HEWLETT PACKARD

NS-ARPA/1000

Quick Reference Guide

Software Services and Technology Division
11000 Wolfe Road
Cupertino, CA 95014-9804

Manual Part No. 91790-90040
E0495

Printed in U.S.A. April 1995
Sixth Edition

Printing History

The Printing History below identifies the edition of this manual and any updates that are included. Periodically, update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this printing history page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past updates; however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the Manual Numbering File. (The Manual Numbering File is included with your software. It consists of an "M" followed by a five digit product number.)

First Edition	Feb	1986	Rev. 2608
... Update 1	May	1986	Rev. 2626
... Update 2	Oct	1986	Rev. 4.1
... Update 3	Aug	1987	Rev. 5.0
... Update 4	Feb	1988	Rev. 5.05
Second Edition	Oct	1989	Rev. 5.16
... Update 1	May	1990	Rev. 5.2
Third Edition	Aug	1991	Rev. 5.24
Fourth Edition	Dec	1992	Rev. 6.0
Fifth Edition	Nov	1993	Rev. 6.1
Sixth Edition	Apr	1995	Rev. 6.2

Preface

Hewlett-Packard Network Services for the HP 1000 (NS-ARPA/1000) provides the networking software that allows HP computer systems to communicate with each other.

Audience

The *NS-ARPA/1000 Quick Reference Guide* is a condensed version of the following two manuals: *NS-ARPA/1000 User/Programmer Reference Manual* and *NS-ARPA/1000 DS/1000-IV Compatible Services Reference Manual*, which are the primary reference sources for programmers and operators who will be writing or maintaining programs for NS-ARPA/1000 systems. The purpose of this guide is to provide a quick reference for users who are already familiar with the concepts and syntax presented in the above two manuals.

For your convenience, the *NS-ARPA/1000 Quick Reference Guide* also contains a master index of NS-ARPA/1000 manuals. This is a combined index from the NS-ARPA/1000 manuals to help you find information that may be in more than one manual.

Assumptions

Since the services described in this manual are both interactive and programmatic, this manual is intended for interactive users as well as programmers. As one of these interactive users or programmers, you should be familiar with the operating systems on the HP 1000, especially the RTE-A operating system. For those operations that deal with HP 3000 systems, a working knowledge of the Multiprogramming Executive (MPE) is also recommended. For those operations that deal with HP 9000 systems, a working knowledge of the HP-UX operating system is also recommended. Network Managers, who have responsibility for generating and initializing nodes and configuring networks, should consult the *NS-ARPA/1000 Generation and Initialization Manual* and the *NS-ARPA/1000 Maintenance and Principles of Operation Manual*.

Organization

Section 1	**TELNET**—describes the commands, format, parameters, and usage of TELNET. TELNET provides a virtual terminal connection to remote nodes in your network.
Section 2	**File Transfer Protocol**—describes the commands, format, parameters, and usage of the File Transfer Protocol (FTP). FTP allows you to transfer files to and from remote nodes in your network. FTP also provides file management operations such as changing, listing, creating, and deleting remote directories.

Section 3	**HP 1000 File Server**—describes the runstring parameters of the FSRV server program and the program's error messages.
Section 4	**Berkeley Software Distribution Interprocess Communication**— describes a set of programming development tools for interprocess communication, originally developed by the University of California at Berkeley. BSD IPC allows programs on the HP 1000 to communicate with programs on HP and non-HP machines that have BSD IPC 4.3.
Section 5	**Network File Transfer**—describes the commands, format, parameters, and usage of the file copying program DSCOPY. DSCOPY allows you to copy files from one node to another in your network.
Section 6	**Network Interprocess Communication**—describes a set of programmatic calls that provide a data exchange interface between peer processes located at the same or different nodes in your network. Their format, parameters, and usage are explained.
Section 7	**Remote Process Management**—describes a set of programmatic calls that provide remote scheduling, controlling, and terminating of programs located at the same or different HP 1000 nodes in your network.
Section 8	**REMAT**—describes the REMAT commands to send RTE commands or special DS/1000-IV commands to any HP 1000 node in your network.
Section 9	**RMOTE**—describes the RMOTE commands to direct commands to an HP 3000 node.
Section 10	**Remote File Access**—describes the RFA calls to manage remote disk and non-disk files from your programs.
Section 11	**DEXEC**—describes the DEXEC calls to control I/O devices located at remote HP 1000 computers in your network.
Section 12	**Program-to-Program Communication**—describes the programmatic PTOP calls that provide remote scheduling, controlling, and terminating of programs located at HP 1000 nodes in your network.
Section 13	**Remote I/O Mapping**—describes IPMAP to redirect I/O requests destined for an LU on an HP 1000 node to an LU at a remote HP 1000 node on your network.
Section 14	**Maintenance Utilities**—summarizes the maintenance utilities available on NS-ARPA/1000. These include NSINF, NSINIT, NS Message Tracing, NS Event Logging, NSLIST, DS/1000-IV (RTE-MPE) Message Tracing, TRC3K, and DSMOD. This section also lists the utility subroutines provided for use in conjunction with PTOP, RFA, and DEXEC calls.

Guide to NS-ARPA/1000 Manuals

The following are brief descriptions of the manuals included with the NS-ARPA/1000 product.

91790-90020 NS-ARPA/1000 User/Programmer Reference Manual

Describes the user-level services provided by NS-ARPA/1000. The NS services are network file transfer (NFT), network interprocess communication (NetIPC), and remote program management (RPM). The ARPA services are TELNET and FTP. Because there are interactive and programmatic services, this manual is intended for interactive users as well as programmers. It should also be read by Network Managers before designing an NS-ARPA/1000 network so that they will have a clear understanding of the full implications of various NS-ARPA/1000 functions and features.

91790-90030 NS-ARPA/1000 Generation and Initialization Manual

Describes the tasks required to install, generate, and initialize NS-ARPA/1000. This manual is intended for the Network Manager. Before reading this manual, the Network Manager should read the *NS-ARPA/1000 User/Programmer Reference Manual* to gain an understanding of the NS-ARPA/1000 user-level services. The Network Manager should also be familiar with the RTE-A operating system and system generation procedure.

91790-90031 NS-ARPA/1000 Maintenance and Principles of Operation Manual

Describes the NS-ARPA/1000 network maintenance utilities, troubleshooting techniques, and the internal operation of NS-ARPA/1000. The Network Manager should use this manual in conjunction with the *NS-ARPA/1000 Generation and Initialization Manual*. This manual may also be used by advanced users to troubleshoot their applications.

91790-90040 NS-ARPA/1000 Quick Reference Guide

Lists and briefly describes the interactive and programmatic services described in the *NS-ARPA/1000 User/Programmer Reference Manual* and the *NS-ARPA/1000 DS/1000-IV Compatible Services Reference Manual*. The purpose of this guide is to provide a quick reference for users who are already familiar with the concepts and syntax presented in those two manuals.

The *NS-ARPA/1000 Quick Reference Guide* also contains abbreviated syntax for certain programs and utilities described in the *NS-ARPA/1000 Generation and Initialization Manual* and the *NS-ARPA/1000 Maintenance and Principles of Operation Manual*. For your convenience, the *NS-ARPA/1000 Quick Reference Guide* also contains a master index of NS-ARPA/1000 manuals. This is a combined index from the NS-ARPA/1000 manuals to help you find information that may be in more than one manual.

91790-90045 NS-ARPA/1000 Error Message and Recovery Manual

Lists and explains, in tabular form, all of the error codes and messages that can be generated by NS-ARPA/1000. This manual should be consulted by programmers and users who will be writing or maintaining programs for NS-ARPA/1000 systems. Because it contains error messages generated by the NS-ARPA/1000 initialization program NSINIT and other network management programs, it should be consulted by Network Managers.

91790-90050 NS-ARPA/1000 DS/1000-IV Compatible Services Reference Manual

Describes the user-level services provided by the DS/1000-IV backward compatible services. These services are Remote File Access (RFA), DEXEC, REMAT, RMOTE, program-to-program communication (PTOP), utility subroutines, remote I/O mapping, remote system download to memory-based DS/1000-IV nodes only, and remote virtual control panel.

91790-90054 File Server Reference Guide for NS-ARPA/1000 and ARPA/1000

Describes information on using and administering the HP 1000 file server, including runstring parameters, files needed for configuration, troubleshooting guidelines, and error messages.

91790-90060 BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000

Describes Berkeley Software Distribution Interprocess Communication (BSD IPC) on the HP 1000. BSD IPC on the HP 1000 offers a programmatic interface on the HP 1000 for multi-vendor connectivity to systems that offers BSD IPC 4.3.

5958-8523 NS Message Formats Reference Manual

Describes data communication messages and headers passed between computer systems communicating over Distributed System (DS) and Network Services (NS) links.

5958-8563 NS Cross-System NFT Reference Manual

Provides cross-system NFT information. It is a generic manual that is a secondary reference source for programmers and operators who will be using NFT on NS-ARPA/1000, NS3000/V, NS3000/XL, NS/9000, NS for the DEC VAX* computer, and PC (PC NFT on HP OfficeShare Network). Information provided in this manual includes file name and login syntax at all of the systems on which NS NFT is implemented, a brief description of the file systems used by each of these computers, and end-to-end mapping information for each supported source/target configuration.

*DEC and VAX are U.S. registered trademarks of Digital Equipment Corporation.

Conventions Used in this Manual

NOTATION	DESCRIPTION
`nonitalics`	Words in syntax statements that are not in italics must be entered exactly as shown. Punctuation characters other than brackets, braces, and ellipses must also be entered exactly as shown. For example: `EXIT;`
`italics`	Words in syntax statements that are in italics denote a parameter that must be replaced by a user-supplied variable. For example: `CLOSE` *`filename`*
`[ ]`	An element inside brackets in a syntax statement is optional. Several elements stacked inside brackets means the user may select any one or none of these elements. For example: $\begin{bmatrix}\texttt{A}\\\texttt{B}\end{bmatrix}$ User *may* select A or B or neither.
`{ }`	When several elements are stacked within braces in a syntax statement, the user must select one of those elements. For example: $\begin{Bmatrix}\texttt{A}\\\texttt{B}\\\texttt{C}\end{Bmatrix}$ User *must* select A or B or C.
`...`	A horizontal ellipsis in a syntax statement indicates that a previous element may be repeated. For example: `[,`*`itemname`*`]...;` In addition, vertical and horizontal ellipses may be used in examples to indicate that portions of the example have been omitted.

