobNum  ---  1      { session num }
InitiatorInfo.Ldev    ---  20     { logical device numb of #S1 }
InitiatorInfo.LocRem  ---  1      { local session. not a VT session}
InitiatorInfo.Logon   ---  'MANAGER.SYS'
InitiatorInfo.NodeLen ---  8
InitiatorInfo.NodeName---  'A.IND.HP'
```

6. Consider the above example. The same information can be obtained by running the following NSSTATUS intrinsic on Node C, which is on the same network.

```
Name := '#S5'           {or logon string of #S5}
NameLength:=3;
ItemNum:=21;
NodeName :='B',
NodeLength :=1;
```

```
NSSTATUS (Name, NameLength, NodeName,NodeLength,
Status, ItemNum, InitiatorInfo);
```

In this case, the intrinsic on Node C first goes to Node B and gets #S5's information. From this information, it gets the initiator's SID and node name and then goes to that node--in this case Node 'A'. It then gets the initiator's session information based on the SID — in this case #S1. Note: all the three systems involved in this scenario must have this fix.

See Table 6-2 for NSINFO Errors.

Table 6-2 NSINFO Errors

Error Number	Meaning
0	Successful completion of NSSTATUS intrinsic.
1	Required parameter missing.
2	Invalid item number. Item number must be an integer in the range 1 to 18.
3	Item number specified without corresponding item array.
4	Item array specified without corresponding item.
5	Name length too large.
6	Name length too small.

Error Number	Meaning
7	Expected a job or session number.
8	Non-numeric character in the job or session number.
9	Expected “#” as first character in Name parameter.
10	Bounds violation.
11	Unknown or invalid node name.
12	Node name specified without node name length.
13	Node name length specified without node name.
14	Communications error occurred with the remote NSSTATUS server. Most likely the remote system does not support the NSSTATUS intrinsic.
15	Node Manager (NM) capability required to retrieve the requested item.
16	DB not stack. (NM) capability required to retrieve the request.
17	Local NSSTATUS service has not been started. See Node Manager.
18	Internal Resource Error. Could not create reply port. Try again.
19	Remote node does not support option 21 (Remote node does not have this enhancement).
20	Option 21 works for only VT sessions. Given session is not a VT session.
21	Could not access initiator node using the node name in $BACK environment.

7 Remote Process Management

Remote Process Management (RPM) is a network service wherein a process using RPM intrinsics can create and terminate other processes. A created process can exist either on the same node as the creator or on another node. You can schedule a created process to be either dependent or independent of its creator. If a created process is independent, it can continue to execute even after its creator has expired.

RPM also permits a process to send information to the process it is creating in the same intrinsic call that creates the new process. The new process can then acquire this information by means of another RPM intrinsic call. This feature may help to facilitate subsequent communication between the processes. For example, the first process can send the name and location of one of its sockets to the process being created. The second process can then use this information to establish a connection to the first process.

NOTE

RPM can be used in conjunction with Network Interprocess Communication (NetIPC) to effectively manage distributed applications. When used in conjunction with NetIPC, the RPM master and slave processes must be executing concurrently. One or more users (or programs) can run these processes independently, or one process can initiate the execution of another by using RPM. You can employ the NetIPC `INITOPT`, `ADDOPT`, and `OPTOVERHEAD` intrinsics to facilitate your use of the *opt* parameter. Descriptions of those three intrinsics are included in this manual. For further information on the *flags*, *opt*, and *result* parameters, and for more information on NetIPC, refer to the *NetIPC 3000/XL Programmers Reference Manual*.

Common RPM Parameters

The following discussion of the *flags* and *result* parameters may help to clarify the more condensed information given under each intrinsic.

Flags Parameter

The *flags* parameter is a bit representation, 32 bits long, of various options. Normally an option is invoked if the appropriate bit is on (that is, set equal to `1`). Borrowing Pascal-type syntax, we shall use **[0]** to refer to the high order bit in the two-word parameter, *flags* **[31]** to refer to the low order bit, and a similar designation to refer to each of the bits in between. Bits that are not defined for a given intrinsic must be off (zero).

Result Parameter

If an RPM intrinsic call is successful, the result parameter will return a zero. Otherwise, the value returned represents an RPM error code. RPM error messages are listed in the *NS 3000/iX Error Messages Reference Manual*. You can also obtain the appropriate error message by calling `IPCERRMSG`.

NOTE

When nowait I/O is used, the *result* parameter is not updated upon completion of an intrinsic. Therefore, the value of result will indicate only whether the call was successfully initiated. To determine whether the call completed successfully, you can use the `IPCCHECK` intrinsic immediately afterward.

In addition, when called on an HP 3000, these intrinsics cause MPE condition codes to be set. Usually CCE indicates successful completion, CCL indicates failure, and CCG is either not used or represents a warning.

RPMCONTROL

Controls the execution of an existing process that was created by an RPMCREATE.

Syntax

```
RPMCONTROL  (pd [,location][,loclen],reqcode [,wrtdata][,wrtlen]
            [,readdata][,readlen][,flags][,result])
```

Parameters

pd (input) **16-byte array, by value.** Eight word program descriptor identifying the remote process to access.

location (input) **Character array, by reference.** Character string identifying the node on which the remote process resides.

loclen (input) **32-bit integer, by value.** Length in bytes of the *location* parameter.

reqcode (input) **32-bit integer, by value.** The number signifying the request for the RPMCONTROL call. Currently defined options for RPMCONTROL are:

- 20001 — Suspend execution of the remote process. No data is required, and none is returned.
- 20002 — Resume execution of the remote process at the point it was suspended. No data is required, and none is returned.

wrtdata (input) **Byte array, by reference.** Any data to be sent to the remote side for the request.

wrtlen (input) **32-bit integer, by value.** Length, in bytes of *wrtdata*.

readdata (output) **Byte array, by reference.** Data to be returned to the caller.

readlen (input/output) **32-bit integer, by reference**. On input, it is the maximum number of bytes expected in the *readdata* parameter. On output, it is the actual number of bytes received in the *readdata* parameter.

flags (input) **32 bits, by reference.** A bit representation of various options. No flags are currently defined.

result (output) **32-bit integer, by reference.** The result of the RPMCONTROL request; zero if no error.

Description

This intrinsic is used to control a remote process that was created with an RPMCREATE request. The only RPM control requests defined are suspend and resume.

Its calling sequence is similar to that of IPCCONTROL. The *pd* and *reqcode* parameters are the only required parameters. To control a remote process from a process which is not its creator, the *location* and *loclen* must be specified. RPMCONTROL uses these parameters to establish a new connection to the remote machine to perform the request.

The *wrtdata* and *wrtlen* parameters must be specified if the *reqcode* specified requires data to be transferred to the remote node.

Similarly, *readdata* and *readlen* should be included if the *reqcode* specified returns data to the caller upon completion of the request. Presently, there are no request codes that require these parameters.

The ability to specify the suspend and resume request codes (20001, 20002) in an RPMCONTROL request is limited to programs that have Privileged Mode (PM) capability. Great care should be taken when using these request codes. If a locally created process is suspended, (that is, a process that was created with a location and logon matching that of the caller, or the location and logon were omitted or blank), and the creator terminates without issuing either an RPMKILL or an RPMCONTROL with the resume request code, the creator process may hang. Therefore, these functions should not be used to manipulate locally created processes unless absolutely necessary.

If the process to be suspended is on a remote node, there is no danger of hanging the process. However, a process that has been suspended in this manner cannot be killed or resumed except by explicitly calling either RPMKILL or RPMCONTROL with the resume request code.

If a process is in a state where the node cannot suspend it, RPMCONTROL will return an error code in the result parameter indicating this. A process may not be suspendable because: the process is suspended by a system process, the process is waiting for critical resources, or the process is dying.

RPMCREATE

Creates and activates a process and, if necessary, creates a remote session for that process to run in. Process Handling (PH) capability is required.

Syntax

RPMCREATE (*progname,namelen* [,*location*][,*loclen*][,*login*][,*loginlen*]
[,*password*][,*passwdlen*][,*flags*][,*opt*][,***pd***][,***result***])

Parameters

progname (input) **Character array, by reference.** The name of the program for which a process is to be created.

namelen (input) **32-bit integer, by value.** The length in bytes of the program name.

location (input) **Character array, by reference.** The node name or environment ID indicating where the process should be created.

loclen (input) **32-bit integer, by value.** The length in bytes of the location name.

login (input) Character array, by reference. A logon sequence for the local or remote node on which the process is to be created, in the form [*session*,]*user.acct*[,*group*].

loginlen (input) **32-bit integer, by value.** The length in bytes of the logon sequence.

password (input) **Character array, by reference.** Password(s) to be used with the logon sequence, in the form [*userpass*][,*acctpass*][,*grouppass*].

passwdlen (input) **32-bit integer, by value.** The length in bytes of the password parameter.

flags (input) **32 bits, by reference.** A bit representation of various options. The following flags are defined:

- **flags [0]** (input). Indicates that all environment information given in a prior **DSLINE** command (for the remote environment in question) should be ignored when the new process is created. For example, if this flag is on, a logon sequence provided in the LOGON option of a **DSLINE** command will not override the logon given in this RPMCREATE call.

- *flags* **[1]** (input). Causes the calling process to wait in this intrinsic call until the new process terminates. (The RPMCREATE call will complete only after the new process finishes executing.)
- *flags* **[2]** (input). Causes RPM to create a process using RPM protocols from HP's software release UB Delta 1 or earlier. In software release UB Delta 1 or earlier, RPM flags 3 and 30 are not supported.
- *flags* **[3]** (input). Causes RPMCREATE to fail if the remote node has a version of RPM installed that is from HP's software release UB Delta 1 or earlier. By default, if the remote node does not have the newer RPM installed, RPMCREATE will act as if *flags* **[2]** is set and conform to the older format.
- *flags* (input). Causes the remote node to create the process into a session that may accommodate multiple RPM created processes. By default RPMCREATE will only create one remote process per remote session.
- *flags* **[31]** (input). Makes the created process dependent on the creator. When the creator process dies, the created process will be killed. (If this bit is off, the created process will continue executing on its own.) Default: off; created process is independent of its creator.

opt **Record or byte array, by reference.** A list of options, with associated information. The following options are defined:

- RPM string (option code 20000, *n-byte* array). Permits information to be sent to the created process in the data portion of the opt parameter, such as a socket name. More than one RPM string can be included. The strings may be retrieved by the new process (using RPMGETSTRING) in the same order that they occur in the *opt* parameter.
- Entry point (code 22001, *n-byte* array). The data portion of this parameter contains a string, terminated by a blank, specifying the entry point (label) in the program where execution is to begin when the process is activated. A blank by itself indicates the primary entry point. This parameter corresponds to item number 1 of the MPE/iX CREATEPROCESS intrinsic. The contents of this

option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic. Default: primary entry point.

- Program parameter (code 22002, 2-byte integer). The data portion of this parameter contains an integer value used to transfer control information to the new process. The word will be accessible on the stack of the new process at location Q-4. This parameter corresponds to item number 2 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the item parameter in the CREATEPROCESS intrinsic.
- Create flags (code 22003, 2-byte bit map). The data portion of this parameter contains a word whose bits specify loading options, as follows. (Bit 15 is the least significant bit, bit 0 the most significant bit.) This parameter corresponds to item number 3 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic.
  - bit 15 — Ignored; should be off.
  - bit 14 — LOADMAP bit. If on, a listing of the newly allocated program is produced on the job/session listing device. This map shows the Code Segment Table (CST) entries used by the new process. Default: off
  - bit 13 — DEBUG bit. If on, DEBUG is called at the first executable instruction of the new program. You must have read/write access to the program file of the new process. If the new process requires privileged mode, you must be a privileged user. Default: off.
  - bit 12 — NOPRIV bit. If on, the slave program is loaded in non-privileged mode. If this bit is off, the program is loaded in the mode specified when the program file was prepared. Default: off.
  - bits 10 and 11 — LIBSEARCH bits. A coded bit pattern that denotes the order in which libraries are to be searched for the program. The default is 00:

    00 = System Library only.

    01 = Account Public Library, then System Library.

10 = Group Library, then Account Public Library, then System Library.

- bit 9 — NOCB bit. If on, file system control blocks are established in an extra segment. If off, control blocks may be established in the Process Control Block Extension (PCBX) area. Default: off. This bit should be set on if the new process uses a large stack.
- bits 7 and 8 — reserved for MPE; should be off (00).
- bits 5 and 6 — STACKDUMP bits. A coded bit pattern that controls the enabling/disabling of stack dumping in the event of a program abort. The default is 00:

  00 = enable stack dumping for new process only if enabled at master level;

  01 = enable stack dump unconditionally;

  10 = same as 00;

  11 = disable stack dump unconditionally.
- bit 4 — reserved for MPE; should be off.
- bits 0 to 3 — These four bits are used only if the STACKDUMP bits are 01; otherwise they are ignored. Bits 1–3 represent portions of the slave program's stack. If bit 3 is on, the portion of the stack from DL to QI (Q-initial) is dumped; if off, this portion is not dumped. If bit 2 is on, the portion of the stack from QI to S is dumped. If bit 1 is on, the portion of the stack from Q-63 to S is dumped. If bit 0 is on, the stack dump is interpreted in ASCII characters in addition to octal values; if off, no ASCII interpretation is performed. The default for each of these bits is off.

The default conditions noted above take effect if the Create flags option (*flags*) or the entire *opt* parameter is omitted.

- Initial stack size (code 22004, 2-byte integer). The data portion of this parameter contains an integer (Z–Q) denoting the size, in words, of the local stack area bounded by the initial Q and Z values. This parameter corresponds to item number 4 of the MPE/iX CREATEPROCESS intrinsic.

The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic. Default: The value specified in the program file

- Initial dlsize (code 22005, 2-byte integer). The data portion of this parameter contains an integer (DB–DL) denoting the size, in words, of the user-managed stack area bounded by the DL and DB values. This parameter corresponds to item number 5 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic. Default: The value specified in the program file.
- Max stack size (code 22006, 2-byte integer). The data portion of this parameter contains an integer (Z–DL) denoting the maximum word size allowed for the process's stack area (bounded by the DL and Z values). This parameter corresponds to item number 6 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic. Default: The value specified in the program file or (if none is specified there) the current stack size.
- Priority (code 22007, 2-byte array). The data portion of this parameter contains a string of two ASCII characters (AS, BS, CS, DS, or ES) indicating the priority class in which the new process is scheduled. Default: The same priority as the calling process. This parameter corresponds to item number 7 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic.
- $STDIN file name (code 22008, *n-byte* array). The data portion of this parameter contains the name of a file to which all input requests to $STDIN will be directed. This parameter corresponds to item number 8 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic. Default: With an interactive logon (HELLO, REMOTE HELLO), input requests will be directed to $STDIN. With the logon option (either the LOGON= parameter in a **DSLINE** command or the *login* parameter of RPMCREATE), input requests will be directed to the empty file $NULL.
- $STDLIST file name (code 22009, *n-byte* array). The data portion of this parameter contains the name of a file to which all output requests to $STDLIST will be directed. This parameter corresponds to item number 9 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic. Default: With an interactive logon (HELLO, REMOTE HELLO), output requests will be directed to $STDLIST. With the logon option (either the LOGON=

parameter in a **DSLINE** command or the *login* parameter of RPMCREATE), output requests will be directed to the empty file $NULL.

- Info string (code 22011, *n-byte* array). The data portion of this parameter contains an information string that is passed to the new process. This string will be accessible on the new process's stack at a byte address that is stored at location Q-5. If this option is included, you must also include the Info string length option (22012). This parameter corresponds to item number 11 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic.
- Info string length (code 22012, 2-byte integer). The data portion of this parameter contains an integer specifying the byte length of the info string given by the previous option. The Info string length option must be included if the Info string option is included. This parameter corresponds to item number 12 of the MPE/iX CREATEPROCESS intrinsic. The contents of this option parameter should conform to the value of the *item* parameter in the CREATEPROCESS intrinsic.
- Logon timeout; real (code 22100, 4-byte real value). The data portion of this parameter contains a real value representing the number of seconds you are willing to wait for a remote logon, performed by RPMCREATE, to complete. If the remote session is not established during this time, you will receive an error. Default: 120.0 seconds.
- Logon timeout; integer (code 22102, 4-byte integer). This is the same as option 22100 except that the data portion of this parameter contains an integer representing the number of milliseconds that you are willing to wait for the remote logon, performed by RPMCREATE to complete. If both options are present in the *opt* array, then the entry with the lowest array index will be the one selected.

pd (output) **16-byte array, by reference.** A program descriptor used to identify the created process. This value, randomly generated, is presumed to be unique across all nodes. A valid program descriptor is always a non-zero value.

result (output) **32-bit integer, by reference.** The error code returned; zero if no error.

Description

The RPMCREATE intrinsic enables the calling process to create and activate another process — that is, to initiate execution of another program. The new process may be on a remote system. (RPM does not extend the process management capabilities of a particular operating system, such as MPE, across a network.)

Normally, RPMCREATE allows only one remote process per remote session. Bit 30 of the RPM *flags* parameter allows multiple RPM remote processes in a remote session. This session-sharing option of RPM is only available in HP's software release UB delta 2 or later. The only required parameters are *progname*, *namelen*, *location*, and *loclen*. (The intrinsic is option variable.) In order for multiple processes to reside in a common remote session, three criteria must be satisfied:

- All the processes must have been created with bit 30 of the *flags* parameter set. Remote processes created without this bit set will not share sessions with processes that do have it set.
- All the processes must have been created from processes residing within the same session on the local node. Processes that are running in different local sessions will RPMCREATE remote processes into different remote sessions.
- All the processes must have the same logon string including session name, if any. All the necessary passwords must match.

The remote session may be any session that RPMCREATE would normally use, including VT-created sessions, that is, sessions created using the **REMOTE HELLO** command.

The new process will run in an existing remote session (created by REMOTE HELLO) if you specify the appropriate environment ID in the *location* parameter. A new session will be created on the remote node if one does not already exist. RPM uses the logon sequence specified in the *login* parameter (and the password in *password*) unless (1) a logon is specified in a prior **DSLINE** command for the remote environment (in the LOGON option), and (2) bit 0 of the *flags* parameter (the high-order bit, which causes the DSLINE information to be ignored) is off. In other words, the order of priority (from high to low) is: existing session; DSLINE logon; RPMCREATE logon. But the "ignore DSLINE information" flag forces the use of the RPMCREATE logon instead of a DSLINE logon.

If the new process is to be created on a remote node and the login parameter is omitted, then a remote session must already exist or logon information must be given in a prior **DSLINE** command. For example, let's say that a previous **DSLINE** command has defined "S" as an environment ID for node FINANCE, with logon *USER.ACCT*. Then an RPMCREATE call giving "S" as the location and omitting the *login* parameter will create a session on FINANCE for *USER.ACCT*.

Once an RPMCREATE call has been made for a remote environment, you cannot issue remote commands for that environment until the remote process has terminated (for example, has been killed via RPMKILL). If an independent RPM process has been left in the remote environment after the process that created it has terminated, and you issue a **DSLINE ;CLOSE** command for the remote environment, you will first be asked whether you want to kill the remote RPM process. If you kill the

remote process and then do not abort the remote session, you can subsequently issue commands in the remote session. See "Releasing a Remote Environment" in the "Virtual Terminal" chapter of this manual.

RPMCREATE can also create a new process on the local node. The new process will be created in your local session if (1) the *location* and *login* parameters are omitted or blank or (2) *location* is the local node name and *login* is the logon for your local session (including session name, if any). A new process will also be created on your local node, but in a different session, if (1) you specify your local node name, and a logon different from your own and (2) software loopback has been configured and activated for your local node.

Bit 31 (the low-order bit) of the *flags* parameter determines whether the newly created process will be dependent on its creator or independent (the default). A local dependent process that was created under the same logon as its creator will terminate automatically when the creator terminates. (In order to conserve system resources, you should make local processes independent; that is, set the bit off.)

A remote dependent process will terminate if:

- RPMKILL has been called for the dependent process.
- The creator process terminates before calling RPMKILL.
- The transport fails.
- The system on which the creator is running fails.

If the remote process is independent, it will continue to execute unless explicitly terminated by RPMKILL. Dependent mode ensures that the new process will not become an "orphan" in the event of a program, system, or network link failure. However, independent mode is less costly in terms of resources: the connection set up for the RPMCREATE is not maintained after the remote process is created. You should normally use independent mode for processes that are expected to terminate themselves.

Preferred Method of Creating Interactive Programs

RPM works best to create non-interactive server programs on a remote system. If you use RPM to create interactive programs, some restrictions exist (described in a following section). Therefore, as an alternative to using the RPMCREATE intrinsic to create interactive programs, HP recommends that you call the **REMOTE RUN** command from the COMMAND intrinsic. Using this method will suspend the master process in the COMMAND intrinsic while the slave program runs. Only use RPMCREATE if you require parallel execution of a master and slave process.

Using $STDIN and $STDLIST in a Precreated VT Session

Beginning with MPE/iX release 2.2, remote RPM slave processes can post interactive I/O to $STDIN and $STDLIST through a precreated VT session. This allows interactive I/O from remote programs to appear on a local terminal.

Before creating an RPM slave, use the **REMOTE HELLO** command to create a remote session. After the session is established, you can use the **RPMCREATE** command to create an RPM son in the remote session. The *location* parameter should contain the same environment name that was specified in the **REMOTE HELLO** command. Do not specify the *login* or *loginlen* parameters, nor are the $STDIN/$STDLIST options 22008/22009 required.

Restrictions when Using RPMCREATE to Create Interactive Programs

Because RPM was not designed to create interactive programs, there are caveats you should be aware of before using RPM for your application.

Do not press the system **BREAK** or the subsystem break **[CTRL] Y** while a remote application is running.

Do not create an RPM slave in a VT session that is in break. This will cause both the local and remote application to hang. Neither the local or remote session can log off.

Opt Parameter Format

The *opt* parameter, which denotes various options, contains an integer code for each option along with associated information. It is not necessary to know the internal structure of this parameter to use it. The "*opt* parameter manipulation intrinsics," INITOPT, ADDOPT, and OPTOVERHEAD, enable you to add option information without concerning yourself with the parameter's structure. However, a knowledge of the *opt* parameter's structure can help you determine an appropriate size for the array. (The parameter must be defined as a byte array or as a record structured in the manner described below. If your program is written in a language that supports dynamically allowed arrays, the OPTOVERHEAD intrinsic may be used to determine the size of the array.)

The *opt* parameter consists of these fields:

- length, in bytes, of option entries and data (2-byte integer);
- number of entries (2-byte integer);
- option entries (8 bytes per entry);

- data associated with the option entries (variable length).

Each 8-byte option entry, in turn, consists of the following fields:

- option code (2-byte integer);
- offset (relative to the base address of the opt parameter) indicating the location of the data for this option entry (2-byte integer);
- length, in bytes, of the data (2-byte integer);
- unused (2 bytes).

If the parameter is declared as a simple byte array, it must be large enough to contain 4 bytes for the first two fixed-length fields, 8 bytes for each option entry, plus the actual data. That is:

```
4  +  8 * numentries  +  datalength
```

RPMGETSTRING

Allows a created process to retrieve information strings passed to it by its creator in the RPMCREATE call.

Syntax

```
RPMGETSTRING    (rpmstring,rpmstringlen,result)
```

Parameters

rpmstring
(output) **Character array, by reference.** The string passed in the *opt* parameter of the RPMCREATE call that created this process.

rpmstringlen
(input/output) **32-bit integer, by reference.** The maximum byte length allowed for the rpmstring. The returned value indicates the actual length of the returned rpmstring, zero if no string was passed by the creator process.

result (output) **32-bit integer, by reference.** The error code returned; zero if no error.

Description

The RPMGETSTRING intrinsic allows a process created by an RPMCREATE to obtain the information string passed to it by the creator process in the RPMCREATE call. All the parameters are required.The string obtained in this manner may contain any useful information. For example, it could contain the name of a (call) socket belonging to the creator along with the name of the node on which the creator is executing. The created process can look up this socket name in order to acquire a destination descriptor for it. After creating a socket of its own, it can establish a connection to the creator process.

If the *opt* parameter of RPMCREATE contained more than one RPM string, you can issue several RPMGETSTRING calls to retrieve the strings.

For example:

Creator process:

```
ADDOPT (Opt, 0, 20000, Length1, RpmString1);
ADDOPT (Opt, 1, 20000, Length2, RpmString2);
ADDOPT (Opt, 2, 20000, Length3, RpmString3);
RPMCREATE (... Opt ...);
```

Created process:

```
RPMGETSTRING (RpmString1, Length1, Result);
RPMGETSTRING (RpmString2, Length2, Result);
RPMGETSTRING (RpmString3, Length3, Result);
```

For another illustration of the use of this intrinsic, see the program examples at the end of this chapter.

RPMKILL

Terminates a process created by the RPMCREATE intrinsic. Process Handling (PH) capability is required.

Syntax

```
RPMKILL ( pd[,location][,loclen][,result])
```

Parameters

pd (input) **16-byte array, by value.** The program descriptor returned by the RPMCREATE intrinsic.

location (input) **Character array, by reference.** The node name or environment ID indicating where the process to be killed is located.

loclen (input) **32-bit integer, by value.** The length in bytes of the location name.

result (output) **32-bit integer, by reference.** The error code returned; zero if no error.

Description

The RPMKILL intrinsic terminates a process that was created by RPMCREATE. It also deletes the remote environment if that environment was established by RPMCREATE. Any process on any node may call RPMKILL to kill a remote RPM process, as long as it has the correct program descriptor. To kill a process on your local node that utilizes another session, you must have configured the network for software loopback. To kill a local process in your session, you must be the creator of that process.

The only required parameter is *pd* (option variable).

The *location* and *loclen* parameters are also required in order to kill a remote process either if the remote process is independent or if the calling process is not the creator. RPM needs these to open a new connection to the remote node before killing the remote process.

Use the RPMKILL intrinsic only to terminate perpetually running remote processes. Otherwise, use the “Shutting Down Sockets and Connections” procedure described in the *NetIPC 3000/XL Programmer's Reference Manual*.

ADDOPT

Adds an option entry to the *opt* parameter.

Syntax

```
ADDOPT (opt, entrynum, optioncode, datalength, data[, result])
```

Parameters

opt
(input/output) **Record or byte array, by reference.** The *opt* parameter to which you want to add an entry. Refer to "Common Parameters" for more information on the structure of this parameter.

entrynum (input) **16-bit integer, by value.** Indicates which entry is to be initialized. The first entry is entry zero.

optioncode
(input) **16-bit integer, by value.** The entry's option code, identifying the option.

datalength
(input) **16-bit integer, by value.** The length (in bytes) of the data associated with the option.

data (input) **Byte array, by reference.** The data associated with the option.

result (output) **16-bit integer, by reference.** The error code returned; zero if no error.

Description

The ADDOPT intrinsic specifies the values of an *opt* parameter's option entry fields and adds any associated data. The intrinsic also updates the size of the *opt* parameter.

The parameter must be initialized by INITOPT before options are added by ADDOPT. Consider this program fragment:

```
data_offset:=10;
```
{10 bytes from beginning of data array]

```
INITOPT(opt, 1);
```
{one option entry}

```
ADDOPT (opt, 0,
8, 2,
data_offset);
```
{first entry is entry zero, option code 8; entry's data area contains a 2 byte integer specifying an offset from *data* parameter address}

```
IPCSEND (cd,
data, dlen, ,
opt, result);
```
{sends data located at offset from data address specified in opt}

INITOPT and ADDOPT allow you to initialize the *opt* parameter for use in another intrinsic. These auxiliary intrinsics make the structure of the *opt* parameter largely transparent.

Condition codes returned by ADDOPT are:

- CCE — Succeeded.
- CCL — Failed because of a user error.
- CCG — Not returned by this intrinsic.

This intrinsic may be called in split stack mode.

INITOPT

Initializes the *opt* parameter so that entries may be added.

Syntax