`,`	A shaded delimiter preceding a parameter in a syntax statement indicates that the delimiter *must* be supplied whenever (a) that parameter is included or (b) that parameter is omitted and any *other* parameter that follows is included. For example:
	
	`itema``[,`*`itemb`*`][,`*`itemc`*`]`
	
	means that the following are allowed:
	
	`itema`
	`itema,itemb`
	`itema,itemb,itemc`
	`itema,,itemc`

Δ

When necessary for clarity, the symbol Δ may be used in a syntax statement to indicate a required blank or an exact number of blanks. For example:

```
SET[(modifier)]Δ(variable);
```

$\underline{\text{underlining}}$

When necessary for clarity in an example, user input may be underlined. For example:

$\texttt{NEW NAME? }\underline{\texttt{ALPHA}}$

Brackets, braces, or ellipses appearing in syntax or format statements that must be entered as shown will be underlined. For example:

$\texttt{LET } \textit{var}\texttt{[}\underline{\texttt{[}}\textit{subscript}\underline{\texttt{]}}\texttt{] = } \textit{value}$

Output and input/output parameters are underlined. A notation in the description of each parameter distinguishes input/output from output parameters. For example:

$\texttt{CREATE (}\textit{parm1}\texttt{,}\underline{\textit{parm2}}\texttt{,}\underline{\textit{flags}}\texttt{,}\underline{\textit{error}}\texttt{)}$

$\boxed{}$

The symbol $\boxed{}$ may be used to indicate a key on the terminal's keyboard. For example, $\boxed{\texttt{RETURN}}$ indicates the carriage return key.

$\boxed{\texttt{CONTROL}}$ *char*

Control characters are indicated by $\boxed{\texttt{CONTROL}}$ followed by the character. For example, $\boxed{\texttt{CONTROL}}$Y means the user presses the control key and the character Y simultaneously.

1

TELNET

Table of Contents

Application and Connectivity Considerations

There are several considerations to keep in mind when using TELNET:

- In certain cases, it may take longer to send terminal data from the physical terminal over the network to the remote node than the time allowed by an application program. If the program fails to receive the needed data, it will result in error. User written applications that are expected to run over TELNET should be written with this in mind.
- TELNET does not support HP 12040D MUX firmware with revision earlier than 5.02.
- Make sure your application runs locally without errors before executing it over a TELNET connection.
- Different terminals and computers may have different configuration requirements.
- Block mode applications have a limited number of supported configurations when using TELNET. Refer to "Block Mode Considerations."

Connection Considerations

There are several connectivity considerations:

- Only one connection for each TELNET user can be open at a time. *HP does not support multiple connections per each TELNET user*.
- A chained session is one where you have TELNET open to one computer and then you use TELNET from that computer to access another (a third) computer. Select a unique escape character for each host you wish to communicate with in a chained session. Refer to the subsection, "Chained TELNET Sessions" later in this section.
- For connections to any computer, always set the HP 1000 host terminal RECVPACE configuration (receive direction) to XON/XOFF.
- For block mode applications, terminals directly connected to an HP 1000 require XON/XOFF in both the transmit and receive directions. If the terminal is not set to XON/XOFF in both directions, a slow TELNET session may be overrun by the terminal and data will be lost or the application may hang.
- For block mode applications, terminals attached to the TS-8 with LSM 2.1 (or greater) software require XON/XOFF in only the *receive* direction. If XON/XOFF is set for the transmit direction, block mode applications may hang.
- You cannot initiate a remote session to a PC. Remote sessions between an HP 1000 and PC can only be initiated from the PC.

Terminal Settings to DEC VAX Computers

If you are using TELNET on the HP 1000 to connect to a remote DEC VAX host, you should set the communication protocol of the HP 1000 host terminal to `XON/XOFF`. The steps are as follows:

1. On the HP 1000, enter `WH` to display information about your terminal. Locate your session number.
2. Execute this command to set your terminal to `XON/XOFF` protocol:
   ```
   CI> cn,$session,34b,1b
   ```
3. Use TELNET to log on to the remote DEC VAX host.
4. Once you are logged on to the DEC VAX host, execute this command: `$ set terminal/vt100`. You can put this command in your LOGIN.COM file for automatic execution whenever you log on to the DEC VAX system.
5. Set your terminal to ANSI term type. See your terminal documentation for instructions.
6. When you have completed your TELNET session on the DEC VAX host and returned to the local HP 1000 host, reset your terminal to HP term type. See your terminal documentation for instructions.
7. Restore the local host to `ENQ/ACK` protocol by executing:
   ```
   CI> cn,$session,34b,2b
   ```

Chained TELNET Sessions

Chaining makes it possible to hop across the network to different hosts.

If you chain several TELNET sessions, you may want to select a unique escape character for each host in the chain, using the `ESCAPE` command. Then you can escape to the node of your choice by issuing the appropriate escape character.

If all nodes use the same escape character, you can only escape to your *local* node; you cannot escape to an intermediate node.

If you chain TELNET sessions, the `QUIT` or `EXIT` command will terminate all sessions, close all connections, and return you to the local host. If, however, you log off the remote host, only the most recent TELNET session is closed. Any other chained sessions are still active.

If TELNET terminates abnormally or is aborted, any remote session chained from your session is automatically terminated.

Block mode applications over chained TELNET sessions are not supported.

Block Mode Considerations

The TELNET standard specifies a character mode protocol. Character mode is the normal operation of a terminal.

With block mode, data is not transmitted one character at a time. Instead, an entire block of data is typed in locally on the terminal. When the enter key is pressed, the data is transmitted from the terminal to the computer.

Block mode for the HP 1000 is technically defined in the *RTE-A Driver Reference Manual*, part number 92077-90011.

The following products support block mode applications to the HP 1000:

- ARPA/Vectra revision 2.0 (or later) with the Advlink B.02.00 Emulator. On ARPA/Vectra, the RS (record separator) is the default escape character for TELNET. The RS character is also a special character in block mode. The TELNET escape sequence on the PC must be changed to another character.
- ARPA/9000 revision 7.0 (or later) with direct connect terminals only. HPTERM is not supported.
- TS-8 with LSM 2.1 (or later) software.
- Datacommunications and Terminal Controller (DTC).

Block mode applications over TELNET are not supported on the PC OfficeShare products.

Any TELNET user can communicate with a block mode application on the HP 1000 as long as the local terminal or terminal emulator can handle block mode I/O.

Block mode applications over chained TELNET sessions are not supported.

For block mode applications, terminals directly connected to an HP 1000 require XON/XOFF in both the transmit and receive directions. If the terminal is not set to XON/XOFF in both directions, a slow TELNET session may be overrun by the terminal and data will be lost or the application may hang.

For block mode applications, terminals attached to the TS-8 with LSM 2.1 (or greater) software require XON/XOFF in only the *receive* direction. If XON/XOFF is set for the transmit direction, block mode applications may hang.

Using Telnet

TELNET is scheduled at the RTE Command Interpreter level (`CI>`). TELNET can be invoked with or without the following parameter.

```
   TELNET [,host]
```

host	The name of the remote node to which you want to log on. The syntax of the host name is `node[.domain[.organization]]`, which is further described in the *NS-ARPA/1000 User/Programmer Reference Manual*. If *host* is not specified, TELNET displays a TELNET prompt and waits for you to enter a TELNET command.

?

Displays TELNET commands and help information. Same as the `HELP` command.

```
   ? [command]
```

command	Any TELNET command. If no command is specified, TELNET lists the TELNET commands, with a one-line description for each command. When a command is specified, TELNET displays a brief description of the command.

CLOSE

Closes the remote connection and logs off the remote session.

```
CL[OSE]
```

ESCAPE

Defines the TELNET escape character.

```
ES[CAPE] escape_char
```

escape_char Any seven-bit ASCII character except those listed in the table below. The default is CONTROL].

The escape character, when typed at the remote session, allows you to temporarily return to the local node. To go back to the remote node, enter a single carriage return at the TELNET prompt.

Illegal TELNET Escape Characters

Decimal Value	ASCII Character	Terminal Keys
0	NUL (null)	CONTROL @
4	EOT (end of transmission)	CONTROL D
8	BS (backspace)	CONTROL H
10	LF (linefeed)	CONTROL J
13	CR (carriage return)	CONTROL M
17	DC1 (XON)	CONTROL Q
18	DC2	CONTROL R
19	DC3 (XOFF)	CONTROL S
24	CAN (cancel)	CONTROL X
25	EM (end of medium)	CONTROL Y
30	RS (record separator)	CONTROL ^
31	US (unit separator)	CONTROL _
127	DEL (rubout)	DEL

The characters in the table cannot be defined as the remote escape character, because they have special terminal functions. Using one of these characters may cause communication problems with the remote node.

Do not define *escape_char* to be the same as the TELNET interrupt character. The TELNET interrupt character default is CONTROL Y and can be redefined by the INTERRUPT command.

EXIT

Closes the remote connection, logs off the remote session, and terminates TELNET. Same as `QUIT`.

```
EX[IT]
```

HELP

Displays TELNET commands and help information. Same as ?.

```
HE[LP] [command]
```

command	Any TELNET command. When a command is specified, `HELP` displays a brief description of the command. If no command is specified, `HELP` lists the TELNET commands, their syntax, and a one-line description for each command. See `?`, covered earlier in this section.

INTERRUPT

Changes the TELNET remote interrupt character.

```
IN[TERRUPT] intr_char
```

intr_char	Any seven-bit ASCII character except those listed in the table below. The default is CONTROL Y. The interrupt character is used to send a "BREAK" indication to the remote system without hitting the BREAK key on the terminal.

Illegal TELNET Escape Characters

Decimal Value	ASCII Character	Terminal Keys
0	NUL (null)	CONTROL @
4	EOT (end of transmission)	CONTROL D
8	BS (backspace)	CONTROL H
10	LF (linefeed)	CONTROL J
13	CR (carriage return)	CONTROL M
17	DC1 (XON)	CONTROL Q
18	DC2	CONTROL R
19	DC3 (XOFF)	CONTROL S
24	CAN (cancel)	CONTROL X
27	ES (cape)	CONTROL]
30	RS (record separator)	CONTROL ^
31	US (unit separator)	CONTROL _
127	DEL (rubout)	DEL

The characters in the table cannot be defined as remote interrupt characters, because they have special terminal functions. Using one of these characters may cause communication problems with the remote node.

Do not define `intr_char` to be the same as the TELNET escape character. The TELNET escape character default is CONTROL] and can be redefined by the `ESCAPE` command.

MODE

Changes the data transmission to either by line or by character.

```
         { L[INE]       }
MO[DE]   { C[HARACTER]  }
```

`L[INE]`	Sends data a line at a time from the local terminal to the remote node. In RTE-A, each line of data ends with a carriage return.
`C[HARACTER]`	Sends data a character at a time from the local terminal to the remote node. Each character is sent without waiting for an end-of-line character.

OPEN

Establishes a connection to a remote host.

```
OP[EN] host
```

host	The name of the remote node to which you want to log on. The syntax of the host name is `node[.domain[.organization]]`, which is further described in the *NS-ARPA/1000 User/Programmer Reference Manual*. If *host* is not specified, TELNET connects you to your local node.

QUIT

Closes the remote connection, logs off the remote session, and terminates TELNET. Same as `EXIT`.

```
QUIT
```

RUN

Runs a program at the local node.

```
RU[N] program
```

program	The name of a program on the local system.