```
INITOPT (opt, eventualentries[, result])
```

Parameters

opt (output)	**Record or byte array, by reference.** The *opt* parameter which is to be initialized. Refer to "Common Parameters" for more information on the structure of this parameter.
eventualentries (input)	**16-bit integer, by value.** The number of option entries that are to be placed in the *opt* parameter.
result (output)	**16-bit integer, by reference.** The error code returned; zero if no error.

Description

The INITOPT intrinsic initializes the length and number-of-entries fields (that is, the first 4 bytes) of the *opt* parameter. This must be done before options are added to the parameter by means of the ADDOPT intrinsic.

Condition codes returned by this intrinsic are:

- CCE — Succeeded.
- CCL — Failed because of a user error.
- CCG — Not returned by this intrinsic.

This intrinsic may be called in split stack mode.

OPTOVERHEAD

Returns the number of bytes needed for the *opt* parameter in a subsequent intrinsic call, not including the data portion of the parameter.

Syntax

optlength := OPTOVERHEAD (*eventualentries*[,***result***])

Parameters

optlength (returned function value) **16-bit integer.** The number of bytes required for the *opt* parameter, not including the data portion of the parameter.

eventualentries (input) **16-bit integer, by value.** The number of option entries that will be placed in the *opt* parameter.

result (output) **16-bit integer, by reference.** The error code returned; zero if no error.

Description

This function returns the number of bytes needed for the *opt* parameter, excluding the data area. The first parameter is required.

Condition codes returned by this intrinsic are:

- CCE — Succeeded.
- CCL — Failed because of a user error.
- CCG — Not returned by this intrinsic.

This intrinsic may be called from split stack mode.

RPM Program Examples

The following two Pascal programs illustrate the use of the RPM intrinsics. The first program creates a new process (the second program) on a remote node. It also creates a remote session for the second program to run in. At the same time, it passes strings containing a socket name and a node name to the remote process. This information enables the second program to establish a connection to the first.

In greater detail, the first program:

- creates a call socket and names it;
- reads the name of the node on which it is running;
- initializes the *opt* parameter with strings containing the socket's name and the name of the node on which it is running;
- creates a new process with the program name CREATURE on a remote node, passing the strings to the new process;
- waits for a connection request from the remote process (and establishes the connection);
- shuts down its call socket;
- executes a loop in which it:
    - calls a procedure that receives a message by executing two IPCRECV loops. The first loop determines the incoming message length. The second loop receives data until all the pieces of the message have been received.
    - prints the message that was received;
- receives a "last message" termination request;
- sends a "termination confirmation message" in response to the termination request;
- receives a *result* parameter value of 64 ("REMOTE ABORTED CONNECTION") in response to a receive request;
- releases its VC socket.

The execution of the second program is initiated by the first program. The second program:

- obtains the socket name and node name passed by the creator;
- uses these names to acquire a destination descriptor for the socket;
- creates a call socket for itself, sends a connection request to the creator's socket, and establishes a connection to the creator process;

- opens a data file;
- executes a loop in which it:
  - reads a line of data from the data file;
  - stores the length (number of bytes) of the data in the first part of the message;
  - stores the data itself in the second part of the message;
  - sends the message on the connection, including the message length as the first two bytes of the message;
- after all the data is transmitted from the data file, sends a "last message" that will be recognized by the receiving program as a termination request;
- receives a "termination confirmation message" and shuts down the connection by releasing its VC socket.

NOTE

Process Handling (PH) capability is required to use the following programs.

Before running the first program, you must associate the remote node with the environment ID "REMNODE" via the **DSLINE** command. For example, issue the command **DSLINE REMNODE=*ASTRO*** if the remote node where the second program resides is named "ASTRO."

RPM Program 1

```
$standard_level 'HP3000', uslinit$
program creator (input,output);
const
   maxdata = 2000;
   maxname = 20;
type
   smallint       = -32768..32767;
   datatype       = packed array [1..maxdata] of char;
   nametype       = packed array [1..maxname] of char;
   byte           = 0..255;
var
   calldesc        : integer;
   result          : integer;
   progname        : packed array [1..15] of char;
   location        : packed array [1..16] of char;
   login           : packed array [1..25] of char;
   flags           : packed array [0..31] of boolean;   {32 contiguous bits}
   pd              : packed array [0..15] of byte;
   destdescriptor  : integer;
   vcdesc          : integer;
   dlen            : integer;
   i               : integer;
   data            : datatype;
   len             : smallint;
   datastr         : string[maxdata];
   socketname      : nametype;
   nodename        : nametype;
   opt             : packed array [1..50] of byte; {INITOPT and ADDOPT
                                    will structure the array for us}

procedure terminate; intrinsic;

{RPM and IPC intrinsic declarations}

procedure ipccreate; intrinsic;

procedure ipcname; intrinsic;

procedure initopt; intrinsic;

procedure addopt; intrinsic;

procedure rpmcreate; intrinsic;

procedure ipcrecvcn; intrinsic;

procedure ipcerrmsg; intrinsic;

procedure ipcrecv; intrinsic;

procedure ipcshutdown; intrinsic;

procedure ipcsend; intrinsic;

procedure leave(result: integer);
     var msg: string[80];
          i, len, newresult: integer;

begin
ipcerrmsg (result, msg, len, newresult);
if newresult = 0 then
     begin
     setstrlen(msg, len);
     writeln(msg);           {print error message}
     end
```

```
else
     writeln('IpcErrMsg result is ', newresult:1);
terminate;
end;
procedure check ( result : integer);
{error procedure}
begin
if result << >> 0 then
     leave (result);       {failed}
end;

{error handling procedure}
{The following procedure receives one message that was sent via an ipcsend call.
It assumes that the length (number of bytes) of the   message was sent as the
first 2 bytes of data and that the length value does not include those 2 bytes.}

procedure receive (     connection : integer;
                     var       rbfr : datatype;
                     var       rlen : smallint;
                     var  errorcode : integer    ) ;

const
     head_len = 2;

type

    length_buffer_type = packed array[1..2] of char;

header_len_type = record case integer of
                           0: ( word: smallint );
                            1: ( byte: length_buffer_type);
                   end;
var i, j          : integer;
    dlen          : integer;
    header_len    : header_len_type;
    tempbfr       : datatype;

@COMPUTERTXT = begin   { procedure receive }

i:=0;
errorcode := 0;
while (i < head_len) and (errorcode = 0) do     { get length of message }

     begin
     dlen := head_len - i;
     ipcrecv ( connection, tempbfr, dlen, ,, errorcode );
     if errorcode = 0
        then strmove(dlen, tempbfr, 1, header_len.byte, i+1);
     i := i + dlen;
     end;

if errorcode = 0 then
     begin
     rlen := header_len.word;
     i := 0;
     while (i < rlen) and (errorcode = 0) do     { get the message }
        begin
        dlen := header_len.word - i;
        ipcrecv
( connection, tempbfr, dlen, , , errorcode );
        if errorcode = 0
          then strmove(dlen, tempbfr, 1, rbfr, i+1);
        i := i + dlen;
        end;

     end
```

```
else
     rlen := 0;
end;    { procedure receive }
begin  {creator}
{create call socket, then name it}
ipccreate ( 3, 4, , , calldesc,  result);    {call socket, TCP protocol}
check (result);      {error procedure}


socketname := 'MYSOCKET';
ipcname (calldesc,  socketname, 8, result);
check (result);
{place rpmstring with socket information in opt parameter}
prompt('What is the name of your local node? ');
readln(datastr);
len := strlen(datastr);
strmove(len,datastr,1,nodename,1);
fginitopt(opt, 2);      {2 option entries}
addopt(opt,0,20000,8,socketname); {option entry 0, rpmstring option code}
addopt(opt,1,20000,len,nodename); {option entry 1, rpmstring option code}
{create remote process and remote session;
the program file CREATURE must exist in the logon group for VPRES.ACCNTG on the
remote node}
progname := 'CREATURE';
location := 'REMNODE';
login := 'VPRES.ACCNTG,PUB';
for i := 0 to 30 do flags [i] := false;   {false=0, true=1
                                           for each bit in array}
flags [31] := true;     {set dependent flag}
rpmcreate (progname,8,location,7,login,14, , ,flags,opt,pd,result);
if result < > 0 then
     begin
     writeln('RPM ERROR # is ',result);
     terminate;
    end;

{wait for connection request from remote process}
ipcrecvcn (calldesc,  vcdesc, , , result);

check (result);
ipcshutdown (calldesc);
{receive messages on connection and print them;
repeat until 'END' message received}
repeat
     begin
     receive (vcdesc, data, len, result);
     check (result);
     setstrlen(datastr, len);
     strmove(len, data, 1, datastr, 1);
    if datastr < > 'END' then writeln (datastr); {print data received}
     end

until datastr= 'END';
writeln('END received');

data := 'Y';                              {shutdown procedure}
ipcsend ( vcdesc, data, 1, , , result );
check (result);
receive ( vcdesc, data, len, result );
if result = 64 then ipcshutdown( vcdesc )
               else check ( result );
end.  {creator}
```

RPM Program 2

```
$ standard_level 'HP3000', uslinit$
program creature(datafile);       {"datafile" must be an already
                                    existing file}
const
     maxdata    = 2000;
     maxmsg     = maxdata + 2;
     maxname    = 20;
     maxloc     = 20;

type
     byte       = 0..255;
     shortint   = -32768..32767;
     datatype   =
         record

len : shortint;
        msg : packed array[1..maxdata] of char;
        end;
     nametype = packed array[1..maxname] of char;
     loctype  = packed array[1..maxloc] of char;

var
     datafile      : text;
     rpmstringlen : integer;
     socketname   : packed array [1..16] of char;
     nodename     : packed array [1..16] of char;
     destdesc     : integer;
     calldesc           : integer;
     vcdesc           : integer;
     result       : integer;
     data         : datatype;
     strdata      : string[maxdata];
     socknmlen  : integer;
     nodenmlen  : integer;
     y_data       : char;
     y_len        : integer;

procedure terminate; intrinsic;

{RPM and IPC intrinsic declarations}
procedure rpmgetstring; intrinsic;
procedure ipcerrmsg; intrinsic;
procedure ipclookup; intrinsic;
procedure ipccreate; intrinsic;
procedure ipcconnect; intrinsic;


procedure ipcrecv; intrinsic;
procedure ipcsend; intrinsic;
procedure ipcshutdown; intrinsic;

{error handling procedure}
procedure check (result : integer);
{error procedure}
begin
if result < > 0 then
     terminate;    {failed}
end;
```

```
begin  {creature}
{get the creator process's socket name and node name, one at a time, from
 the rpmstrings}
rpmstringlen:= maxname;
rpmgetstring (socketname, rpmstringlen, result);
socknmlen := rpmstringlen;
strmove (socknmlen, socketname, 1, strdata, 1);
rpmstringlen := maxloc;
rpmgetstring (nodename, rpmstringlen, result);
nodenmlen := rpmstringlen;
setstrlen(strdata,0);
strmove(nodenmlen, nodename, 1, strdata, 1);
{look up socket name to get destination descriptor}
ipclookup (socketname,socknmlen,nodename,nodenmlen,,destdesc, , ,result);

check (result);    {error procedure}
{create a call socket for this process}
ipccreate (3, 4, , , calldesc, result);   {call socket, TCP protocol}
check (result);
{send a connection request to the creator process and receive the creator's
reply to complete the connection}

ipcconnect (calldesc,  destdesc, , , vcdesc, result);
check (result);
ipcrecv (vcdesc, , , , ,result);
check (result);
ipcshutdown (alldesc);
ipcshutdown (destdesc);
{send messages on connection}
{ prompt for messages and send them }

reset(datafile);                          {open input file}

while not EOF(datafile) do
     begin
     setstrlen(strdata, 0);
     readln(datafile, strdata);                    {read message}
     data.len := strlen(strdata);                  {store message length}
     strmove(data.len, strdata, 1, data.msg, 1);   {store message}
     ipcsend(vcdesc, data, data.len+2, , , result); {send message,
                                                       including length
                                                         as first 2 bytes}

check(result);     {failed}
end;
{connection shutdown procedure}

data.len := 3;
data.msg := 'END';                       {termination request}
ipcsend(vcdesc, data, 5, , , result);
check(result);
y_len := 1;
ipcrecv (vcdesc, y_data, y_len, , , result);
if (y_data = 'Y') and (result = 0)
   then ipcshutdown(vcdesc)
   else
   check(result);

end.  {creature}
```

8 NetCI

NetCI is a network command interpreter that enables you to more efficiently manage and operate the systems in a network. It allows you to execute MPE commands (or UDCs) and to run programs on any node from one location. It lets you automatically establish remote sessions on several nodes and then broadcast commands to selected MPE V and MPE/iX nodes.

NetCI also allows you to redirect input and output, sequentially execute commands on multiple nodes, and gather data from multiple nodes.

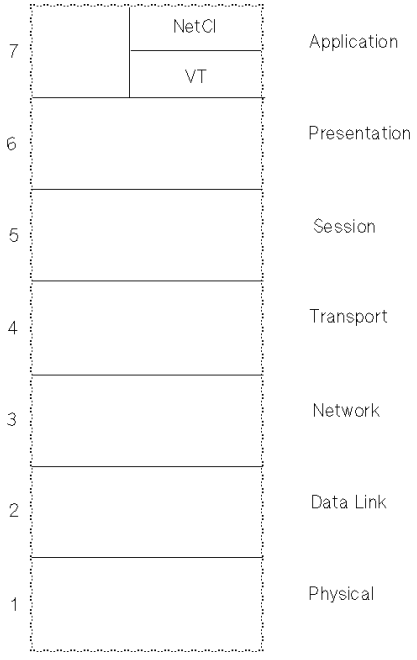
NetCI is an enhanced capability of the NS 3000/iX network services. On MPE/iX based machines, NetCI requires the MPE/iX operating system version A.30.00 or later.

NetCI is a special application program which utilizes Virtual Terminal (VT) service. Figure 8-1 shows how the VT service corresponds to the application layer and how NetCI is logically located on top of VT. NetCI uses VT to establish a connection with each of the remote nodes. This VT connection allows you to execute MPE commands, UDCs, or user programs on the remote nodes.

You only need to install NetCI on one MPE/iX or MPE V node (called the management node). NetCI does not need to be installed on remote nodes of a network. Remote nodes can be either MPE V or MPE/iX nodes.

NetCI can establish remote sessions and can broadcast commands to other networks in an internetwork (network with gateways).

Figure 8-1 NetCI and OSI Model



How to Use NetCI

There are many applications in which you can use NetCI to help access multiple HP 3000s on the network. Some of the more typical applications include:

- gathering similar data from all systems such as status information about jobs/sessions/scheduled jobs;
- performing system check procedures;
- monitoring applications on remote nodes and verifying that all the necessary processes are operational;
- executing global commands such as running application status programs on systems supporting that application.

Refer to “Sample Applications” at the end of this section for descriptions of specific applications that you can develop using NetCI.

NetCI is not a tool for troubleshooting the network. It simply allows you to reduce troubleshooting time and effort by enabling you to quickly isolate some network problems from one terminal.

Running NetCI

To run NetCI from the MPE prompt, enter the following:

```
:RUN NETCI.PUB.SYS
```

Now press **[Return]**.

After MPE accepts the **RUN** command, NetCI displays a message similar to the following message:

```
NetCI/iX A.00.00 (c)Copyright Hewlett-Packard Co. 1994
```

NetCI then displays the NetCI prompt. This prompt tells you that you are in NetCI. You can now execute MPE commands and UDCs, and run programs on any node. Before you do so, you must configure the nodes and logon information. This allows you to access systems through NetCI without having to manually establish session logons for each node.

To exit NetCI, enter the following:

```
NetCI>EXIT (or E)
```

which returns you to the MPE prompt.

Commands

You may enter any NetCI command at the NetCI> prompt. However, before you execute any MPE command, you must specify the name of the node and then the MPE command.

To execute a NetCI command, enter

```
NetCI>SHOWLIST
```

To execute an MPE command, enter

```
NetCI>K SHOWJOB
```

which displays the status information for all jobs/ sessions/scheduled jobs on node or list K.

NetCI HELP Command

At the NetCI> prompt, entering HELP provides a quick reference to all the NetCI commands and a brief description of each command's syntax.

NetCI Security

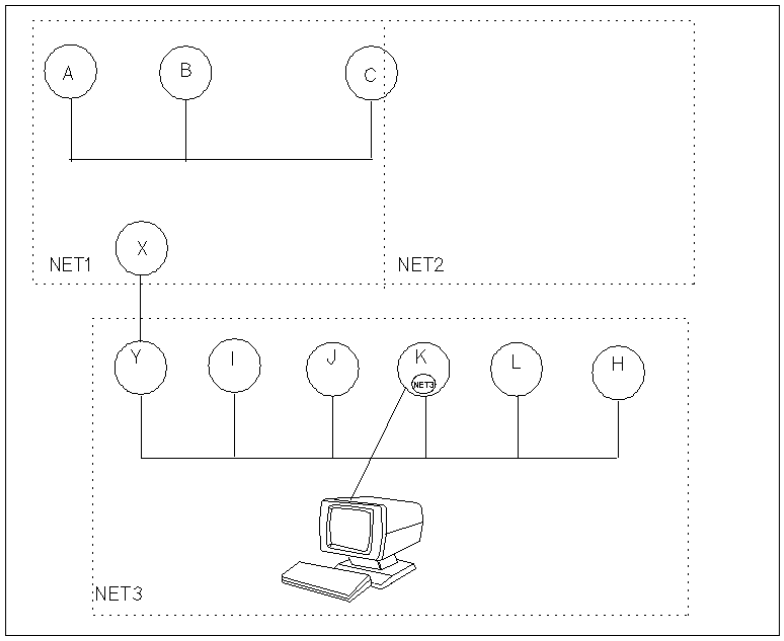
The same file security is provided for the NetCI configuration file as other MPE files. The HP 3000 account structure provides security at the account, group, and file levels. You can alter file level security so that only certain types of access are allowed to the configuration file. NetCI provides further security in the configuration file by encrypting all passwords that are part of the logon information to a node. Passwords are never displayed.

To prevent unauthorized persons from running NetCI or accessing the configuration file, do not leave NetCI in operational mode on your screen.

Configuring Network Data

Figure 8-2 shows an internetwork that includes three networks (NET1, NET2, and NET3) connected together. We will use NET3 as the sample LAN network for our discussion. NET3 shows six nodes connected together by a LAN. NetCI is installed on node K, which is the management node.

Figure 8-2 Sample Internetwork



Configuring Logon Information

NetCI establishes multiple remote sessions on one or more nodes when executing a particular command or running a program. Before remote logon sessions can be established, system and logon information for each node must be stored in the NetCI internal data structure.

To configure each node, use the following commands:

- **NEWNODE** to configure each node and its logon sessions.
- **ALTNODE** to change the node's logon information.
- **PURGENODE** to delete a node from configuration

You can then use the **SHOWNODE** command to list all the nodes and logon sessions that you configured, and to check whether you correctly changed a node's logon information or deleted a node. Refer to the commands on the following pages for more specific details.

Node Names

Each node in the network is identified by its NS 3000 node name. Any node may be added to the NetCI configuration and identified in NetCI by a unique NetCI node name. The NetCI node name may be the NS 3000 node name or another NetCI name you want to assign to the node. Each NetCI node name is associated with a logon session on a node. You can have several logon sessions established on one node but each session must have an individual NetCI node name.

Sample LAN

Using our sample LAN, we will configure all the nodes in NET3 from one location, node K.

In order to configure node K, use the **NEWNODE** command:

```
NetCI>NEWNODE K OPERATOR.SYS/NET3K
```

NetCI will now have in its configuration a node called K. The logon session for K will be OPERATOR.SYS/NET3K.

For our example, we will configure the remaining nodes on the LAN in NET3 using their NS 3000 node names:

```
NetCI>NEWNODE Y OPERATOR.SYS/NET3Y
NetCI>NEWNODE I OPERATOR.SYS/NET3I
NetCI>NEWNODE J OPERATOR.SYS/NET3J
NetCI>NEWNODE L OPERATOR.SYS/NET3L
NetCI>NEWNODE H OPERATOR.SYS/NET3H
```

Reassigning Node Names

You can reassign each node a unique NetCI name instead of using the NS 3000 node name. For example, you can assign a session on node Y the NetCI name, BURGUNDY. NetCI will then recognize the logon session (OPERATOR.SYS) on node Y as BURGUNDY. To assign a unique NetCI name to a node and logon session, you use the **NEWNODE** command with the ;dsline= *NS nodename* option. For example, you enter

```
NetCI>NEWNODE BURGUNDY OPERATOR.SYS/NET3Y;DSLINE=Y
```

After configuring your network, you may then assign the nodes with their associated sessions to different lists and assign an identifier to each list. If four nodes are assigned to a list, a session will be initiated on each of the four nodes when the command is executed on each node. Refer to “Configuring for Command Broadcast” for more information.

When you reassign an NS node a NetCI name, you need to only specify the NetCI name instead of the fully-qualified NS node name to perform an operation on the node. For example, to assign a NetCI name, you enter

```
NetCI>NEWNODE PORT OPERATOR.SYS/NET1A,;DSLINE=A.NET1.BND
```

to configure node A with its logon session in NET1 as PORT. If the NS 3000 node name is not a unique node name in the internetwork, the node name following `;dsline=`*NS nodename* option must be a fully-qualified node name, *nodename.domain.organization*. Whenever you want to establish a session on node A in NET1, you simply need to specify PORT instead of the fully-qualified NS 3000 node name.

NEWNODE

Adds a node and its logon information to the user's NetCI configuration.

Syntax

NEWNODE *node logon*[;dsline=*NS nodename*]

Parameters

node NetCI name or identifier for the node to be added. The NetCI name may be a maximum of 15 characters.

logon Node's logon identification which is a valid logon sequence in the form:

username[/userpass].acctname[/acctpass][,groupname] [/grouppass].

For information on additional MPE logon parameters and options, refer to the *MPE/iX Commands Reference Manual*.

NS nodename If the NetCI name (specified in the *node* field) is the same as the NS 3000 node name, leave this field blank. If the NetCI name is not the same as the NS 3000 node name, specify the NS node name configured in the NS 3000 network.

Discussion

Make sure that the NS node name specified in ;dsline=*NS nodename* is a valid NS node name and is entered correctly (not misspelled). If you enter an erroneous node name into the NetCI configuration, NetCI will only discover the invalid node name when it attempts to log on to a target node.

There are special considerations that apply to scripting (refer to "Redirecting NetCI Input and Output" discussed later in this chapter for further details) that you must consider during configuration:

- If a password is required to initiate a logon session, *make sure to include the password*.
- If remote application programs will poll terminals for termtype during logon (which occurs when the application is run as part of a logon UDC), make sure that you specify the termtype option with the logon information. If it is not specified, connection to that remote node on which the application is running will hang.

Examples Example 1

This example adds nodes with logon sessions to the NetCI configuration.

```
NetCI>NEWNODE K OPERATOR.SYS/NET3K
NetCI>NEWNODE Y OPERATOR.SYS/NET3Y
NetCI>NEWNODE I OPERATOR.SYS/NET3I
NetCI>NEWNODE J OPERATOR.SYS/NET3J
NetCI>NEWNODE L OPERATOR.SYS/NET3L
NetCI>NEWNODE H OPERATOR.SYS/NET3H
```

Since the ;dsline= *NS nodename* option is not specified after the logon information, the NetCI name for each node will be the actual NS 3000 node name. In the example, the default environment for each of the above nodes will be:

```
;DSLINE=K
;DSLINE=Y
;DSLINE=I
;DSLINE=J
;DSLINE=L
;DSLINE=H
```

Example 2

This example assigns a NetCI name (or identifier) to a logon session on node Y.

```
NetCI>NEWNODE CHABLIS OPERATOR.SYS/NET3K;DSLINE=Y
```

Y is the NS node name. The node name specified for the ;dsline=*NS nodename* option must be the actual NS 3000 node name specified in the NS 3000 configuration. In NetCI, CHABLIS now refers to a specific session on remote node Y under the user name OPERATOR.

An NS node may be assigned more than one NetCI name and logon session. For example,

```
NetCI>NEWNODE PINOIT MGR.SYS/NET3K;DSLINE=Y
```

assigns node Y another NetCI name and user on that node. In NetCI, PINOIT refers to a session established on remote node Y under the user name MGR. This example shows that two NetCI names, CHABLIS and PINOIT, are assigned to node Y. These two names refer to node Y with different user sessions being established.

ALTNODE

Changes the NetCI node name and logon information in the user's NetCI configuration for an NS 3000 node.

Syntax

`ALTNODE` *node logon* `[;dsline=`*NS nodename*`]`

Parameters

node — NetCI name of the node whose logon information is to be changed.

logon — New logon identification which is a valid logon sequence in the form:

username`[/`*userpass*`].`*acctname*`[/`*acctpass*`][.`*groupname*`/`*grouppass*`];dsline=`*NS nodename*

For information on additional MPE logon parameters and options, refer to the *MPE/iX Commands Reference Manual*.

NS nodename — If the NetCI name (specified in the node field) is the same as the NS 3000 node name, leave this field blank. If the NetCI name is not the same as the NS 3000 node name, specify the NS node name in the NS 3000 network.

Examples

Example 1

This example changes the user of the logon session for node `I`. The remote environment for NetCI node I remains as previously configured, which is node `I` (the default environment), so it is not necessary to enter the same remote environment information again.

```
NetCI>ALTNODE I NETADMIN.SYS/BRIE
```

Example 2

This example changes the user of the logon session for the NetCI node `PINOIT` whose remote environment remains as previously configured which is node `Y` (the default environment). However, it is necessary to enter the *NS nodename* again since it is not the same as the NetCI *node* name.

```
NetCI>ALTNODE PINOIT NETADMIN.SYS/NET3K;DSLINE=Y
```

Example 3

This example changes the remote environment for the NetCI node `CHABLIS` to NS node `L`.

```
NetCI>ALTNODE CHABLIS OPERATOR.SYS/NET3K;DSLINE=L
```

You still need to enter the user logon information even though you are only changing the remote environment. The NetCI node CHABLIS now refers to a session on NS 3000 node L instead of node Y (which was the previous configuration).

PURGENODE

Deletes a node and its logon information from the user's NetCI configuration and from any list of which the node is a member.

Syntax

PURGENODE *node*

Parameters

node NetCI name of the node to be deleted from configuration and from all lists of which that node is a member.

Example This example deletes node J from the data base.

```
NetCI>PURGENODE J
```

SHOWNODE

Shows the node's logon information and the lists of which the node is a member.

Syntax

`SHOWNODE` *node*

Parameters

node NetCI name of the node whose information is to be displayed. If you want to display the information for all nodes in the NetCI configuration, specify

```
SHOWNODE @
```

Discussion

Passwords are *not* displayed for security reasons.

Example

This example shows the logon information and the lists of which node `Y` is a member. The information displayed shows that node `Y` is a member of `LIST1` and no commands or programs are being executed on this node (no session is established).

```
NetCI>SHOWNODE Y
Y               Connection is closed
Login Data: OPERATOR.SYS
Dsline = Y
Node is on lists:
          LIST1
```

Configuring for Command Broadcast

You may group nodes in the network in various combinations. By grouping nodes and assigning each group to a list, you can "broadcast" to all nodes on the list by issuing a command referencing that list name.

To configure nodes for command broadcast, use the following commands:

- **`NEWLIST`** to first create a list.
- **`ALTLIST`** to add nodes to or delete nodes from a list
- **`PURGELIST`** to delete a list and all its nodes.

You can then use the **SHOWLIST** command to 1) display all the nodes belonging to a list, 2) check whether you correctly added a particular node to or deleted a node from a list, 3) verify whether a list is deleted, or 4) display all lists. Refer to the commands on the following pages for more specific details.

When you broadcast or execute a command on a list, you execute the command sequentially on each node on the list. If a node is down, execution will continue to the next node on the list.

Sample LAN

Using our sample LAN (Figure 8-2), we will create for NET3 three lists identified as LIST1, LIST2, and ALLNODES with different nodes assigned to each list. First, we must use the **NEWLIST** command to create and assign a name to each list.

```
NetCI>NEWLIST LIST1
NetCI>NEWLIST LIST2
NetCI>NEWLIST ALLNODES
```

Next we will use the **ALTLIST** command to assign nodes to each list.

```
NetCI>ALTLIST ADD LIST1 K,I,H
NetCI>ALTLIST ADD LIST2 K,L,H
NetCI>ALTLIST ADD ALLNODES K,Y,I,L,H
```

The first list, named LIST1, will have three members, nodes K, I, and H. The second list, named LIST2, will also have three members, nodes K, L, and H. The third list, named ALLNODES, will have five members, nodes K, Y, I, L, and H. We will now use the **SHOWLIST** command to verify that we correctly added or assigned the nodes to each list.

```
NetCI>SHOWLIST @
LIST1
Nodes on list:
         H   I   K
LIST2
Nodes on list:
         H   L   K
ALLNODES
Nodes on list:
        H   L   I   Y   K
```

NEWLIST

Creates a new list in NetCI configuration.

Syntax

```
NEWLIST list
```

Parameters

list Name of the new list to be created in the NetCI configuration. The *list* identifier may be a maximum of fifteen characters.

Discussion

The new list must be created before you can add nodes to the list.

Example

```
NetCI>NEWLIST NET3
```

ALTLIST

Adds nodes to or deletes nodes from a list.

Syntax

```
ALTLIST {ADD} list nodes
        {DEL}
```

Parameters

ADD	Adds a node to a list.
DEL	Deletes a node from a list. If you delete the last node from the list, the list will still exist in the configuration with no node members.
list	Name of the list to which a new node is to be added, or from which an existing member node is to be deleted.
nodes	Name of the nodes to be added to or deleted from the list. More than one node can be specified here separated by spaces or commas.

Discussion

You must create a list before you can add a node to the list. If you add or delete a node to or from a non-existing list, you will get an error message that the list name is an unknown list name.

Examples

This example shows how to add nodes H, L, K, I and Y to a list named NET3 with the parameter ADD. This list now has five members:

```
NetCI>ALTLIST ADD NET3 H,L,K,I,Y
```

This example shows how to delete node Y from NET3. However, nodes H, L, K and I still exist in the NetCI configuration.

```
NetCI>ALTLIST DEL NET3 Y
```

To check whether node Y is deleted from NET3, use the **SHOWLIST** command (which is discussed later in this section). This command shows only nodes I, K, L, and H as members of NET3.

```
NetCI>SHOWLIST NET3

NET3

Nodes on list:

       I   K   L   H
```

PURGELIST

Deletes an existing list from the NetCI configuration.

Syntax

`PURGELIST` *list*

Parameters

list Name of the list to be deleted.

Discussion

This command deletes the name of the list and the configuration specifying which nodes are members of the list.

```
NetCI>PURGELIST LIST2
```

SHOWLIST

Displays the names of the nodes included in a list.

Syntax

SHOWLIST *list*

Parameters

list Name of the list whose member nodes are to be displayed. If you want to display all lists and the nodes on each list, specify

```
NetCI>SHOWLIST @
```

Examples

Example 1

This example displays the nodes belonging to a specific list.