SEND

Sends special characters or commands to the remote node.

```
          ⎧ E[SCAPE]     ⎫
          ⎪ IN[TERRUPT]  ⎪
    SE[ND]⎨ A[YT]        ⎬
          ⎪ B[REAK]      ⎪
          ⎩ IP           ⎭
```

`E[SCAPE]`	The `SEND ESCAPE` command sends the TELNET escape character as a data character to the remote node. Normally, the TELNET escape character is removed from any data sent to the remote node. The `SEND ESCAPE` command is needed and helpful when you *do* need to send the escape character as a data character. If the application program running on the remote system requires you to input the current escape character, you can do one of two things: • Change the escape character for the duration of the program with the `ESCAPE` command. • Press the escape character to return to the TELNET prompt (`TELNET>`), then use `SEND ESCAPE` to send the escape character as input to the remote application program. (See example below.)
`IN[TERRUPT]`	The `SEND INTERRUPT` command sends the interrupt character as a data character to the remote node. Normally, the interrupt character will suspend, interrupt, abort, or terminate the remote process. The `SEND INTERRUPT` command is needed and helpful when you *do* need to send the interrupt character as a data character.
`A[YT]`	The `SEND AYT` command asks the remote node to return evidence that the TELNET connection is still open. If the connection is still open, the remote node returns an affirmative response (e.g., terminal beep, `yes`, etc.). `AYT` stands for "Are you there?"
`B[REAK]`	The `SEND BREAK` command invokes a break at the remote node. This command is equivalent to pressing the BREAK key at the remote node.
`IP`	The `SEND IP` command sends an interrupt to the remote node. This command is equivalent to pressing the interrupt character (CONTROL `Y`) at the remote node.

STATUS

Displays the current state of the TELNET connection.

```
    ST[ATUS]
```

Error Messages

The following error messages are returned to the current list device when an error is encountered by the TELNET user program. After a message is printed, the user may again be presented with the TELNET prompt.

TELNET User Error Messages

Message	Meaning
`Internal TELNET error` `(TELNET ERR 1)`	An internal TELNET error has occurred.
`Unable to connect to target computer` `(TELNET ERR 2)`	A connection could not be established to the target computer, because of one of the following problems: (1) the target computer name is incorrect, (2) the target computer is not connected to the network, (3) there are not enough system resources, or (4) TELNET has not been initialized at the target computer.
`Input line too long` `(TELNET ERR 3)`	The input line was greater than 80 characters.
`Cannot find closing quotation mark` `(TELNET ERR 4)`	An opening quotation mark was found but not the closing quotation mark.
`Illegal command` `(TELNET ERR 5)`	A TELNET command was entered incorrectly with the wrong syntax or parameter(s).
`Unknown command` `(TELNET ERR 6)`	A TELNET command was entered incorrectly.
`Input command is too big` `(TELNET ERR 7)`	You typed in a command with more than 256 characters.
`Unable to initialize TELNET` `(TELNET ERR 8)`	This error may be returned for one of the following reasons: (1) TELNET was unable to acquire sufficient system resources (such as sockets); or (2) an error occurred in accessing Distributed System Available Memory (DSAM) or tables in DSAM; (3) user's terminal is not connected to a D-MUX or A400 MUX, or is not a remote TELNET session.
`Illegal escape character` `(TELNET ERR 9)`	An illegal escape character was entered.
`^Y is the interrupt character` `(TELNET ERR 10)`	The interrupt character was specified in the `ESCAPE` command instead of an escape character.
`Illegal interrupt character` `(TELNET ERR 11)`	An illegal character was specified for the interrupt character in the `INTERRUPT` command.
`^[ is the escape character` `(TELNET ERR 12)`	The escape character was specified in the `INTERRUPT` command instead of an interrupt character.
`Mode change unavailable` `(TELNET ERR 13)`	A `MODE` command was entered when there is no remote connection open.
`Already line` `(TELNET ERR 14)`	A line `MODE` command was entered, and the transmission mode is already in line mode.
`Already character` `(TELNET ERR 15)`	A character `MODE` command was entered, and the transmission mode is already in character mode.
`Unknown parameter` `(TELNET ERR 16)`	The parameter entered is unknown for the command.
`Ambiguous parameter` `(TELNET ERR 17)`	Not enough characters have been entered for a parameter to differentiate it from another parameter.
`No connection open` `(TELNET ERR 18)`	A TELNET command was entered that requires an open connection.
`Node name is too long` `(TELNET ERR 19)`	A node name was entered with more than 50 characters.

2

FTP

Table of Contents

Using FTP

FTP is scheduled at the RTE Command Interpreter level (`CI>`). FTP can be invoked with or without the following parameters.

```
   FTP [-i] [-l[filename]] [-n] [-tfilename] [-v] [-g] [-q]
       [-u[username:password]] [host]
```

`–i`	Disables interactive prompting during multiple-file operations. Interactive prompting lets you selectively proceed with each file.
`–l[`*`filename`*`]`	Logs FTP output to *`filename`* in addition to the user's terminal. If *`filename`* is omitted, then `FTP.LOG` is used. There must be no space between `–l` and the file name. If *`filename`* already exists, output is appended to the file.
`–n`	Disables auto-login. If auto-login is disabled, you must use the `USER` command to login to a remote host. If auto-login is enabled, FTP prompts for a user name once a connection is established to a remote host.
`–t`*`filename`*	Accepts input from the transfer file specified by *`filename`*. There must be no space between `–t` and the file name.
`–v`	Enables verbose output. Verbose output displays all responses from any remote host that you are connected to. These responses indicate whether FTP commands completed successfully and the file transfer statistics.
`-g`	Disables file name globbing during multiple file operations. Globbing expands the wild card characters before proceeding with the multiple command.
`-q`	Enables quiet mode for transfer files. The normal informative messags are not output to the terminal.
`–u[username:password]`	Specifies the user and password to use. FTP will use the username and password to automatically logon to the host system. If either one needs lower case characters, the string must be surrounded by back quotes ('). An example is: `–u'MyName:mypass'`.
`host`	Specifies the host to which you want to log on. You may use the host's node name or IP address for the *`host`* parameter. The syntax for the host's node name is *`node`*`.[.`*`domain`*`[.`*`organization`*`]`, and the syntax for the IP address is *`nnn.nnn.nnn.nnn`*. Both are further described in the *NS-ARPA/1000 User/Programmer Reference Manual*.
	If *`host`* is not specified, FTP displays the FTP prompt and waits for you to enter an FTP command. In this case, you must specify the `OPEN` command to open a connection to a host.

!

Invokes CI or runs the specified program on the local HP 1000 host.

```
    ! [prog_name]
```

prog_name	Any program that you can execute singly. Once the command is executed, you automatically return to FTP. If no command is specified after the exclamation mark, you will remain in CI until you execute the CI `EX` command.

?

Displays FTP commands and help information. You may use a single question mark (`?`) or double question marks (`??`). Same as `HELP` command.

```
    ?[?] [command]
```

command	Any FTP command. If no command is specified, FTP lists the currently supported FTP commands. When a command is specified, FTP displays a brief description of the command. A space or comma must separate the question mark and the *command* parameter.

..

Sets the working directory on the remote host to the parent directory.

```
    ..
```

/

Displays the FTP command stack. Similar to the command stack display function (/) in RTE-A.

```
        ┌ linecount ┐
    /   │ /...      │
        └ .text     ┘
```

linecount	Optional command line count integer, from 1 to 12, that specifies the number of command lines from the last command entered to be displayed.
/...	Optional extra slashes, up to 12 slashes, that you may specify. If two extra slashes are specified, the last two commands executed are displayed. If three extra slashes are specified, the last three commands are displayed, and so on.
.text	String of text that FTP uses to search the command stack. The text must be preceded by a period (.). FTP displays commands in the stack that contain the specified string of text.

APPEND

Transfers a local file to the end of a remote file.

```
   AP[PEND] local_file [remote_file]
```

local_file	Specifies a valid file on the local host to be appended to the remote file.
remote_file	Specifies a valid file path on the remote host to append the local file. If the *remote_file* does not exist, FTP creates it before appending the local file. If the *remote_file* parameter is omitted, FTP uses the *local_file* name as the *remote_file* name.

ASCII

Sets the file transfer type to `ASCII`.

```
   AS[CII]
```

BELL

Specifies that a bell sound is generated after each file transfer completes. This command toggles.

```
   BE[LL]
```

BINARY

Sets the FTP file transfer type to binary.

```
   BI[NARY]
```

BYE

Closes the remote connection and exits from FTP. Same as `EXIT` and `QUIT`.

```
   BY[E]
```

CD

Sets the working directory on the remote host to the specified directory.

```
   CD remote_directory
```

remote_directory	Specifies a valid directory on the remote host to be the working directory. By default, the working directory is the default login directory.

CLOSE

Closes the remote connection and remains in FTP.

```
   CL[OSE]
```

DEBUG

Prints the commands that are sent to the remote host. Used for debugging the current FTP session. This command toggles debug mode.

```
   DEB[UG]
```

DELETE

Deletes the specified remote file or remote directory.

   DEL[ETE] *remote_file*

remote_file	Specifies a valid file path on the remote host to be deleted. This can be a file or an empty directory.

DIR

Writes an extended directory listing of a remote directory or file to the terminal or to an output file.

   DI[R] [*remote_listing*] [*local_file*]

remote_listing	Specifies the remote directory or file mask from which a directory listing is to be generated. If this parameter is not specified, a directory listing of the remote working directory is generated.
local_file	Specifies the output file on the local host to store the directory listing. If this parameter is not specified, the directory listing is written to your terminal.

DL

Writes an extended RTE-A directory listing of a remote directory or file to the terminal or to an output file.

   DL [*remote_listing*] [*local_file*]

remote_listing	Specifies the remote directory or file mask from which a directory listing is to be generated. If this parameter is not specified, a directory listing of the remote working directory is generated.
local_file	Specifies the output file on the local host to store the directory listing. If this parameter is not specified, the directory listing is written to your terminal.

EXIT

Closes the remote connection and exits from FTP. Same as `BYE` and `QUIT`.

```
   E[XIT]
```

FORM

Sets the FTP file transfer form to the specified format. The only supported format is `non-print`.

```
   F[ORM] format
```

format	Specifies the file transfer format. Currently the only supported format is `non-print`. Non-print format specifies that no vertical format information is contained.

GET

Transfers a remote file to a local file. Same as `RECV`.

```
   GE[T] remote_file [local_file]
```

remote_file	Specifies a valid file path on the remote host to be copied to the local host.
local_file	Specifies the file on the local host to copy into. If this parameter is not specified, FTP uses the remote file path as the local file path. If a local file name is specified without a directory, the current working directory on the local host is used. *If a local file with the same file name already exists before the file transfer, it is overwritten without warning*.

GLOB

Toggles file name globbing (expansion) for multiple file operations. When file name globbing is enabled, FTP expands wild card characters in multiple file and directory operations. In other words, FTP uses the wild card characters as wild cards and not as the characters they normally represent. The wild card characters used depend on the file system processing the FTP command.

```
   GL[OB]
```

HASH

Specifies the printing of a hash sign (#) for each data block transferred. The size of the data block is 1024 bytes. This command toggles the printing of hash signs.

```
   HA[SH]
```

HELP

Displays FTP commands and help information. Same as `?` or `??`.

```
   HE[LP] [command]
```

command	Any FTP command. When a command is specified, FTP displays a brief description of the command. If no command is specified, FTP lists the currently supported FTP commands.