```
NetCI>SHOWLIST LIST1
LIST1
Nodes on list:
         H   I   K
```

This example displays all the lists and the nodes belonging to each list.

Example 2

```
NetCI>SHOWLIST @
LIST1
Nodes on list:
         H   I   K
ALLNODES
Nodes on list:
         H   L   I   Y   K
NET3
Nodes on list:
         I   K   L   H
```

Saving Your NetCI Configuration

Your NetCI configuration is automatically saved in the file called NCICNFG. This file will reside on the default or home group (where you were logged on when you configured NetCI). If you log on to a group in which NetCI's configuration is not located, you will receive a warning message but still be able to run NetCI.

It is recommended that you keep a backup copy of the NetCI configuration file under another file name. If there is a system failure while the configuration file is being saved, some of the data may be lost or corrupted. This would cause NetCI to operate improperly. If this happens,

1. Delete the corrupted configuration file.
2. Rename the backup copy as NCICNFG.

NetCI only recognizes the configuration filename NCICNFG. Ensure that the backup copy of the configuration file is renamed as NCICNFG.

Executing Remote Commands

NetCI automatically establishes a session on a node when a remote command is executed. Simply specify after the NetCI> prompt the NetCI node or list name followed by the NetCI or MPE command. NetCI will establish a session on the NS 3000 node associated with the NetCI node name, or on the nodes belonging to the list. This eliminates your having to log on remotely to each node when executing commands and running programs. For example, when you enter

```
NetCI>PINOIT SHOWJOB
```

NetCI will automatically log you on to NS 3000 node Y as `operator.sys` and execute the **SHOWJOB** command. If you recall from our sample LAN, `PINOIT` is associated with logon session `operator.sys` on NS node Y. This command will display the status information for all jobs/sessions/scheduled jobs on NS node Y.

Node Prompt

After a command is executed on the remote node, the NetCI> prompt automatically changes to the node's name. For example, when you enter

```
NetCI>PINOIT SHOWJOB
```

the status information for all jobs/sessions/scheduled jobs on `PINOIT` is displayed. The prompt then changes to

```
PINOIT>
```

which indicates the node against which you last executed commands or programs.

You can also change the default prompt to any node prompt by entering the NetCI node name after the current prompt. For example, when you enter

```
PINOIT>CHABLIS
```

the prompt changes to

```
CHABLIS>
```

which is now the default prompt.

By specifying a default prompt, you can execute several commands on a node without specifying the node name each time you specify a command. For example, when you enter

```
CHABLIS>SHOWJOB
CHABLIS>SHOWJOB JOB=@JOB
CHABLIS>SHOWJOB JOB=@S
```

it eliminates your having to specify the node each time.

List Prompt

You can also specify a list name against which a script file, command, or program can be executed. After the prompt, enter the list name and command. For example, when you enter

```
CHABLIS>LIST1 SHOWJOB
```

the **SHOWJOB** command sequentially executes on all nodes that are members of LIST1 (which includes nodes H, I, and K). It also eliminates your having to log on and execute the **SHOWJOB** command on each node.

You can also change the default prompt to any list prompt by entering the list name after the current prompt. For example, when you enter

```
LIST1>ALLNODES
```

the prompt changes to

```
ALLNODES>
```

which is now the default prompt.

By specifying a default list prompt, you can execute several commands against the list without specifying the list name each time you specify a command. For example, when you enter

```
ALLNODES>SHOWJOB
ALLNODES>SHOWJOB JOB=@JOB
ALLNODES>SHOWJOB JOB=@S
```

it eliminates your having to specify the list name each time.

NetCI Prompt

You can return to the NetCI prompt by typing a colon (:) or NetCI after a node or list prompt. For example, when you enter

```
ALLNODES>:
```

or

```
ALLNODES>NETCI
```

you will be returned to the NetCI> prompt.

Command Operation Modes

The prompt indicates the command operation mode in which you are presently operating. The command operation mode that is in effect at any one time may be the

- NetCI mode which is indicated by the NetCI> prompt
- MPE mode which is indicated by a node or list name prompt.

NetCI Mode

You are in the NetCI mode when the NetCI> prompt is displayed. Only NetCI commands may be entered in this mode. For example, when you enter

```
NetCI>SHOWNODE @
```

you are executing a NetCI command in NetCI. If you enter an MPE command after the NetCI> prompt, you will receive an error message.

MPE Mode

To execute MPE commands, you should be in the MPE mode by entering the node or list name preceding the MPE command. For example, when you enter

```
NetCI>PINOIT SHOWJOB
```

you will be in the MPE mode in order to execute the MPE **SHOWJOB** command. The PINOIT prompt will now be the default prompt. Whenever the prompt is a node or list name, you know you are in the MPE mode. NetCI assumes all commands entered in the MPE mode are MPE commands. If you want to enter a NetCI command in the MPE mode, a slash (/) must precede the NetCI command.

For example, you enter

```
PINOIT>/SHOWLIST @
```

to execute the NetCI **SHOWLIST** command when you are in the MPE mode. Refer to “Writing and Executing Script Files” discussed later in this section for details on how the special slash character would be especially useful in script files.

If you want to return to the NetCI mode, you must enter a colon or NetCI after the node or list prompt. For example, when you enter

```
PINOIT> :
```

or

```
PINOIT>NetCI
```

you will receive the NetCI> prompt indicating you are in NetCI mode.

Interrupting Processing (Using [BREAK])

On MPE/iX machines, type **[BREAK]** twice to return to the MPE prompt. On MPE V machines, type **[BREAK]** once to return to the MPE prompt. Pressing **[BREAK]** sends a signal to NetCI to interrupt the process. This signal also passes through a “virtual” terminal to the remote process so that both the local and remote processes are suspended. To resume the local process, type RESUME which displays the current NetCI> prompt. After receiving the NetCI> prompt, type RESUME again to resume the process that was suspended on the remote node. The prompt for the suspended process will be displayed. In summary, if you execute a command on a remote node, type RESUME twice to return to where you were when you pressed **[BREAK]**.

Special Considerations When Using DSLINE

NetCI establishes DSLINE connections in quiet mode. Thus, no messages or prompts will be forwarded to your terminal while a connection is being established. If the remote node prompts for information (for example, password or termtype), the terminal or user would not know that a response is expected. No response would be sent, and the remote node connection will hang. Therefore, if a logon UDC runs a program requiring information such as termtype, make sure the termtype is specified with your logon information. If a password is required but not provided, you will receive a logon error message indicating an incorrect password. You should specify a correct password in the logon information.

Failed Connections

A session will be established on a node when you execute a command. However, a session will fail to be established when the link or node is down (not operating properly) or the configuration data is incorrect. NetCI maintains a record of all attempted logon sessions that failed. If you attempt to execute a command again on the failed node before 15 minutes have elapsed, NetCI will not attempt the execution. You must wait 15 minutes after the last failure before NetCI will attempt to execute the command again.

If the failure to establish connection to a node occurs because the configuration data is incorrect (for example, because of incorrect logon information), you can use the **ALTNODE** command to change the node's logon information. Once this change is made, you do not need to wait the required 15 minutes to attempt another command execution. You can execute the command and NetCI will immediately attempt to establish a session on the node and execute the command.

Redirecting NetCI Input and Output

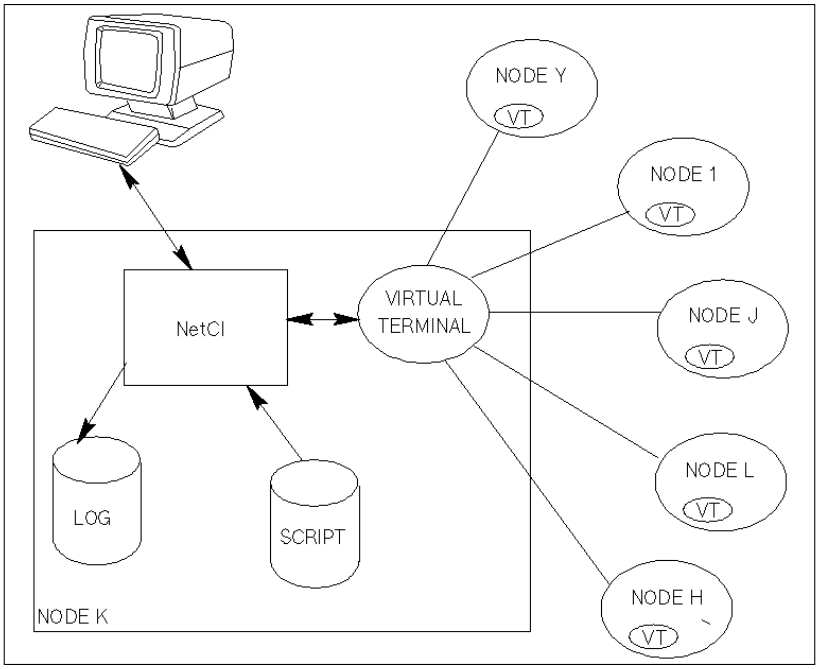
NetCI allows you to easily redirect input and output instead of using $STDIN and $STDLIST. You redirect input through script files and output through log files. Figure 8-3 shows NetCI installed on node K with input and output passing through a virtual terminal configured on the remote nodes. Commands are transmitted over network connections through VT and are executed on the remote nodes. This figure shows that the script file contains the input, while the log file contains the output.

Scripting and logging may be used separately or simultaneously. When you use the scripting and logging operational modes, NetCI redirects the input and output.

The input is redirected from the keyboard to the script file and the output is redirected from the terminal screen to the log file. The different modes of operation are:

- Scripting only (input from script file and output to screen)
- Logging only (input from keyboard and output to log file and screen)
- Scripting and logging (input from script file and output to log file and not to screen)
- No scripting and logging. (interactive session where input is from keyboard and output is to screen)

Figure 8-3 Redirecting Input and Output



Scripting (Redirecting Input)

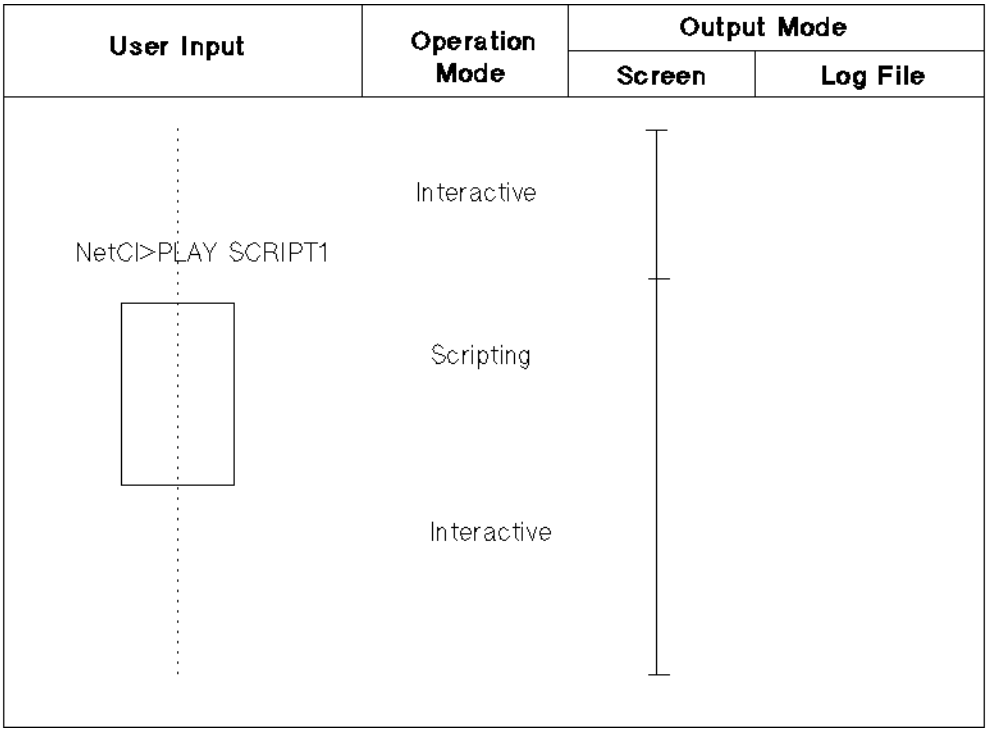
Scripting gives you the capability to store a sequence of commands and data in a file to be used as input into NetCI. You execute the script file with the **PLAY** command to sequentially perform operations instead of issuing a series of commands. Refer to the following pages for more information on the **PLAY** command.

You may execute PLAY in either the NetCI or MPE mode. For this command only, both modes recognize the **PLAY** command and will execute it.

Example

Figure 8-4 shows scripting being activated. Before you execute the script file, input is received from the keyboard, and output is sent to the screen (interactive operational mode). When you execute the script file with PLAY, input is received from the script file, and output continues to be sent to the screen, since it has not been redirected. The straight line under Output Mode indicates when and how long the output is redirected to the screen. Note that the scripting operational mode is in effect whenever you execute a script file.

Figure 8-4 Scripting Activated



PLAY

Executes a block or sequence of commands in a script file.

Syntax

[*list/node*] PLAY [*times*] *file parms*

Parameters

list/node Name of a list of nodes, or the NetCI name of a specific node on which a script file is to be executed. All MPE and NetCI (when preceded by a slash) commands in the script file will be executed on this list or node. If you do not specify a list or node name, only NetCI commands will be executed. Any MPE commands in the script file will not be executed and error messages will display.

times Number of times script file is to be executed. If you do not specify a value, the default is one. The maximum value is 32,767.

file Name of an existing script file with one of the following fully qualified file names:

- file
- `file.group.account`
- file reference (allows you to back-reference a :**FILE** command or to reference a previously defined file)

parms Values or strings passed to the script file. If the script file does not contain any input value or string, leave blank. The maximum number of *parms* that you can specify is 9. The parameter will be used when the script file contains an exclamation mark followed by the parameter position (for example, `!1`, `!2`). The first parameter following the file name is considered to be position 1.

Discussion

The script file must reside on the node on which NetCI is installed before you can execute it. While the script file is executing, input from the keyboard is temporarily deactivated since input is from the file. After script file execution, you can resume with input from the keyboard.

This command cannot be used within a script file.

Examples

Example 1

This example shows the **PLAY** command executing a script file named SCRIPT1 on default node K. The parameter, FINANCE, will be used whenever !1 is encountered in the file since this is the first parameter specified after the file name. Refer to the following page for more information.

```
K>PLAY 2 SCRIPT1 FINANCE
```

The script file will be executed two times, and data will be gathered each time.

Example 2

This example executes the script file twice on all nodes that are members of a list named LIST1.