LCD

Sets the local working directory to the specified directory.

```
   LC[D] [local_directory]
```

local_directory	Specifies a valid directory on the HP 1000 host to be the local working directory. If *local_directory* is not specified, FTP returns you to your home directory.

LL

Specifies a local log file to which FTP sends commands and messages in addition to displaying them on the user's terminal.

```
   LL [local_file]
```

local_file	Specifies a valid file name on the HP 1000 host as a log file. All terminal output generated by FTP is logged into this file in addition to your terminal. If *local_file* is not specified, FTP prompts you for the log file name. If the file already exists, output is *appended* to it. To close the log file, use `LL,1`.

LS

Writes an extended directory listing of a remote directory or file to your terminal or to a local file on the HP 1000.

```
   LS [remote_listing] [local_file]
```

remote_listing	Specifies the remote directory or file mask from which a directory listing is to be generated. If this parameter is omitted, `LS` lists the remote working directory.
local_file	Specifies a valid file path on the local HP 1000 host to store the directory listing. If this parameter is omitted, the directory listing is displayed on your terminal.

MDELETE

Deletes multiple remote files.

```
   MDE[LETE] remote_file [remote_file ...]
```

remote_file	Specifies a valid file path on the remote host to be deleted. This can be a file or an empty directory. The ellipsis (. . .) means that you may specify multiple remote files or directories, delimited by a comma or by one or more blank spaces. You may use wild card characters in the remote file names.

MDIR

Writes an extended directory listing of multiple remote directories or files to a local file.

```
   MD[IR] remote_listing [remote_listing ...] local_file
```

remote_directory	Specifies the remote directories or file masks from which a directory listing is to be generated. The ellipsis (. . .) means that you can specify multiple remote directories or files, delimited by a comma or by one or more blank spaces.
local_file	A valid file path on the local HP 1000 host to store the remote listing. This parameter is required, because `MDIR` does not output the remote listing to the terminal. FTP always uses the last parameter in the `MDIR` command string as the *local_file*.

MGET

Transfers multiple remote files to the local host.

```
   MG[ET] remote_file [remote_file ...]
```

remote_file	Specifies a valid file path on the remote host for the file to be transferred. The ellipsis (...) means that you may specify multiple remote files, delimited by a comma or by one or more blank spaces. You may use wild card characters in the remote file names. The files are transferred to local files with the same directory paths and names as the source files. *If a local file with the same file name already exists before the file transfer, it is overwritten without warning*.

MKDIR

Creates a directory on the remote host.

```
   MK[DIR] remote_directory [lu]
```

remote_directory	Specifies a valid directory path on the remote host.
lu	Specifies the LU on which the remote directory will reside.

MLS

Writes an abbreviated directory listing of multiple remote directories or files to a local file.

```
   ML[S] remote_listing [remote_listing ...] local_file
```

remote_listing	Specifies the remote directories or file masks from which a directory listing is to be generated. The ellipsis (. . .) means that you may specify multiple remote directories or files, delimited by a comma or by one or more blank spaces.
local_file	A valid file path on the local host to store the remote listing. This parameter is required, because `MLS` always outputs the remote listing to a local file and not to a terminal. FTP always uses the last parameter in the `MLS` runstring as the *local_file*.

MODE

Specifies the file transfer mode.

```
   MO[DE] mode_name
```

mode_name	A valid FTP file transfer mode. The only currently supported mode is `stream`. Stream mode specifies that the data is transmitted as a stream of bytes. There is no restriction on the representation type used. If the structure is a file structure (which is the default), the End-Of-File is indicated by the sending host closing the data connection and all bytes are data bytes.

MPUT

Transfers multiple local files to the remote host.

```
   MP[UT] local_file [local_file ...]
```

local_file	Specifies a valid file path on the local host to be transferred to the remote host. The ellipsis (. . .) means that you can specify multiple local files, delimited by a comma or by one or more blank spaces. You may use wild card characters in the file names. The files are transferred to the remote host, under the same directory and file names as the source files. *If a remote file with the same file name already exists before the file transfer, it is overwritten without warning.*

NLIST

Writes an abbreviated directory listing of a remote directory or file to your terminal or to a local file on the HP 1000.

```
   N[LIST] [remote_listing] [local_file]
```

remote_listing	Specifies the remote directory or file mask from which a directory listing is to be generated. If this parameter is omitted, `NLIST` lists the remote working directory.
local_file	Specifies a valid file path on the local HP 1000 host to store the directory listing. If this parameter is omitted, the directory listing is displayed on your terminal.

OPEN

Establishes a connection with a specified remote host.

```
   O[PEN] host
```

host	Specifies the host to which you want to log on. You may use the host's node name or IP address. The syntax for the host node name is *node*[.*domain*[.*organization*], and the syntax for the IP address is *nnn.nnn.nnn.nnn*. Both are further described in the *NS-ARPA/1000 User/Programmer Reference Manual*.

PROMPT

Toggles interactive prompting for multiple file operations. Interactive prompting occurs during multiple file operations to allow you to selectively proceed with each file. By default, interactive mode is enabled.

```
   PR[OMPT]
```

PUT

Transfers a local file to the remote host. Same as `SEND`.

```
   PU[T] local_file [remote_file]
```

local_file	Specifies a valid file path on the local host to be transferred.
remote_file	Specifies a valid file path on the remote host to be transferred into. If this parameter is omitted, FTP uses the local file path as the file name on the remote host. If a remote file name is specified without a directory, the current working directory on the remote host is used. *If a remote file with the same file name already exists, it is overwritten without warning*.

PWD

Writes the name of the remote working directory to the terminal.

```
PW[D]
```

QUIT

Closes the remote connection and exits from FTP. Same as `BYE` and `EXIT`.

```
QUI[T]
```

QUOTE

Sends arbitrary server commands to the remote host.

```
QUO[TE] arguments
```

arguments	Specifies a valid FTP server command to be sent to the remote host. The *arguments* are sent "as is," including commas if included. Use for debugging purposes.

RECV

Transfers a remote file to the local host. Same as `GET`.

```
REC[V] remote_file [local_file]
```

remote_file	Specifies a valid file path on the remote host to be transferred to the local host.
local_file	Specifies a file on the local host to be copied into. If this parameter is omitted, the local file will have the same directory path and file name as the remote file. If a local file name is specified without a directory, the current working directory of the local host is used. *If a local file with the same file name already exists, it is overwritten without warning*.

REMOTEHELP

Displays the currently supported FTP server commands on the remote host.

```
REM[OTEHELP] [command]
```

command	Any FTP server command on the remote host. FTP displays help information on the specified server command from the remote host. If *command* is omitted, FTP displays a list of currently supported FTP server commands on the remote host.

RENAME

Renames a remote file or remote directory.

```
   REN[AME] remote_old remote_new
```

remote_old	Specifies the original name of a remote file or remote directory to be renamed.
remote_new	Specifies the new name for the remote file or remote directory. If *remote_new* already exists, FTP issues a warning and ignores the command.

RMDIR

Removes an empty directory from the remote host.

```
   RM[DIR] remote_directory
```

remote_directory	Specifies a valid directory path on the remote host to be removed. The directory must be empty or FTP issues a warning and ignores the command.

RTEBIN

Sets the transfer type to `BINARY` such that for subsequent `PUT` or `MPUT` commands if only the filename is specified, then the file type, size, and record length are included in the destination file descriptor.

```
   RT[EBIN]
```

SEND

Transfers a local file to the remote host. Same as `PUT`.

```
   SE[ND] local_file [remote_file]
```

local_file	Specifies a valid file path on the local host to be transferred.
remote_file	Specifies a valid file path on the remote host to be transferred into. If this parameter is omitted, FTP uses the local file path as the remote file name. If a remote file name is specified without a directory, the current working directory of the remote host is used. *If a remote file with the same file name already exists, it is overwritten without warning*.

SITE

Sends arguments, verbatim, to the server host as a `SITE` command.

```
   SI[TE] arguments
```

arguments	Specifies a valid FTP server `SITE` command to be sent to the remote host. The *arguments* are sent “as is,” including commas if they were included.

STATUS

Writes the current status of FTP to the terminal.

```
STA[TUS]
```

STRUCT

Sets the FTP file transfer structure to the specified structure.

```
STR[UCT] struct_name
```

struct_name	Specifies the FTP file transfer structure. Currently, the only supported file transfer structure is `file`. File structure means that there is no internal structure and the file is considered to be a continuous sequence of data bytes.

SYSTEM

Returns the type of the operating system running on the server.

```
SY[STEM]
```

TR

Specifies a local command input file (also called a transfer file) containing FTP commands.

```
TR local_file
```

local_file	Specifies a local transfer file containing FTP commands. FTP executes the commands in this file. The `TR` command lets you execute FTP from a command file rather than entering each FTP command via your terminal keyboard. You may include any valid FTP commands in the transfer file. FTP terminates when the `EXIT`, `BYE`, or `QUIT` command is executed. If the end-of-file is found before any of these commands, control is passed back to FTP.

TYPE

Sets the FTP file transfer type to the specified type.

```
TY[PE] [type_name]
```

type_name	Specifies the FTP file transfer type. Currently, *type_name* may be one of the following: - `ASCII`, `A` = set the file transfer type to ASCII. - `BINARY`, `B`, `I`, = set the file transfer type to binary. If this parameter is omitted, `TYPE` displays the current FTP file transfer type on the terminal.

USER

Logs on as a different user on the currently connected remote host.

```
   U[SER] [user_name] [password]
```

user_name	Specifies the account on the remote host to log on. If the *user_name* is not specified, FTP prompts you for it.
password	Specifies the password for the account, if required. If a password is required and not specified, FTP prompts you for it.

VERBOSE

Specifies verbose output. This command toggles verbose output. Verbose output displays all responses from any remote host to which you are connected. These responses tell you whether or not FTP commands completed successfully. By default, verbose output is enabled if FTP input comes from your keyboard. Verbose output is disabled if FTP input comes from an FTP transfer file.

```
   V[ERBOSE]
```

Error Messages

The following error messages are returned to the current list device when an error is encountered by the FTP user program. After a message is printed, the user may again be presented with the FTP prompt.