```
NetCI>LIST1 PLAY 2 SCRIPT1 FINANCE
```

Writing and Executing Script Files

A script file includes all commands, flow control statements, and data that allow you to remotely access and perform operations on nodes. To execute the script file, it must reside on the management node on which NetCI is installed.

Creating a Script File

You can create a script file with any text editor while you are at the MPE colon prompt or from within NetCI. If you are in NetCI, you need to run the editor program from the management node. For example, you enter

```
NetCI>K RUN TDP.PUB.SYS
```

to run the editor on node K which is the management node.

Make sure the editor program resides on the management node, and the user associated with the session is allowed access to the editor. The script file you created will then reside on the management node in the session's user account/group that was specified in NetCI configuration. In this example, the new script file will reside on management node K in the operator.sys account.

Special Symbols

When you create a script file, remember the following:

- Use the symbol % (percentage sign) preceding an MPE or NetCI command. Subsystem command and program input records should not be preceded by %.
- Use the symbol ! (exclamation mark) and a *parms* position number within the script file to indicate the value or string (which is specified with the **PLAY** command) to be used.

In the **PLAY** command, the first parameter specified after the file name is considered as position 1. If you specify !1 within your script file, NetCI will substitute the first *parms* value or string for !1. The file name is considered as position 0.

Reserved Characters

There are also reserved characters that you can use in the script file. Whenever the reserved character is encountered, NetCI substitutes it with specific data as shown in Table 8-1. NetCI does not refer to the **PLAY** command for a *parms* value or string. For example, when the script file contains !a, NetCI substitutes the current account name for !a.

Table 8-1 Reserved Characters

Reserved Characters	Substituted String
!u	Current user name
!g	Current group name
!a	Current account name
!h	Home node
!n	Current or default node on which execution is occurring
!!	Indicates an exclamation mark and not a substitution for a *parms* value or string

Special Slash Character

When a script file is executed on a node or list, the MPE mode is in effect. For example, when you enter

```
NetCI>Y PLAY SCRIPT1
```

you are in MPE mode. MPE assumes only MPE commands are executed in MPE mode. If the script file contains NetCI commands, you must precede these commands with a special slash (/) character in order for the commands to be interpreted as NetCI commands. Refer to SCRIPT1 in the following example on the use of the slash character.

Comment Command

A NetCI **COMMENT** command is available to allow you to include comments in a script file (containing NetCI commands) that will execute in NetCI mode. Refer to Example 1 for using the NetCI **COMMENT** command. Since there is an MPE **COMMENT** command, you can also use this command in a script file (containing MPE commands) that will execute on a node or list. When a script file executes on a node/list, you will be in MPE mode. Refer to Example 2 for using the MPE **COMMENT** command in a script file executing in MPE mode.

Example 1

This example executes a script file called SCRIPT1 containing NetCI commands. To execute the script file, you enter

```
NetCI>PLAY SCRIPT1
```

Since you are executing SCRIPT1 in the NetCI mode, you are using the NetCI COMMENT in the script file. The SCRIPT1 script file contains the following commands:

```
%comment show NetCI nodes'
%comment and lists' configu-
%comment ration
```
adds comment that this file will list the NetCI configuration for the NET3 network.

```
%log logfile1
```
redirects output to log file called LOGFILE1. A slash does not need to precede the NetCI command since you will be in a NetCI mode. Refer to the **LOG** command discussed later in this section.

`%shownode @` displays the NetCI configuration for all nodes.

`%showlist @` displays all the lists and the nodes on each list.

`%logreset` resets output back to the screen. Refer to the **LOGRESET** command discussed later in this section.

`Example 2`

This example executes a script file called SCRIPT2 on node Y, causing you to be in MPE mode. Since there is also an MPE **COMMENT** command, you do not need to use the NetCI **COMMENT** command. To execute the script file, you enter

```
NetCI>Y PLAY SCRIPT2 FINANCE
```

The SCRIPT2 script file contains the following commands:

```
%comment run listdir5
%comment program
```
adds comment that this file runs the listdir5 program.

```
%/log logfile2
```
redirects output to a log file called LOGFILE2. A slash must precede the NetCI **LOG** command since you will be in MPE mode. Refer to the **LOG** command discussed later in this section.

```
%run
listdir5.pub.sys
```
executes program.

`listacct !1` lists attributes for the first *parms* value (which is FINANCE) specified in **PLAY** command. This is a subsystem command within the program called listdir5.

```
listgroup @.!1
```
lists attributes for all groups in FINANCE. This is a subsystem command within listdir5.

`listuser @.!1` lists attributes for all users in FINANCE. This is a subsystem command within listdir5.

`exit`	exits program. This is a subsystem command to exit the `listdir5` program.
`%/logreset`	resets output back to the screen. Refer to the **LOGRESET** command discussed later in this section.

`SCRIPT2` runs a program that lists attributes according to groups and users for an account called `FINANCE`. We know this information is for the `FINANCE` account because `!1` references the first *parms* value specified in `PLAY`. The output will be stored in the log file called `LOGFILE2`.

`Example`

This example shows how the slash must precede the NetCI **LOG**, **LOGRESET**, **LET**, **WHILE**, **INC**, and **ENDWHILE** commands since the MPE mode will be established when you execute the script file on a node or list. For example, when you enter

```
NetCI>I PLAY SCRIPT3
```

you will be in MPE mode. A slash must precede each NetCI command as shown in `SCRIPT3`. The `SCRIPT3` script file contains the following:

```
%/LOG LOGFILE2
%/LET V=1
%/WHILE V 2
%SHOWJOB
%/INC V
%/ENDWHILE
%/LOGRESET
```

Special Considerations

There are several considerations that apply to scripting which must be considered since input is redirected from the script file instead of the terminal. These considerations are:

- Include a password with the logon information during configuration. If you do not include a password, NetCI will automatically assign a password to prevent the system from prompting you for a password.
- Include the termtype option with the logon information during configuration to prevent a remote application from polling the terminal for termtype during connection establishment. This will occur if the application was part of a logon UDC (such as setting function keys). If you do not include the termtype option, and the remote application requires it, the remote node connection will hang.

Flow Control Statements

Flow control statements may be used in script files to control execution of NetCI commands. Refer to the following pages for more details of the flow control statements.

IF Statement

The IF statement controls the execution of a block of commands or a single command depending on whether the expression (or condition) is true.

The IF statement consists of the reserved word IF, an expression (condition), commands, the reserved word **ELSE** and other commands which are optional, and the reserved word ENDIF.

When NetCI executes an IF statement, the following occurs:

1. NetCI evaluates the expression which is the condition.
2. If the condition is true and ELSE is specified, it executes the subsequent commands that follow the condition and skips the commands after ELSE to statements or commands after ENDIF. If the condition is true and ELSE is not specified, it executes the subsequent commands that follow the condition.
3. If the condition is false and ELSE is specified, it executes the commands after ELSE. If the condition is false and ELSE is not specified, flow control skips to statements or commands after ENDIF.

Syntax

```
IF expression
commands
[ELSE commands]
ENDIF
```

Parameters

expression — Specifies the condition that determines whether the commands following it will execute. The expression must be in the following format:

```
identifier operator identifier
```

The `identifier` is a variable name, numerical value, or known variable (set flag) in NetCI. The `operator` is an equal sign (=), not equal sign (< >), greater than sign (>), or less than (<) sign.

commands — Specifies the commands to be executed provided the `expression` (condition) is true.

commands — Specifies the commands following ELSE to be executed provided the `expression` (condition) is not true. The ELSE statement and commands are optional.

Discussion

You may nest both `IF` and `WHILE` statements in script files. A maximum of 20 `IF` statements and 20 `WHILE` statements (for a total of 40 statements) may be nested together within the same script file.

INC Statement

The INC statement increases the value of a variable by one.

Syntax

INC *variable*

Parameters

variable Specifies the variable to which the value is assigned. This variable must begin with an alpha character.

Discussion

If the value of the variable is 32,767, an error message will display.

Example

```
%/LOG LOGFILE2
%/LET V=1
%/WHILE V <2
%SHOWJOB
%/INC V
%/ENDWHILE
%/LOGRESET
```

LET Statement

The LET statement assigns a value to a variable, or a variable to a variable.

The LET statement consists of the equal sign which is an assignment operator. It does not indicate equality but is a signal that the value or variable on the right of the equal sign be assigned to the variable on the left.

Syntax

LET *variable* = *value*

Parameters

variable	Specifies the variable to which the value is assigned. The variable must begin with an alpha character and cannot be numeric, greater than 15 characters, or a node or list name.
value	Specifies a constant between -32,768 and 32,767 or a variable identifier.

Discussion

This statement can be entered interactively or specified in a script file.

WAIT Statement

The WAIT statement returns control to the next command after waiting the specified number of seconds.

Syntax

WAIT *seconds*

Parameters

seconds Specifies the number of seconds before the next command is executed. The number must be a positive integer not greater than 32,767.

Discussion

If you want to wait longer than the maximum number of seconds allowed, specify the WAIT statement as many times as needed.

WHILE Statement

The WHILE statement executes commands repeatedly as long as a given expression is true.

The WHILE statement consists of the reserved word WHILE, an expression (condition), commands, and the reserved word ENDWHILE.

When NetCI executes a WHILE statement, the following occurs:

1. NetCI evaluates the expression which is the condition.
2. If the condition is true, it executes the subsequent commands in the script file until ENDWHILE is encountered, and then re-evaluates the condition. When the condition becomes false, execution resumes at the next statement or command after ENDWHILE.
3. If the condition is false, the subsequent commands following this condition will not execute, and flow control skips to statements or commands after ENDWHILE.

Syntax

```
WHILE expression
commands
ENDWHILE
```

Parameters

expression — Specifies the condition that determines whether the commands following it will execute. The expression must be in the following format:

```
identifier operator identifier
```

The identifier is a variable name, numerical value, or known variable (set flag) in NetCI. The operator is an equal sign (=), not equal sign (< >), greater than sign (>), or less than (<) sign.

commands — Specifies the commands to be executed provided the *expression* (condition) is true.

Discussion

The INC statement may be used with the WHILE statement to increase the value of the expression specified with WHILE. Refer to the INC statement which was previously discussed in this section.

Logging (Redirecting Output)

Logging allows you to store in a log file all output from a process or operation that takes place on configured nodes. You can redirect output to a log file with the **LOG** command and direct output solely to the screen with the **LOGRESET** command. These two commands may be used either inside or outside a script file. Refer to the pages that follow for more information on these two commands. When you are in logging only mode, output will be directed to the log file and to the screen. This capability enables you to respond with input from the keyboard since the scripting mode is not activated.

Accessing Log File

The log file containing the output will reside on the management node (where NetCI is installed) in the *user.account* configured for this node. For example, when you enter

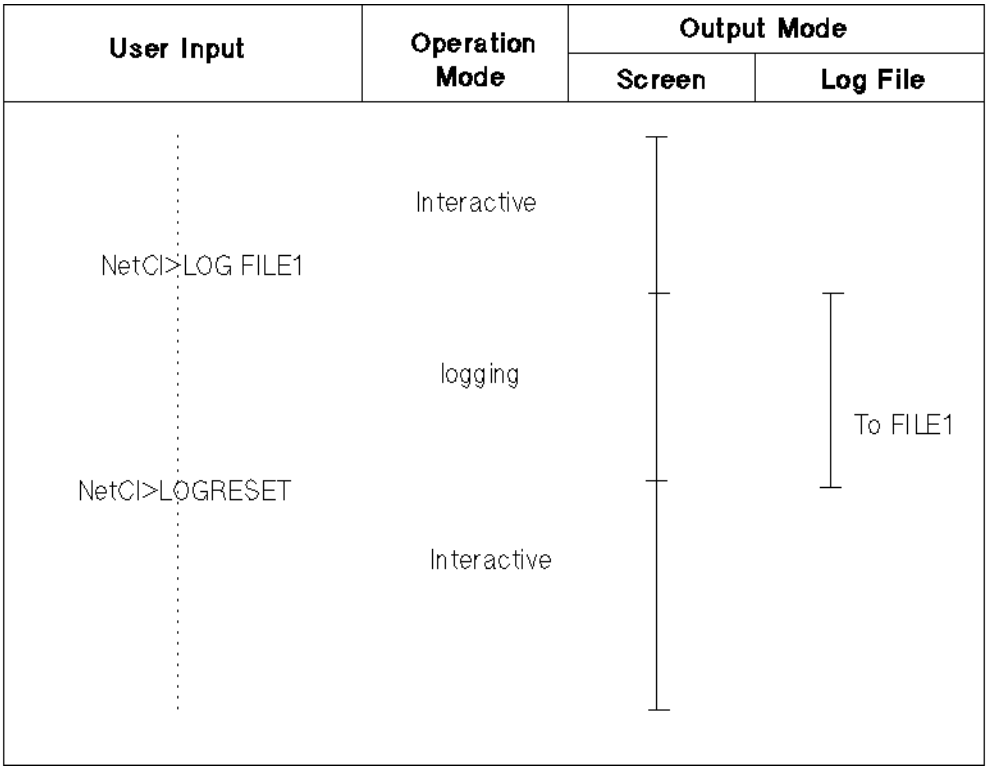
```
NetCI>Y PLAY SCRIPT1 FINANCE
```

NetCI executes a script file called `SCRIPT1` on node `Y`. If you recall our sample internetwork, node `K` is the management node. If you also recall, the script file called `SCRIPT1` (refer to "Writing and Executing Script Files") redirects output to the log file called `LOGFILE1`. When you execute `SCRIPT1` on node `Y` with `PLAY`, the output will be stored in `LOGFILE1` on node `K` in the `operator.sys` account.

Example

Figure 8-5 shows logging being activated with input from the keyboard and output to both the screen and to FILE1. The straight line under Output Mode in the example indicates when and how long the output is redirected to the screen and/or to the FILE1. Notice how LOGRESET returns output back to the screen.

Figure 8-5 Logging Activated



LOG

Redirects the output of a process or operation to a log file.

Syntax

LOG *file*

Parameters

file Name of the file where output is to be stored. This file name is one of the following fully qualified file names:

- file
- `file.group.account`
- file reference (allows you to back reference a **`:FILE`** command, to reference a previously defined file, or to reference a device such as a printer)

Discussion

If the specified log file does not exist, the file will be created except for a back-referenced file which must exist or be defined already.

If the specified log file exists, the new data will append to the end of the existing file.

If the specified log file is full, a warning message will display. The default size of the log file is 1024 records. If you will need a log file with a bigger record size, use the **`BUILD`** command to create a bigger file.

If you are in the MPE mode, a special slash (/) character must precede the **`LOG`** command since NetCI assumes only MPE commands are executed in this mode.

Example This example redirects all output to the file called `FILE1`.

```
NetCI>LOG FILE1
    .
    .
```

LOGRESET

Resets NetCI so that output appears only to the screen.

Syntax

```
LOGRESET
```

Example

This example shows how LOG redirects output to a log file called FILE1. LOGRESET then resets the output back to the screen. If output is not reset back to the screen, output will continue to be directed to FILE1.

```
NetCI>LOG FILE1
   .
   .
   .
NetCI>LOGRESET
```

Scripting and Logging

NetCI redirects both input and output when scripting and logging are used simultaneously. When output is redirected, remember the following:

1. During execution of the script file, the output mode specified in the script file is always in effect. If the script file does not specify an output mode, the mode prior to script file execution remains in effect.
2. After execution of the script file, the output mode prior to execution takes effect again.

Input and output determine the operational mode in effect. It is only in the logging operation mode that output is to both the screen and log file. Refer to Table 8-2 to determine the input and output applicable to each mode of operation.

Table 8-2 Operation Modes

Operation Mode	Input		Output	
	Keyboard	Script File	Screen	Log File
Scripting		X	X	
Logging	X		X	X
Scripting & Logging		X		X
Interactive (No Logging and Scripting)	X		X	

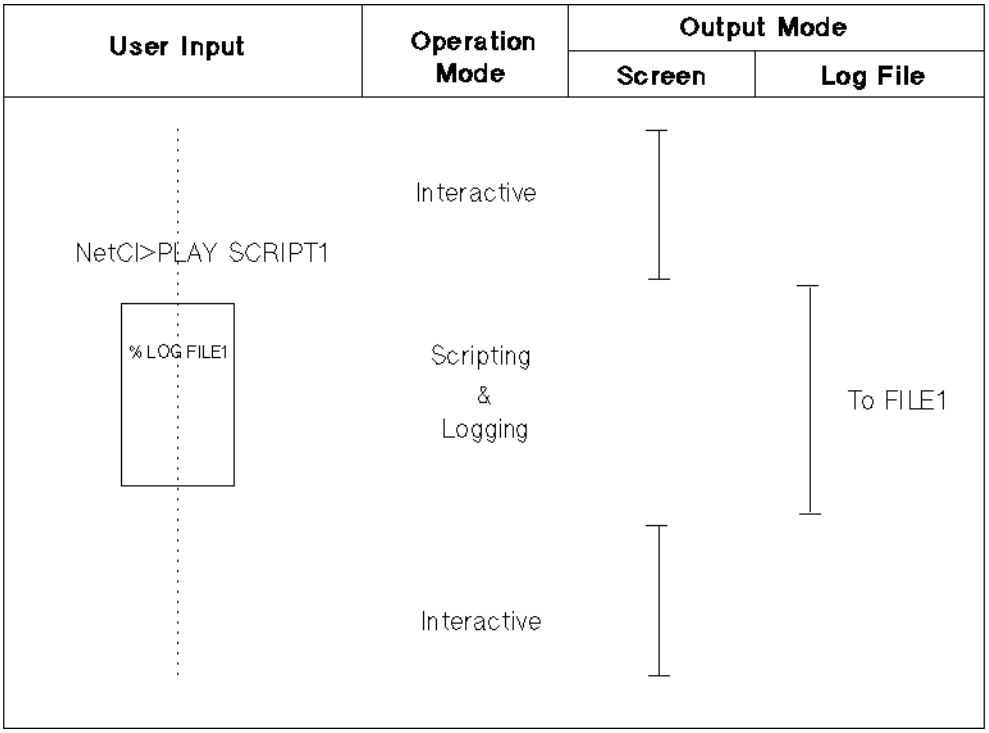
Examples

The following examples begin and end with the interactive operation mode. Each example shows the operation and output mode in effect at a particular time. The output can be redirected to either the screen or a log file, or to both the screen and log file. The straight line under `Output Mode` in each example indicates when and how long the output is redirected to the screen, log file, or both. Notice from the examples that the scripting operation mode is always in effect whenever a script file is executing. However, logging mode may *not* always be in effect.

`Example 1`

Figure 8-6 shows both scripting and logging being activated. Before execution of the script file, output is to the screen. When you execute the script file with PLAY, the script file redirects output to FILE1. After execution, output returns to the screen since this is the output mode prior to execution. You do not need to specify the LOGRESET command.

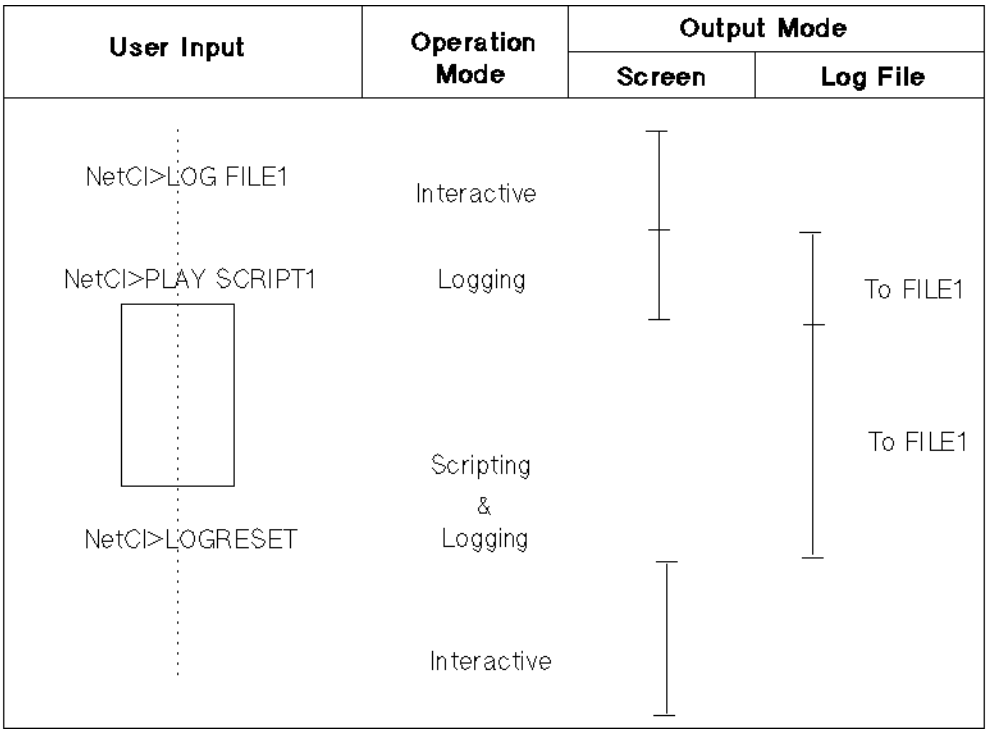
Figure 8-6 Scripting and Logging Activated (Example 1)



Example 2

Figure 8-7 first shows logging being activated by input from the keyboard. Output is to the screen and to a log file called FILE1. Since you may need to interactively respond to the output, it will also be displayed to the screen. Next, you execute a script file called SCRIPT1 with PLAY. While the script file is executing, input is from the script file and output continues to FILE1 since the script file has not redirected the output. The output continues to FILE1 since the user is not required to interactively respond during execution. The LOGRESET command then resets the output to the screen. If LOGRESET is not specified, output will continue to the screen and to FILE1 since the output modes prior to script file execution take effect.

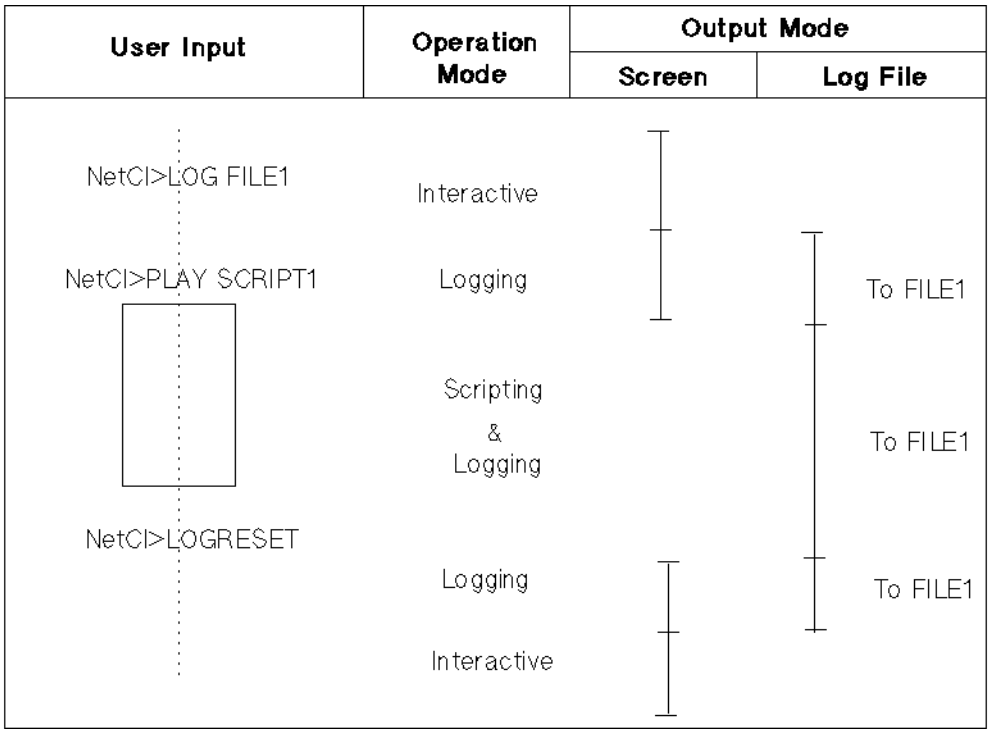
Figure 8-7 Scripting and Logging Activated (Example 2)



Example 3

Figure 8-8 first shows logging being activated by input from the keyboard. Output is to the screen and to a log file called FILE1. Since you may need to interactively respond to the output, it will also be displayed to the screen. Next, you execute a script file called SCRIPT1 with PLAY. While the script file is executing, input is from the script file and output is to FILE1. After script file execution, output returns to the screen and to FILE1 which are the output modes prior to execution. At this point, only logging is in effect. Output continues to the screen and to FILE1 until LOGRESET where output is only to the screen.

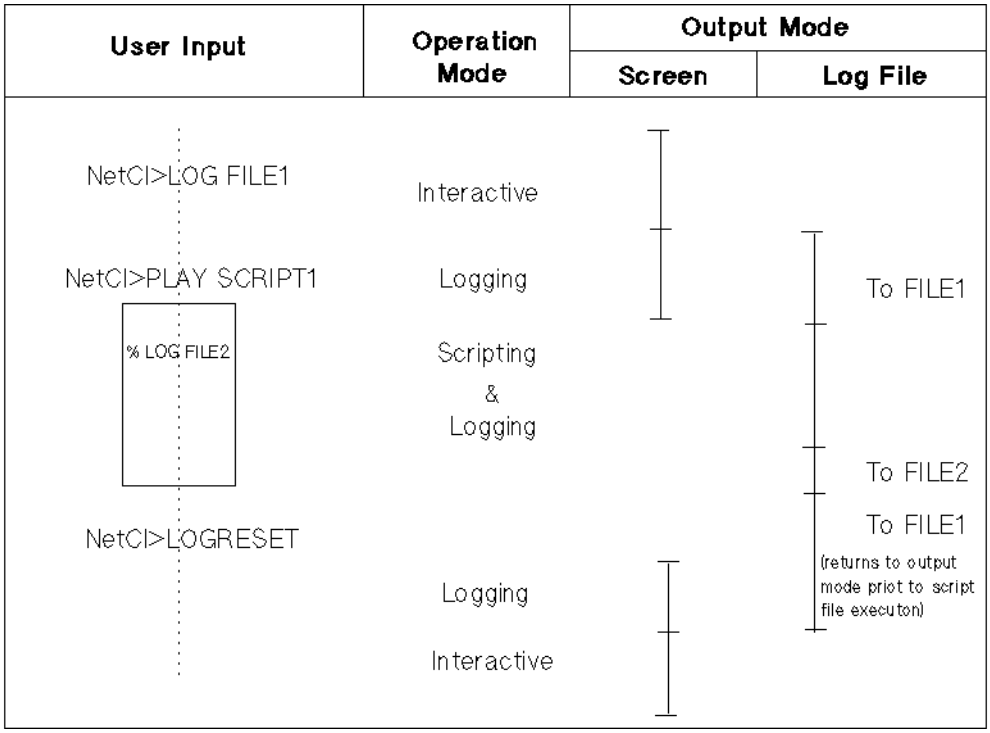
Figure 8-8 Scripting and Logging Activated (Example 3)



Example 4

Figure 8-9 first shows logging being activated by input from the keyboard. Output is to the screen and to a log file called FILE1. Since you may need to interactively respond to the output, it will also be displayed to the screen. Next, you execute a script file called SCRIPT1 with PLAY. While the script file is executing, input is from the script file and output continues to FILE1 until the output is redirected to FILE2 with % LOG FILE2. After script file execution, output returns to the screen and to FILE1 which are the output modes prior to execution. At this point, only logging is in effect. Output continues to the screen and to FILE1 until LOGRESET where output is only to the screen.

Figure 8-9 Scripting and Logging Activated (Example 4)



Sample Applications

The following are sample NetCI applications that you can develop to use in a production environment. You can write a script file containing applicable commands, flow control statements, and data that will automatically perform different operations on multiple remote nodes.

Sample Script File 1

This application summarizes the network configuration for each node in NET3 of our sample internetwork. The following script will display all the entries in the network directory file for the NETXPORT (Network Transport) subsystem. We will write a script file called GETCONF that contains NMMGR commands to be executed on each node in NET3. Logging will also be used to send all output to an MPE log file called ALLCONF. This log file will be interactively entered with the **LOG** command before you execute the script file.

The GETCONF script file contains the following commands:

`%file nmmgrcmd=$stdinx`
reads input from the script file.

`%run nmmgr.pub.sys`
runs the NMMGR configuration program.

`opendir nsdir.net.sys`
opens the network directory file.

`listdir` displays information on all entries for all nodes.

`openconf nsconf.net.sys`
opens the current NS configuration information file.

`versionconf "A.02.03"`
verifies the configuration version.

`pathconf=netxpor t.ni.lan1 protocol.ip` specifies the path for the configuration file to be read.

`readconf` displays the network configuration for the NETXPORT subsystem.

`exit` exits NMMGR.

We are now ready to execute the script file, GETCONF, on all nodes in NET3 of our sample internetwork. If you recall our sample configuration file, node J was deleted from the NetCI configuration. However, we will assume we have added node J back into the configuration. Now we will crate a list called NET3 with the **NEWLIST** command.

```
NetCI>NEWLIST NET3
```

We will then assign all the nodes in NET3 to this list with the **ALTLIST** command.

```
NetCI>ALTLIST ADD NET3 H,L,K,J,I,Y
```

We can verify this by entering the **SHOWLIST** command at the NetCI prompt:

```
NetCI>SHOWLIST NET3
NET3
Nodes on list:
  Y  I  J  K  L  H
```

Before executing the script file, we should create a disc file with a bigger record size so there will be no data overflow. We will use the **BUILD** command to create a disc file called ALLCONF with 10,000 records, each 80 characters long, and ASCII format. Next, we will run NetCI and redirect all output to be saved in this file with the **LOG** command.