FTP User Error Messages

Message	Meaning
`FTP internal error encountered.` `(FTP ERR 1)`	An internal FTP error has occurred.
`Unable to connect to specified computer.` `(FTP ERR 2)`	A connection could not be established to the remote host, because of one of the following problems: (1) the remote computer name is incorrect, (2) the remote computer is not connected to the network, (3) there are not enough system resources, (4) FTP has not be initialized at the remote host.
`Illegal option found. Valid ones are -G, -I, -L, -N, -Q, -T, -U, -V.` `(FTP ERR 3)`	An FTP runstring was entered incorrectly with an invalid option.
`Two nodenames in runstring; ignoring second one.` `(FTP ERR 4)`	An FTP runstring containing more than one *host* parameter was entered. FTP will use the first *host* parameter.
`Ambiguous command.` `(FTP ERR 5)`	An FTP command was entered incorrectly without enough characters to make it a legal FTP command.
`Invalid command.` `(FTP ERR 6)`	An invalid FTP command was entered.
`FTP login failed.` `(FTP ERR 7)`	FTP server did not recognize the user name or password that was specified.
`Illegal reply code.` `(FTP ERR 8)`	An illegal reply code was received from the FTP server. Legal reply codes begin with 1, 2, 3, 4, or 5.
`Unexpected reply code from server. Expected reply code` *code*. `(FTP ERR 9)`	An incorrect reply code was received from the FTP server. The command sequence between FTP and FTP server can be out of sequence.
`Unable to retrieve information on local host.` `(FTP ERR 10)`	FTP unable to obtain IP and port address information on local node for sending the `port` command to FTP server.
`Not connected. Please use the OPEN command first.` `(FTP ERR 11)`	You are currently not connected to a remote host.
`Already connected to` *host*. `Please use USER or CLOSE first.` `(FTP ERR 12)`	You are already connected to a remote host.
`Not a valid type. Only ASCII and BINARY are supported.` `(FTP ERR 13)`	An invalid transfer type was specified.
`Unimplemented command.` `(FTP ERR 14)`	A command, not implemented on ARPA/1000, was specified.
`Unimplemented option.` `(FTP ERR 15)`	An option, not implemented on ARPA/1000, was specified.
`No transfer file specified. Ignoring option -T.` `(FTP ERR 16)`	No file name was specified with the −T option in the FTP runstring.

Message	Meaning
`Two log files in runstring. Ignoring second one. (FTP ERR 17)`	Two log files were specified in the FTP runstring. The second log file was ignored.
`Two transfer files in runstring. Ignoring second one. (FTP ERR 18)`	Two transfer files were specified in the FTP runstring. The second file was ignored.
`Same name given for log and transfer files. Ignoring both. (FTP ERR 19)`	The same file name was used for log and transfer file.
`Unknown nodename. No such node exists. (FTP ERR 20)`	An unknown remote host was specified.
`No username or password specified (FTP ERR 21)`	The username and password were not specified for the `-u` option.
`Error encountered sending` *command* `to the server. (FTP ERR 22)`	FTP received a NetIPC error.
`Error encountered receiving reply code from server. (FTP ERR 23)`	FTP encountered an error when receiving reply code from the server.
`Network is not up. (FTP ERR 24)`	The network is currently down.
`Require network resource is not available. (FTP ERR 25)`	A required resource, such as DSAM, resource number, or sockets, is not currently available.
`Connection aborted. Remote host unreachable. (FTP ERR 26)`	Connection to remote host was aborted, because (1) remote host has been shut down, (2) some network links are malfunctioning, (3) the network is extremely congested.
`Illegal nodename. Possible syntax error. (FTP ERR 27)`	An invalid remote host was specified.
`A timeout has occurred. Try request later. (FTP ERR 28)`	An internal timeout occurred while running FTP.
`Break acknowledged. Terminating transfer. (FTP ERR 29)`	A break was issued by the user, and accepted by FTP.
`FTP: Connection lost to peer node. Connection reset. (FTP ERR 30)`	The connection to the peer node was aborted, so the connection has been reset.
`Unexpected network error encountered. (FTP ERR 31)`	An internal NetIPC error has occurred.
`Input line too long. (FTP ERR 32)`	The input line was greater than 256 characters.
`APPEND is not allowed in BINARY mode. (FTP ERR 33)`	`APPEND` is not supported for binary file transfers.
`FTP internal error encountered (FTP ERR 34)`	An internal FTP error has occurred.
`Unable to use CMNDO editing. HpStartCmndo error (FTP ERR 35)`	An error was returned from HpStartCmndo.

3

FSRV – HP 1000 File Server

Table of Contents

Using FSRV − HP 1000 File Server

Networking must already be initialized and enabled before running the FSRV file server program. Only one copy of FSRV is allowed to run at one time. FSRV is typically scheduled in the welcome file with any of the following runstring parameters required for your environment:

```
   fsrv [-cemnuv] [-a type] [-d seconds] [-s blks] [-t timezone]
```

where,

`-a` *type*	(ASCII conversion) Variable length RTE files of the *type* specified are transferred in ASCII mode. A line-feed character is added at the end of every record in the file. FSRV always converts type 4 files. Note that file types used for binary data should not be specified.
`-c`	(Case folding) Inhibit the case folding of all file names. When you select this option, file names specified with any uppercase characters are rejected.
`-d` *seconds*	(Directory timeout) Use the *seconds* argument as the timeout for the server's directory buffer cache. This timeout specifies the maximum length of time that a directory buffer can remain in cache on the server. A timeout of "0" disables the directory buffer cache. (Default: 10 seconds)
`-e`	(EOF cache) Disable EOF caching of the server. By default, the server caches eof information for files being written by a client. The −e runstring option causes the ^EOF pointer to be updated on the disk each time a write request extends a file.
`-m`	(File name mapping) Inhibit the reserved character mapping. When you specify this option, all of the RTE file name restrictions are in effect when accessing or creating files from a client.
`-n`	(Mount table file) Specifies not to use the mount table file to restore previous mount requests and not to update the mount table file after mounts and umounts.
`-s` *blks*	(Default file size) Use the *blks* argument as the default file size for files created from a client. Performance considerations apply when using this option. (Default: 128 blocks)
`-t` *timezone*	(Time Zone) Use the information in `/etc/tztab` for the time zone specified in the *timezone* argument. If the −t option is not used, the local RTE system is assumed to be running in UTC (Coordinated Universal Time).

`-u`	(Update) The update mode causes the modification time for all directories to always be returned as the current time.
`-v`	(Verbose) The verbose mode is provided for diagnostic purposes only. Usage of this option is not recommended under normal conditions. Setting the break flag on the FSRV server program also toggles the state of the verbose flag. The verbose mode messages are written to the NS event logger as protocol-specific information (P bit, class 1). The `LOGCHG` utility can be used to change the log mask that `EVMON` is using.

FSRV Error Messages

The following error messages are returned by the FSRV server program. When these errors occur they are reported to the event logger for NS-ARPA. Event logging is enabled with the EVMON utility. An explanation of the error and suggested action are indicated as appropriate. The actual error message returned to the NFS* client under these conditions will depend upon the error handling by the client.

fsrv: atach error : <#>

FSRV was unable to attach to the new session. See the ATACH utility information in the *RTE-A Relocatables Reference Manual*, part number 92077-90037, for an explanation of error numbers returned by ATACH.

fsrv: Cannot find the time zone information for <TIMEZONE>

The time zone specified in the −t runstring option could not be found in the `/etc/tztab` file.

fsrv: clgon error : <#>

FSRV was unable to logon. (Requires LOGON Rev.6200.) See the CLGON utility information in the *RTE-A Relocatables Reference Manual*, part number 92077-90037, for an explanation of error numbers returned by CLGON.

fsrv: Error decoding /etc/tztab.

The `/etc/tztab` file cannot be decoded. Verify that the file entries are in the correct format.

fsrv: Invalid file type specified with −a option.

The −a option can only be used with variable record length file types.

* NFS is a trademark of Sun Microsystems, Inc.

fsrv: MOUNT procedure: DUMP is not implemented.

MOUNTPROC_DUMP procedure to return server's mount entries is not implemented.

fsrv: Multiuser session not enabled.

Multiuser session must be enabled for the HP 1000 file server to operate.

fsrv: NFS procedure: LINK is not implemented.

NFSPROC_LINK procedure to create a link to a file is not implemented; links to a file are not supported on RTE-A.

fsrv: No more room in mount table.

There is insufficient free space in the program to store the mount table. Increase the HEAP area of the FSRV program. If the HEAP area cannot be increased, the size of the time zone table may possibly be reduced. This can be achieved by eliminating the tztab rule in your time zone for future years and/or past years.

fsrv: Not enough room to restore the mount table from
fsrv: /etc/fsrv.mnt. Only mounting <#> directories.

There is insufficient free space in the program to read the prior mount table. Increase the HEAP area of the FSRV program. If the HEAP area cannot be increased, the size of the time zone table may possibly be reduced. This can be achieved by eliminating the tztab rule in your time zone for future years and/or past years.

fsrv: Not enough room to store the time zone table.

There is insufficient free space in the FSRV program to store the time zone table. The time zone table is stored in the HEAP area of the FSRV program. If the HEAP area cannot be increased, the size of the time zone table must be decreased. This can be achieved by eliminating the tztab rule in your time zone for future years and/or past years.

fsrv: PORTMAP procedure: CALLIT is not implemented.

PMAPPROC_CALLIT procedure to allow remote procedure (RPC) calls to any registered RPC program is not implemented.

fsrv: PORTMAP procedure: SET is not implemented.

PMAPPROC_SET procedure to allow users to register an RPC program with the port mapper is not implemented. RTE users are unable to register their own applications with FSRV.

fsrv: PORTMAP procedure: UNSET is not implemented.

PMAPPROC_UNSET procedure to allow users to unregister an RPC program with the port mapper is not implemented. RTE users are unable to register their own applications with FSRV.

fsrv: retry request from <000.000.000.000>
fsrv: prog=<PROG NAME> proc=<PROC NAME>

Client's request timed out. FSRV detected the retry.

fsrv: RPC authentication error.

Only the AUTH_UNIX and AUTH_NONE authentication protocols are supported.

fsrv: RPC version mismatch.

Only RPC version 2 is supported.

fsrv: Syntax error in /etc/hosts.

Review the entries in the `/etc/hosts` file for syntax errors.

fsrv: Unable to get a session number.

GETSN failed; the system is out of session numbers.

fsrv: Unable to obtain RTE group ID for <GROUPNAME>

A group name was found in the `/etc/ux_groups` file that is not a valid group on the RTE system.

fsrv: Unable to obtain RTE user ID for <USERNAME>

A user name was found in the `/etc/ux_users` file that is not a valid user on the RTE system.

fsrv: Unknown host <HOSTNAME>

A HOSTNAME specified in `/etc/exports` could not be found in the `/etc/hosts` file.

fsrv: Unknown option in /etc/exports.

Review the entries in the `/etc/exports` file for syntax errors.

BSD IPC Errors Returned by FSRV

The following types of errors are reported by FSRV when any of the networking calls used by FSRV report an error. See the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000*, part number 91790-90060, for information on the specific BSD IPC error message number that is returned by FSRV.

fsrv:socket error : <#>

fsrv:bind error : <#>

fsrv:setsockopt error : <#>

fsrv:getsockname error : <#>

fsrv:listen error : <#>

fsrv:select error : <#>

fsrv:accept error : <#>

fsrv:recv error : <#>

fsrv:recvfrom error : <#>

fsrv:send error : <#>

fsrv:sendto error : <#>

fsrv:shutdown error : <#>

fsrv:getpeername error : <#>

4

Berkeley Software Distribution Interprocess Communication

Table of Contents

Using BSD IPC

Berkeley Software Distribution Interprocess Communication (BSD IPC) provides a programming interface for client-server processes on the same or different machines to exchange data in a peer-to-peer manner. BSD IPC processes communicate with each other via *sockets*. Sockets are local data structures with associated resources used for interprocess communication. Table 4-1 lists the BSD IPC calls involved in creating, using, and terminating a BSD IPC connection with stream sockets, and Table 4-2 lists the calls used for datagram sockets.