```
:BUILD ALLCONF;REC=- 80,100,F,ASCII;DISC=10000
:RUN NETCI.PUB.SYS
```

To execute the GETCONF script file, enter

```
NetCI>LOG ALLCONF
NetCI>NET3 PLAY GETCONF
ALLNODES>/LOGRESET
ALLNODES>/EXIT
```

The script file will be executed sequentially on all the nodes in NET3. After the script file is executed, we can then print the logfile, ALLCONF, to view the network directory and configuration file information for all the nodes in NET3.

Since it is likely that the script file will take awhile to execute, you may want to create a job stream to execute it.

Sample Script File 2

This application shows how to install a new version of software and to convert existing files for use with this new software. We will write a script file, INSTALL, that copies the file, PROG, and the conversion program, CONVERT, to a new group, NEWVERS, at each node in NET3. We will then execute the CONVERT program to convert the file (PROG.NEWVERS). After the conversion, we will purge the old file in the APPLIC group (PROG.APPLIC) and rename the new converted file to the same file name and group as the old file. You can use NetCI to perform this task using one script file, INSTALL, and the **PLAY** global command.

The INSTALL script file contains the following commands:

`%dsline` `local=!1`	file equation for the node defined as local. The referenced node will be specified with the **PLAY** command.
`%newgroup` `newvers`	creates a new group called `newvers` on the node.
`%dscopy`	runs the DSCOPY subsystem.

`prog.newvers:local`
`[operator.sys]`
`to @.NEWVERS;REP`
	specifies that the target file will have the same file name as the source file. REP option replaces the existing file in the destination node with the new file if a file with the same name exists.

`convert.newvers:local`
`[operator.sys] to`
`@.NEWVERS;REP`
	specifies that the target file will have the same file name as the source file. REP option replaces the existing file in the destination node with the new file if a file with the same name exists.

`//`	exits DSCOPY.
`%run` `convert.newvers`	executes the CONVERT program to convert the PROG file.
`%purge convert.newvers`	purges CONVERT program file on target node.
`%purge prog.applic`	purges old existing file in the APPLIC group on target node.

`%rename`
`prog.newvers,pro`
`g. applic`
	renames converted file on target node to the same file and group name as the old file.

`%purgegroup` `newvers`	purges the `newvers` group on the target node.
`yes`	response to the question verifying that the newvers group is to be purged.

```
%run
prog.applic;info
= “showversion”
```
checks that the converted file has been converted for use with the new version of software.

```
%tellop...**applic:
version a.01.01
installed & ready**
```
displays a message that the current software version has been successfully installed and is ready for use.

We are now ready to execute the script file, INSTALL, on all nodes in NET3 of our sample internetwork. If you recall from Sample Script File 1, all nodes in NET3 are members of the list called NET3.

To execute the INSTALL script file, enter

```
NetCI>NET3 PLAY INSTALL H
```

If you specify the output to a log file, you can later scan the file for errors. Additional MPE statements can be added to provide for more robust error checking such as the handling of unexpected errors. For example, if someone is running the application, the **:PURGE** command will fail. We can use MPE commands such as **:IF** or **:ELSE** to also modify the execution of the script file.

Sample Script File 3

This application shows how you can create a script file containing a job stream for the nodes in NET3 of our sample internetwork. The following script will contain the **:STREAM** job command for the different nodes. When you execute the script file called SYSCONF, this will stream a job file called CONFJOB which will then execute the SYSINFO program (which lists the configuration information for the node).

All the commands in the CONFJOB job file may also be included in a script file. However, we will assume you already have an existing job file that contains these commands. You only need to create a script file to stream the job file. By streaming an existing job file within a script file, you do not need to type again all the commands from the job file into the script file, and you can always run the job file in MPE.

The SYSCONF script file contains the following stream command:

```
%stream
confjob.util.sys
```
streams a job on a node.

Although the script file contains only one command line, it is sometimes easier to enter PLAY SYSCONF than entering the entire command line.

The CONFJOB job file contains the following commands and data:

```
!job net3,operator.sys/net3k;hipri;pri=cs;outclass=epoc,13
!comment this will print the system configuration report
!file conflist=sysinfo;dev=pp;env=elite.hpenv.sys
!run sysinfo.prv.telesup
out pp
title "NET3"
all
exit
!eoj
```

The job file, CONFJOB, will run a program on the node or list that you specify with the **PLAY** command. This program called SYSINFO will generate a report listing the node's configuration. The configuration includes the devices on the system, other system information such as MPE tables, program stack size usages, directory usage, system logging, user logging, spooling, virtual memory, GIC hardware configuration summary, data communication device summary, and so forth.

To execute this script file against all the nodes in NET3, we will execute the SYSCONF script file on the previously configured list named NET3. To execute this script file, enter

```
NetCI>NET3 PLAY SYSCONF
```

This will stream the job for each node in NET3. When the job is completed for nodes Y, I, J, K, L, and H, the configuration listing will be sent to the printer.

Example

The following example shows how you can also create a script file which can be executed on each node/list instead of a script file containing a job stream. We will write a script file called NODCONF that will execute the SYSINFO program on the node that you specify with the **PLAY** command. Logging will be used to send all output to a log file called REPORT.

The NODCONF script file contains the following commands:

`%/log report` redirects output to logfile called REPORT.

`%run sysinfo.prv.telesup`
executes program.

`title "NET3"` information required by program.

`all` information required by program.

`exit` exits program.

`%/logreset` resets output back to the screen.

We are now ready to execute the script file, NODCONF, on any node. For example, to execute the script file on node L, enter

```
NetCI>L PLAY NODCONF
```

which will generate the system configuration listing for node L.

To execute the script file on all nodes in NET3, enter

```
NetCI>NET3 PLAY NODCONF
```

which will generate the system configuration listing for NET3. The listings for node L and NET3 will be in the REPORT log file residing on node K which is the management node.

Sample Script File 4

This application shows how you can create a script file called SYSOP containing operations or jobs to be run at a particular time. The following script file shows how you can effectively use flow control commands to control the execution of these operations or jobs to be run at particular times.

The SYSOP script file contains the following commands:

```
%/while hour < 24

%    /comment The following operations are done between 8am and 5pm.
%    /if hour > 7
%        /if hour < 18
%            showjob job=@j
%        /else
%            /comment This operation is done only at 8pm.
%            /if hour = 20
%                showjob job=@s
%            /endif
%            /comment At 10pm the following operation must be done.
%            /if hour = 22
%                listf @ipc@.maildb.hpoffice,2
%            /endif
%        /endif
%    /endif
%    /inc hour
%    /comment Replace v in the following statement with the remaining
%    /comment number of seconds until the next approximate hour.
%    /wait v
```

```
%/endwhile
```

We are now ready to execute the script file, SYSOP, on any node. Before we execute the script file, we must interactively assign a value (the current time) to the variable called “hour”. To set the current time, enter

```
NetCI>LET HOUR=X (X is current hour, for example, 6 for 6 A.M.)
```

After specifying the current time, we are now ready to execute the SYSOP script file. To execute this script file, enter

```
NETCI>NET3 PLAY SYSOP
```

to run particular operations at specified times on all the nodes that are members of the NET3 list.

If you are executing the script file on a list, you should change the wait duration to a value that will allow the file enough time to execute on all nodes during each hour. For example, if it takes approximately 1200 seconds to execute the script file on all nodes that are members of the NET3 list, you could assign 2400 seconds as the wait value. This would be enough time for execution to have occurred on all the nodes. The script file will take about 1200 seconds to execute and then wait 2400 seconds to execute again. This will be approximately the next hour.

Troubleshooting NetCI

The first step in troubleshooting NetCI is to look at the error message returned by NetCI. In order to understand the error message, refer to the "NetCI Error Messages" section in the *NS 3000/iX Error Message Reference Manual* for the possible causes of the error and recommended recovery actions). If NetCI returns a specific error message, find it in the manual, try to understand the cause of the error, and take the action recommended. If the error is not clear to you or the recommended action does not correct the problem, call your HP representative.

A Migration From NS 3000/V to NS 3000/iX Network Services

This Appendix discusses the migration from NS 3000/V to NS 3000/iX network services. If your system currently uses DS/3000, please consult Appendix B, “Migration From DS/3000 to NS 3000/iX Network Services.”

For network management (configuration) migration, refer to the NS 3000/iX Configuration Planning and Design Guide. For more information on migration in general, and for migration from MPE V to MPE/iX in particular, refer to the HP Migration Overview manual in the Migration Series of manuals.

Assistance from Hewlett-Packard is available if you want help installing your NS 3000/iX software, configuring your nodes and system, and verifying the operation of your software.

Differences Between NS 3000/V and NS 3000/iX

NS 3000/iX is a compatible subset of NS 3000/V, the powerful networking software available on the MPE V/E based members of the HP 3000 family.

Special Note on Terminal Echo

When communicating between an MPE/iX based node an MPE V based node, if echo ever gets turned off, proceed as follows:

1. Issue a colon (:) to return to your local node.
2. Restore the echo on your local system according to the method for that node (the method is different depending on whether your physical terminal is connected to an MPE V based node or an MPE/iX based node). Remember: the node that your physical terminal is connected to always determines how you restore your terminal's echo.
3. Note that this procedure restores echo on the local system. To ensure that echo is also restored on the remote system, you may wish to run a program that issues an `FCONTROL 12` on the remote system.

Missing Features

The following list specifies the features that were available in NS 3000/V but are not available in NS 3000/iX:

- Program-to-Program communication (PTOP) is not available. For information on translating your PTOP applications so that they can run on an MPE/iX based machine, please refer to the "Conversion Checklist" section.
- No-wait I/O Remote File Access is not available.

Changed Features

The features that changed from NS 3000/V to NS 3000/iX are as follows:

- When using Network File Transfer (NFT) or Remote File Access (RFA) in NS 3000/iX, only files with MPE V characteristics are supported. For instance, the mapped access file system feature of MPE/iX is not supported by NFT and RFA in NS 3000/iX.
- Remote Data Base Access in NS 3000/iX is supported only for TurboIMAGE databases. Remember that an IMAGE database on a DS/3000 node can be accessed by using an intermediate NS 3000/V node. The **`SHOWCOM`** command has been replaced by the **`LINKCONTROL STATUS`** command.

Error Messages: NS 3000/V to NS 3000/iX

To learn about NS 3000/iX error messages and recovery procedures, refer to the *NS 3000/iX Error Messages Reference Manual*.

Conversion Checklist: NS 3000/V to NS 3000/iX

The following checklist is to help you with migrating NS 3000/V software to NS 3000/iX on MPE/iX:

1. Identify applications using Program-to-Program Communications (PTOP). PTOP is not supported, so any PTOP applications must be modified so that they can run on an MPE/iX based machine. If you wish to convert your PTOP applications so that they can run on RPM and NetIPC, refer to the translation procedures in Appendix B of the *NetIPC 3000/iX Programmer's Reference Manual*
2. To move applications on HP3000s that use the NS 3000/V capabilities of Virtual Terminal, Remote File Access, Remote Data Base Access, Network File Transfer, and Remote Process Management to an MPE/iX based system, use either: a) the **DSCOPY** command, or b) the MPE/iX **STORE** command to store the applications on a tape, and then use the MPE/iX **RESTORE** command to restore them on the new system. They will run with NS 3000/iX with no recompilation necessary in most cases.

B Migration From DS/3000 to NS 3000/iX Network Services

This Appendix discusses the migration from DS/3000 to NS 3000/iX. If your system currently uses NS 3000/V, and you are migrating to NS 3000/iX, refer to Appendix A, “Migration From NS 3000/V to NS 3000/iX Network Services.”

For network management (configuration) migration, refer to the N*S 3000/iX Configuration Planning and Design Guide*. For more information on migration in general, and for migration from MPE V to MPE/iX in particular, refer to the *HP Migration Overview Manual* in the *Migration Series* of manuals.

Assistance from Hewlett-Packard is available if you want help installing your NS 3000/iX software, configuring your nodes and system, and verifying the operation of your software.

Differences Between DS/3000 and NS 3000/iX

DS is an acronym for Distributed Systems.

NS is an acronym for Network Services. An NS 3000/iX link product is required in addition to the NS 3000/iX Network Services.

A DS installation can consist of 1) hardwired point-to-point links, 2) point-to-point modem links on either dial-up or leased telephone lines, 3) X.25 network links, and 4) satellite network links. The maximum data rate over a DS link is 56 kilobits per second.

An NS installation, on the other hand, may be an internetwork consisting of a LAN with a 10 megabit-per-second signalling rate connected to other LANs by way of a wide-area, point-to-point network link. NS 3000/iX cannot directly access a DS link. However, an NS 3000/V node can be used as an intermediary to connect from an NS 3000/iX node to a DS link.

New Features

NS 3000/iX provides the user and programmer with many features that are not available in DS/3000. These new features are summarized here:

- Direct access to any node on the network or internetwork.
- Direct access to remote files, devices, or databases.
- Configurable remote prompts.
- Redirection of node names coded into programs or job streams.
- Programmatic access of remote terminals by using Reverse VT.
- Network File Transfers (NFT) from any node on the network or internetwork to any other node on the network or internetwork.
- Temporary remote logons for NFT transfers.
- `DSCOPY` options to move, replace, or overwrite files and to copy privileged files.
- New and existing (DS) syntax is supported.

Missing Features

This list specifies the features that were available in DS/3000 but are not available in NS 3000/iX:

- The following `DSLINE` parameters are ignored: `LINEBUF=` `EXCLUSIVE`, `PHNUM=, LOCID=`, `REMID=`, `SELECT=`, `FROMADDR`, `FROMADR=`, `TOADDR=`, `TOADR=`, `QUEUE`, `NOQUEUE`.

- 3000 to 1000 or DEXEC calls are not supported.
- Program-to-Program communication (PTOP) is not available. To convert your PTOP applications to NetIPC and RPM, please refer to the *NetIPC 3000/iX Programmer's Reference Manual*.

Changed Features

The features that changed from DS 3000 to NS 3000/iX are as follows:

- `DSLINE` *ldev#* (for DS device) is replaced by `DSLINE` *envID* or `DSLINE` *envnum*.
- Some DSERR error codes changed.
- Remote Process Management is available on NS 3000/iX.

Error Messages: DS/3000 to NS 3000/iX

To learn about NS 3000/iX error messages and recovery procedures, refer to the *NS 3000/iX Error Messages Reference Manual*.

Conversion Checklist: DS/3000 to NS 3000/iX

Identify applications using Program-to-Program Communications (PTOP). PTOP is not supported, so any PTOP applications must be converted so that they can run on an MPE/iX based machine. If you wish to convert your PTOP applications so that they can run on NetIPC and RPM, refer to the conversion procedures in the *NetIPC 3000/iX Programmer's Reference Manual*.

Index