Table 4-1. Building a BSD IPC Connection

Server Process	Client Process
1. `socket()` creates a socket	1. `socket()` creates a socket
2. `bind()` binds an address	2. `bind()` binds an address
3. `listen()` sets up a listen queue	
4. `accept()` waits & accepts a connection	
	5. `connect()` requests a connection
6. `send()` sends data `recv()` receives data `sendmsg()` sends vectored data `recvmsg()` receives vectored data	6. `send()` sends data `recv()` receives data `sendmsg()` sends vectored data `recvmsg()` receives vectored data
7. `shutdown()` shuts down a connection	7. `shutdown()` shuts down a connection

Usually, the server process is scheduled first. It creates a socket, binds an address to the socket, sets up a listen queue, and waits for requests from client processes.

The client process creates a socket and requests connection to the server. Once the server accepts the request, full-duplex connection is established between the two processes and the distinction between client and server can cease to exist. Both peer processes can send and receive data, as well as terminate the connection.

With datagram sockets there is no concept of a connection between the client and server processes. A client initiates a transaction by sending a datagram to the server. Both processes can send and receive datagrams to complete the transaction.

Table 4-2. Using Datagram Sockets

Server Process	Client Process
1. `socket()` creates a socket	1. `socket()` creates a socket
2. `bind()` binds an address (See note below)	2. `bind()` binds an address (See note below)
3. `recvfrom()` waits and receives datagram	
	4. `sendto()` sends datagram
5. `sendto()` sends datagram `recvfrom()` receives datagram	5. `sendto()` sends datagram `recvfrom()` receives datagram
6. `shutdown()` releases a socket	6. `shutdown()` releases a socket

The BSD IPC calls are summarized on the following pages in alphabetical order.

accept()

Accepts a connection on a socket and creates a new socket. The call returns the new socket descriptor. The `accept()` call is used by the server process to wait for and accept a connection request from the client process.

Syntax

```
newsocket = accept(socket, addr, addrlen)

int                 newsocket, socket, *addrlen;
struct sockaddr_in  *addr;
```

Parameters

newsocket	New socket descriptor created by `accept()`. If the call is successful, the value returned is an integer equal to or greater than `0`. If the call fails, it returns `-1`.
socket	Original socket descriptor, created by a previous `socket()` call.
addr	Pointer to address structure. The address structure should be of `sockaddr_in` type. Refer to “Preparing Socket Addresses” in the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for details. On return, this structure contains the socket address of the client process that is connected to the server’s new socket.
addrlen	Pointer to an integer variable that contains the length, in bytes, of the address structure specified by *addr* (for example, length of structure `sockaddr_in`, which is 16 bytes). On return, *addrlen* contains the length, in bytes, of the actual client socket address returned in *addr*.

bind()

Binds the specified socket address to the socket.

Syntax

```
result = bind(socket, addr, addrlen)

int                 result, socket, addrlen;
struct sockaddr_in  *addr;
```

Parameters

result	`0` if `bind()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
addr	Pointer to socket address structure that is to be bound to *socket*. The socket address should use a structure of `sockaddr_in` type. Refer to "Preparing Socket Addresses" in the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for details.
addrlen	Length (in bytes) of the socket address structure (for example, size of structure `sockaddr_in`, which is 16 bytes). *Addr* should be at least 16 bytes.

connect()

Initiates a connection request on a socket. This call is issued by the client process to connect to a specified server process.

Syntax

```
result = connect(socket, addr, addrlen)

int                 result, socket, addrlen;
struct sockaddr_in  *addr;
```

Parameters

result	`0` if `connect()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket requesting a connection.
addr	Pointer to a structure containing the socket address of the remote (server) socket to which the connection is to be established. The socket address should be of `sockaddr_in` type. Refer to "Preparing Socket Addresses" in the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for details.
addrlen	Length, in bytes, of the address structure specified by *addr* (for example, length of structure `sockaddr_in`, which is 16 bytes). *Addrlen* should be at least 16 bytes.

fcntl()

Provides socket I/O control. Can be used to set nonblocking I/O mode for the specified socket.

Syntax

```
result = fcntl(socket, cmd, status)

int   socket, cmd;
long  result, status;
```

Parameters

result	`0` if `fcntl()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
cmd	Command to get or set socket status. The possible values for *cmd* are: `F_GETFL` Get the socket status. The status value is returned in *status*. `F_SETFL` Set the status to value as specified in *status*. The only status setting currently supported is `O_NONBLOCK`.
status	Specify the socket status. It is a 32-bit data type, each bit of which represents a characteristic of the socket. Setting and unsetting the bit in this parameter sets or unsets the socket characteristic, respectively. For `F_SETFL`, `fcntl()` sets the current status to the value specified in *status*. The currently supported value for *status* is: `O_NONBLOCK` This option designates the socket as nonblocking. A request on a nonblocking socket that cannot complete immediately returns to the caller and sets *errno* to `EAGAIN`. This option affects the following calls: `accept()`, `connect()`, `recv()`, and `send()`. In a nonblocking `connect()` call, the *errno* value returned is set to `EINPROGRESS` instead of `EAGAIN`. Sockets are created in blocking mode by default.

getsockopt()

Returns status of current socket options.

Syntax

```
   result = getsockopt(socket, level, optname, optval, optlen)

   int  result, socket, level, optname;
   char *optval;
   int  *optlen;
```

Parameters

result	`0` if `getsockopt()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
level	The protocol level at which the socket option resides. To specify "socket" level, *level* should be `SOL_SOCKET`. To specify "TCP" level, *level* should be `IPPROTO_TCP`.
optname	Socket option name. The following options are supported for "socket" level (`SOL_SOCKET`) options:

`SO_KEEPALIVE`	(Toggle option) Sets a timer for 90 minutes for connected sockets. After 90 minutes expire, and if the connection has been idle during this period, `SO_KEEPALIVE` forces a transmission every 60 seconds for up to 7 minutes, after which the idle connection is shut down. In summary, `SO_KEEPALIVE` allows an idle period of 97 minutes before connection shutdown. If this option is toggled off, an indefinite idle time is allowed. This option is set by default.
`SO_REUSEADDR`	(Toggle option) Allows local socket address reuse. This allows multiple sockets to be bound to the same local port address. This option modifies the rules used by `bind()` to validate local addresses. `SO_REUSEADDR` allows more than one socket to be bound to the same port number at the same time; however, it only allows one single socket to be actively listening for connection requests on the port number. The host will still check at connection time to be sure any other socket with the same local address and local port does not have the same remote address and remote port. `Connect()` fails if the uniqueness requirement is violated.

`SO_RCVBUF`	Returns the buffer size of a socket's receive socket buffer. The default buffer size is 4096 bytes. A stream socket's buffer size can be increased or decreased only prior to establishing a connection.
`SO_SNDBUF`	Returns the buffer size of a socket's send socket buffer. The default buffer size is 4096 bytes. A stream socket's buffer size can be increased or decreased only prior to establishing a connection.

The following options are supported for "TCP" level (`IPPROTO_TCP`) options:

`TCP_MAXSEG`	Returns the maximum segment size in use for the socket. The value for this option can only be examined, it cannot be set. If the socket is not yet connected, TCP returns a default size of 512 bytes.
`TCP_NODELAY`	(Toggle option) Instructs TCP to send data as soon as it receives it and to bypass the buffering algorithm that tries to avoid numerous small packets from being sent to the network.

`optval` Byte pointer to a variable into which an option value is returned. *`optval`* returns a NULL if the option information is not of interest and not to be passed to the calling process. Although *`optval`* is typed as (`char *`), the value that it points to is not terminated by `\0`.

`optlen` Pointer to a variable containing the maximum number of bytes to be returned by *`optval`*.

On return, it contains the actual number of bytes returned by *`optval`*.

listen()

Sets up a listen queue for the specified socket on the server process and listens for connection requests.

Syntax

```
result = listen(socket, backlog)

int result, socket, backlog;
```

Parameters

result	`0` if `listen()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
backlog	Defines the maximum allowable length of the queue for pending connections. The current valid range for *backlog* is 1 to 5. If any other value is specified, the system automatically assigns the closest value within range. If the queue is greater than the backlog, additional incoming requests will be rejected.

recv()

Receives data from a socket. The `recv()` call may be used by both the server and client processes.

Syntax

```
count= recv(socket, buffer, len, flags)

int  count, socket, len;
char *buffer;
long flags;
```

Parameters

count	Returns the number of bytes actually received. Returns `0` if the remote process has gracefully shut down and there is no more data in the receive buffer. Returns `-1` if the call encounters an error.
socket	Socket descriptor of the local socket receiving data.
buffer	Byte pointer to the data buffer.
len	Maximum number of bytes that will be returned into the buffer referenced by *buffer*. No more than *len* bytes of data are received. If there are more than *len* bytes of data on the socket, the remaining bytes are received on the next `recv()`.
flags	Optional flag options. The currently supported values for *flags* are: `0` — No option. `MSG_PEEK` — Option to preview incoming data. If this option is set on the `recv()` call, any data returned remains in the socket buffer as though it had not been read yet. The next `recv()` call returns the *same data*.

recvfrom()

Receives datagrams from a socket. The `recvfrom()` call may be used by both the server and client processes.

Syntax

```
count= recvfrom(socket, buffer, len, flags, addr, addrlen)

int                  count, socket, len, *addrlen;
char                 *buffer;
long                 flags;
struct sockaddr_in   *addr;
```

Parameters

count	Returns the number of bytes actually received. Returns `-1` if the call encounters an error.
socket	Socket descriptor of the local socket receiving data.
buffer	Byte pointer to the data buffer.
len	Maximum number of bytes that will be returned into the buffer referenced by buffer. No more than *len* bytes of data are received. If the next datagram is larger than *len* bytes, the remaining bytes are discarded.
flags	Optional flag options. The currently supported values for *flags* are: `0` — No option. `MSG_PEEK` — Option to preview incoming data. If this option is set on the `recvfrom()` call, any data returned remains in the socket buffer as though it had not been read yet. The next `recvfrom()` call returns the same data.
addr	Pointer to a structure containing the socket address of the remote socket which sent the data.
addrlen	Length, in bytes, of the returned address structure.

recvmsg()

Receives vectored data on a socket. The `recvmsg()` call may be used by both the server and client processes.

Syntax

```
count= recvmsg(socket, msg, flags)

int            count, socket;
struct msghdr  *msg;
long           flags;
```

Parameters

count	Returns the number of bytes received. Returns `0` if the remote process has gracefully shut down and there is no more data in the receive buffer. Returns `-1` if the call encounters an error.
socket	Socket descriptor of a local socket that is receiving the data.
msg	A pointer to the data structure, `msghdr`, which has two fields called `msg_iov` and `msg_iovlen`. `Msg_iov` is a pointer to an array of data elements, and `msg_iovlen` contains the number of data elements in the array. See "Discussion" below for more information.
flags	Optional flag options. The currently supported values for *flags* are:

`0`	No option.
`MSG_PEEK`	Option to preview incoming data. If this option is set on the `recvmsg()` call, any data returned remains in the socket buffer as though it had not been read yet. The next `recvmsg()` call returns the *same data*.

select()

Provides synchronous socket I/O multiplexing.

Syntax

```
result = select(count, reads, writes, exceptions, timeout)

int             result, count;
fd_set          *reads, *writes, *exceptions;
struct timeval  *timeout;
```

Parameters

result	Returns the number of socket descriptors contained in the `select()` call bitmasks. `-1` means an error has occurred. `0` means the time limit has expired and all the bitmasks are cleared.
count	Specifies the number of sockets for `select()` to examine. `Select()` examines socket descriptors from `0` to `(count-1)`. Currently, users are allowed a maximum of 31 socket descriptors, so the valid range for count is 1 to 31. Since socket descriptors are numbered starting with `0`, callers should specify *count* as their highest socket descriptor `+ 1`. (Note: *count* specifies the *number* of socket descriptors for selection. Hence, a count of 5 means that the `select()` call will examine socket descriptors from 0 through 4.)
reads	Pointer to a bitmask to specify which socket descriptors (from `0` to `count-1`) to select for reading. Set the bitmask to `0` with `FD_ZERO` if no descriptors need to be selected for reads. On return, it contains a pointer to the bitmask specifying which socket descriptors (from `0` to `count-1`) are ready for reading. Use the `FD_SET` macro and `fd_set` variable type to set the socket descriptors for reads before you issue the `select()` call. After issuing `select()`, use `FD_ISSET` to test for the bits in the bitmask. Refer to "Socket Descriptor Utilities" for more information on clearing, setting, and testing the bits in the bitmasks.
writes	Pointer to a bitmask to specify which socket descriptors (from 0 to `count-1`) to select for writing. Set the bitmask to 0 with `FD_ZERO` if no descriptors need to be selected for writes. On return, it contains a pointer to the bitmask specifying which socket descriptors (from `0` to `count-1`) are ready for writing. Use the `FD_SET` macro and `fd_set` variable type to set the socket descriptors for writes before you issue the `select()` call. After issuing `select()`, use `FD_ISSET` to test for the bits in the bitmask. Refer to "Socket Descriptor Utilities" for more information on clearing, setting, and testing the bits in the bitmasks.

exceptions	Pointer to a bitmask to specify which socket descriptor (from `0` to `count-1`) to select for exceptional conditions. Set the bitmask to `0` with `FD_ZERO` if no descriptors need to be selected for exceptions. On return, it contains a pointer to the bitmask specifying which socket descriptors (from `0` to `count-1`) have an exceptional condition pending. Use the `FD_SET` macro and `fd_set` variable type to set the socket descriptors for exceptions before you issue the `select()` call. After issuing `select()`, use `FD_ISSET` to test for the bits in the bitmask. Refer to "Socket Descriptor Utilities" for more information on clearing, setting, and testing the bits in the bitmasks. The currently supported condition is when connections get terminated.
timeout	Pointer to the `timeval` structure which specifies the interval in which to examine the socket descriptors. The `timeval` structure contains two fields: `tv_sec` and `tv_usec`. The *timeout* parameter works as follows: If both `tv_sec` and `tv_usec` are `0`, `select()` returns immediately after checking the descriptors. If either `tv_sec` or `tv_usec` is non-zero, then `select()` returns when one of the specified descriptors is ready for I/O, but `select()` does not wait beyond the specified amount of time (in number of seconds and microseconds). If the `timeval` pointer itself is 0, then `select()` waits indefinitely and returns only when one of the selected descriptors is ready for I/O.

send()

Sends data on a socket. The `send()` call may be used by both the server and client processes.

Syntax

```
count = send(socket, buffer, len, flags)

int  count, socket, len;
char *buffer;
long flags;
```

Parameters

count	Returns the number of bytes actually sent. Returns `-1` if the call encounters an error.
socket	Socket descriptor of a local socket that is sending the data.
buffer	Byte pointer to a buffer which contains the data to be sent.
len	Number of bytes that need to be sent from the data buffer.
	In blocking mode, there is no restriction on the size of data to be sent except for that imposed by the system, which currently is 32767 bytes.
	In nonblocking mode, if the data is too long to pass atomically through the underlying protocol, the message is not transmitted, `-1` is returned and *errno* is set to `EMSGSIZE`.
flags	Optional flag options. Currently there are no supported options.

sendmsg()

Sends vectored data on a socket. The `sendmsg()` call may be used by both the server and client processes.

Syntax

```
count = sendmsg(socket, msg, flags)

int           count, socket;
struct msghdr msg;
long          flags;
```

Parameters

count	Returns the number of bytes actually sent.
	Returns `-1` if the call encounters an error.
socket	Socket descriptor of a local socket that is sending the data.
msg	Pointer to a `msghdr` structure, which has two fields called `msg_iov` and `msg_iovlen`. `Msg_iov` is a pointer to an array of data elements, and `msg_iovlen` contains the number of data elements in the array.
flags	Optional flag options. Currently there are no supported options.

sendto()

Sends data on a socket. The `sendto()` call may be used by both the server and client processes.

Syntax

```
count = sendto(socket, buffer, len, flags, addr, addrlen)

int                  count, socket, len, addrlen;
char                 *buffer;
long                 flags;
struct sockaddr_in   *addr;
```

Parameters

count	Returns the number of bytes actually sent. Returns `-1` if the call encounters an error.
socket	Socket descriptor of a local socket that is sending the data.
buffer	Byte pointer to a buffer which contains the data to be sent.
len	Number of bytes that need to be sent from the data buffer. The size of data that can be sent is limited to 32767 bytes. However, the HP 1000 cannot receive UDP datagrams larger than 9216 bytes.
flags	Currently there are no supported options.
addr	Pointer to a structure containing the address of the remote socket to which the data will be sent. The socket address should be of *sockaddr_in* type.
addrlen	Length, in bytes, of the address structure specified by *addr* (for example, length of structure *sockaddr_in*, which is 16 bytes). *Addrlen* should be at least 16 bytes.

setsockopt()

Sets socket options.

Syntax

```
result = setsockopt(socket, level, optname, optval, optlen)

int  result, socket, level, optname, optlen;
char *optval;
```

Parameters

result	`0` if `setsockopt()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
level	The protocol level at which the socket option resides. To specify “socket” level, *level* should be `SOL_SOCKET`. To specify “TCP” level, *level* should be `IPPROTO_TCP`.
optname	Socket option name. The following options are supported for “socket” level (`SOL_SOCKET`) options:

`SO_KEEPALIVE`	(Toggle option) Sets a timer for 90 minutes for connected sockets. After 90 minutes expire, and if the connection has been idle during this period, `SO_KEEPALIVE` forces a transmission every 60 seconds for up to 7 minutes, after which the idle connection is shut down. In summary, `SO_KEEPALIVE` allows an idle period of 97 minutes before connection shutdown. If this option is toggled off, an indefinite idle time is allowed. This option is set by default.
`SO_REUSEADDR`	(Toggle option) Allows local socket address reuse. This allows multiple sockets to be bound to the same local port address. This option modifies the rules used by `bind()` to validate local addresses. `SO_REUSEADDR` allows more than one socket to be bound to the same port number at the same time; however, it only allows one single socket to be actively listening for connection requests on the port number. The host will still check at connection time to be sure any other socket with the same local address and local port does not have the same remote address and remote port. `Connect()` fails if the uniqueness requirement is violated.

setsockopt()

`SO_RCVBUF`	Changes the buffer size of a socket's receive socket buffer. The default buffer size is 4096 bytes. The maximum buffer size is 32766 bytes. A stream socket's buffer size can be increased or decreased only prior to establishing a connection.
`SO_SNDBUF`	Changes the buffer size of a socket's send socket buffer. The default buffer size is 4096 bytes. The maximum buffer size is 32766 bytes. A stream socket's buffer size can be increased or decreased only prior to establishing a connection.

The following options are supported for "TCP" level (`IPPROTO_TCP`) options:

`TCP_MAXSEG`	Returns the maximum segment size in use for the socket. The value for this option can only be examined, it cannot be set. If the socket is not yet connected, TCP returns a default size of 512 bytes.
`TCP_NODELAY`	(Toggle option) Instructs TCP to send data as soon as it receives it and to bypass the buffering algorithm that tries to avoid numerous small packets from being sent over the network.

`optval` Byte pointer to a value or boolean flag for the specified option.

(Since *`optval`* is a byte pointer, Pascal and FORTRAN users should use `ByteAdrOf` to get the byte address of the option value.) Although *`optval`* is a byte pointer, the value itself is not terminated by a `\0`.

`optlen` Size, in bytes, of *`optval`*.

shutdown()

Shuts down a socket. This call may be used by either the server or client process.

Syntax

```
result = shutdown(socket, how)

int result, socket, how;
```

Parameters

result	`0` if `shutdown()` is successful. `-1` if a failure occurs.
socket	Socket descriptor of local socket to be shut down.
how	Method of shutdown, as follows:

`0`	Disallows further receives. Once the socket has been shut down for receives, all further `recv()` calls return `-1`, with *errno* set to `ESHUTDOWN`.
`1`	Disallows further sends. Once the socket has been shut down for sends, all further `send()` calls return `-1`, with *errno* set to `ESHUTDOWN`.
`2`	Disallows further sends and receives.

socket()

Creates a socket, an endpoint for communication, and returns a socket descriptor for the socket. This must be the first BSD IPC call used in the process. Both server and client processes need to create a socket with the `socket()` call.

Syntax

```
socket = socket(af, type, protocol)

int socket, af, type protocol;
```

Parameters

socket	Socket descriptor for the newly-created socket. It is an integer with a valid range of `0` to `30`. This socket descriptor is used in subsequent BSD IPC calls to reference this socket. If the call fails, a `-1` is returned in *`socket`* and the global variable *`errno`* contains the error code.
af	Address family for the socket being created. It must be set to `AF_INET`, for Internet address family. The address family defines the address format used in socket operations. The `AF_INET` address family uses an address structure (`sockaddr_in`) of 16 bytes. Refer to "Address Family Type" in Appendix D of the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for more information.
type	Type of socket being created. It must be set to `SOCK_STREAM`. The socket type specifies the semantics of communication for the socket. A `SOCK_STREAM` type provides sequenced, reliable, two-way, connection-based bytes streams. Refer to "Socket Type" in Appendix D of the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for more information.
protocol	Underlying protocol to be used. `0` causes the system to choose a protocol type to use. The default is `tcp`.

BSD IPC Utilities

BSD IPC utilities are used for the following purposes:

- Manipulate and return information on the following database files: `/etc/hosts`, `/etc/networks`, `/etc/protocols`, and `/etc/services`.
- Obtain the socket address of the local and peer sockets.
- Manipulate Internet (IP) addresses and ASCII strings that represent IP addresses in Internet "dot" notation.
- Convert bytes from network order to host order and vice versa. (HP 1000, HP 9000, and TCP/IP protocols all use network order. These functions are provided for portability.)

The utilities are summarized here in alphabetical order for easy referencing.

Special Considerations

In order to successfully use the BSD IPC utilities, you must be aware of the following:

- The `/ETC` directory must be created before you run the BSD IPC utilities.
- Most of the utilities return pointers to structures that are dynamically allocated. If any of these functions that allocate dynamic memory are called repeatedly without freeing the allocated memory, they will eventually fail and return a null pointer.

  The BSD IPC utility functions that allocate dynamic memory include:

```
gethostbyaddr  getnetbyaddr  getprotobyname    getservbyname
gethostbyname  getnetbyname  getprotobynumber  getservbyport
gethostent     getnetent     getprotent        getservent
                                               inet_ntoa
```

In order to release space that has been dynamically allocated by the above utility functions, use `free()` from the standard C library (generally found in `HPC.LIB`).

endhostent()

Closes the `/etc/hosts` file.

```
result = endhostent()

int  result
```

result	`0` if the call is successful. `-1` if a failure occurs.

endnetent()

Closes the `/etc/networks` file.

```
   result = endnetent()

   int  result
```

result	`0` if the call is successful. `-1` if a failure occurs.

endprotoent()

Closes the `/etc/protocols` file.

```
   result = endprotoent()

   int  result
```

result	`0` if the call is successful. `-1` if a failure occurs.

endservent()

Closes the `/etc/services` file.

```
   result = endservent()

   int  result
```

result	`0` if the call is successful. `-1` if a failure occurs.

gethostbyaddr()

Returns host information on the host with the specified IP address.

```
   host = gethostbyaddr(addr, len, type)

   struct hostent *host;
   char           *addr;
   int            len, type;
```

host	Pointer to `hostent` structure that contains the host information. The `hostent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
addr	Character pointer to a variable that contains the IP address of the host. The IP address must be in network order (that is, bytes ordered from left to right).
len	Number of bytes of an IP address.
type	The type of socket address family used. Must be set to `AF_INET`.

gethostbyname()

Returns host information on the host with the specified host name.

```
host = gethostbyname(name)

host struct    *host;
char           *name;
```

host	Pointer to the `hostent` structure that contains host information. The `hostent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
name	Pointer to string that contains the name of the host about whom you need to obtain information. Terminate the string with the `\0` character.

gethostent()

Reads the next line of the `/etc/hosts` file and returns the host information.

```
host = gethostent()

struct hostent *host;
```

host	Pointer to a `hostent` structure containing host information. The `hostent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.

getlocalname()

Returns the name of the host/local system.

```
error = getlocalname(hostname)

character*(*)          hostname;
integer*2              error;
```

error	Is an integer that returns 0 for success or non-zero if the network is down.
hostname	Returns a FORTRAN character string for the local node name. The *hostname* string is limited to 50 characters.

getnetbyaddr()

Returns network information on the specified network number.

```
network = getnetbyaddr(net, type)

struct netent *network;
long          net;
int           type;
```

network	Pointer to a `netent` structure that contains network information returned by `getnetbyaddr()`. The `netent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
net	Network number from which to get network information.
type	The socket address family type. It must be set to `AF_INET`.

getnetbyname()

Returns network information on the specified network name.

```
network = getnetbyname(name)

struct netent  *network;
char           *name;
```

network	Pointer to a `netent` structure that contains network information returned by `getnetbyname()`. The `netent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
name	Pointer to string that contains the network name from which to get network information. Terminate the name string with the character `\0`.

getnetent()

Reads the next line of the `/etc/networks` file and returns the network information.

```
network = getnetent()

struct netent *network;
```

network	Pointer to a `netent` structure that contains network information returned by `getnetent()`. The `netent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.

getpeername()

Returns the socket address of the peer socket connected to the specified local socket.

```
   result = getpeername(socket, addr, addrlen)

   int                 result, socket, *addrlen;
   struct sockaddr_in *addr;
```

result	`0` if the call is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
addr	Pointer to an address structure that contains the socket address of the peer socket that is connected to *socket*. The address structure should be of `sockaddr_in` type. Refer to "Preparing Socket Address Variables" in the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for details.
addrlen	Pointer to an integer variable that contains the length, in bytes, of the address structure specified by *addr* (for example, length of structure `sockaddr_in`, which is 16 bytes). On return, pointer to an integer variable that contains the actual length of the peer socket address. If *addr* does not point to enough space to contain the whole socket address of the peer socket, only the first *addrlen* bytes of the address are filled in the structure pointed to by *addr*.

getprotobyname()

Returns protocol information on the specified protocol name.

```
   protocol = getprotobyname(name)

   struct protoent  *protocol;
   char             *name;
```

protocol	Pointer to a `protoent` structure that contains the protocol information returned by `getprotobyname()`. The `protoent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
name	Pointer to string that contains the protocol name from which to get protocol information. It can be either an official protocol name or an alias. Terminate the string with the character `\0`.

getprotobynumber()

Returns protocol information on the specified protocol number.

```
protocol = getprotobynumber(protonumb)

struct protoent  *protocol;
int               protonumb;
```

protocol	Pointer to a `protoent` structure that contains protocol information returned by `getprotobynumber()`. The `protoent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
protonumb	Protocol number from which to get protocol information.

getprotoent()

Reads the next line of the `/etc/protocols` file and returns the protocol information.

```
protocol = getprotoent()

struct protoent *protocol;
```

protocol	Pointer to a `protoent` structure that contains protocol information returned by `getprotoent()`. The `protoent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.

getservbyname()

Returns service information on the specified service name.

```
   service = getservbyname(name, proto)

   struct servent  *service;
   char            *name, *proto;
```

service	Pointer to a `servent` structure that contains service information returned by `getservbyname()`. The `servent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
name	Pointer to string that contains the service name from which to get information on the service. It can be either an official service name or an alias. Terminate the string with the character `\0`.
proto	Pointer to string that contains the name of the transport protocol to use when contacting the service. Use "`tcp`" or `0` if TCP is the only protocol for the service. (Remember to terminate the string with the character `\0`.)

getservbyport()

Returns service information on the specified port number.

```
   service = getservbyport(port, proto)

   struct servent  *service;
   int             port;
   char            *proto;
```

service	Pointer to a `servent` structure that contains service information returned by `getservbyname()`. The `servent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.
port	Port number from which to get information on the service.
proto	Pointer to string that contains the name of the transport protocol to use when contacting the service. Use "`tcp`" or `0` if TCP is the only protocol for the service. Terminate string with character `\0`. Set this value to NULL if you do not want to specify any specific protocol.

getservent()

Reads the next line of the `/etc/services` file and returns information on the service.

```
   service = getservent()

   struct servent  *service;
```

service	Pointer to a `servent` structure that contains service information returned by `getservent()`. The `servent` structure is defined in the include files `<netdb.h>`, `SOCKET.PASI`, and `SOCKET.FTNI`, for C, Pascal, and FORTRAN programs, respectively.

getsockname()

Returns the socket address of the specified local socket.

```
   result = getsockname (socket, addr, addrlen)

   int                 result, socket, *addrlen;
   struct sockaddr_in  *addr;
```

result	`0` if the call is successful. `-1` if a failure occurs.
socket	Socket descriptor of a local socket.
addr	Pointer to a socket address variable to contain the address of the specified socket. The socket address should be of `sockaddr_in` type. On return, the socket address structure will contain the local socket address information.
addrlen	Pointer to an integer variable that contains the length, in bytes, of the address structure specified by *addr* (for example, length of structure `sockaddr_in`, which is 16 bytes). On return, it is the pointer to an integer that contains the actual length of the socket address returned in *addr*. If *addr* does not point to enough space to contain the whole address of the socket, only the first *addrlen* bytes of the address are returned.

htonl()

Converts a 32-bit quantity from host order to network order.

```
   netlong = htonl(hostlong)

   u_long netlong, hostlong;
```

netlong	32-bit integer in network order, returned by `htonl()`.
hostlong	32-bit integer in host order.

htons()

Converts a 16-bit quantity from host order to network order.

```
netshort = htons(hostshort)

u_short netshort, hostshort;
```

netshort	16-bit integer in network order, returned by `htons()`.
hostshort	16-bit integer in host order.

inet_addr()

Interprets character strings representing numbers in the Internet standard “dot” notation, and returns numbers suitable for use as Internet (IP) addresses.

```
IPaddr = inet_addr(string)

struct in_addr IPaddr;
char           *string;
```

IPaddr	Internet (IP) address returned by `inet_addr()`.
string	Pointer to a character string representing numbers expressed in the Internet standard “dot” notation, such as: “192.41.233.2”. Terminate the string with the character `\0`.

inet_lnaof()

Breaks apart an IP address and returns the node address portion of the IP address.

```
node = inet_lnaof(IPaddr)

u_long         node;
struct in_addr IPaddr;
```

node	Node address portion of the IP address returned by `inet_lnaof()`.
IPaddr	IP address of a host. IP addresses for the Internet family are stored in an address variable of type `in_addr`.

inet_makeaddr()

Constructs an Internet (IP) address from an Internet network address and a local node address.

```
   IPaddr = inet_makeaddr(net, node)

   struct in_addr  IPaddr;
   u_long          net, node;
```

IPaddr	Internet (IP) address constructed from the specified network address and node address. IP addresses for the Internet family are stored in an address variable of type `in_addr`. Refer to “Preparing Socket Addresses” in *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for more information on `in_addr`.
net	Internet network number that defines the network on which a node resides. The network number makes up a portion of an IP address.
node	Internet node address that defines the address of a node within a network. The node address makes up a portion of an IP address.

inet_netof()

Breaks apart an IP address and returns the network address portion of the IP address.

```
   network = inet_netof(IPaddr)

   u_long          network;
   struct in_addr  IPaddr;
```

network	Network address portion of the IP address returned by `inet_netof()`.
IPaddr	IP address of the local host. IP addresses for the Internet family are stored in an address variable of type `in_addr`.

inet_network()

Interprets character strings representing numbers in the Internet standard “dot” notation, and returns numbers suitable for use as Internet network numbers.

```
   network = inet_network(string)

   struct in_addr  network;
   char            *string;
```

network	Internet network number returned by `inet_network()`.
string	Pointer to character string representing numbers expressed in the Internet standard “dot” notation, such as: “192.41.233.2”. Terminate the string with the character `\0`.

inet_ntoa()

Takes an Internet (IP) address and returns an ASCII string representing the address in "dot" notation.

```
   string = inet_ntoa(IPaddr)

   char            *string;
   struct in_addr  IPaddr;
```

string	Pointer to character string representing numbers expressed in the Internet standard "dot" notation, such as: "192.41.233.2". Terminate the string with character `\0`. See "Internet Dot Notation" in Appendix D of the *BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000* for more information on dot notation.
IPaddr	Internet (IP) address. IP addresses for the Internet family are stored in an address variable of type `in_addr`.

ntohl()

Converts a 32-bit quantity from network order to host order.

```
   hostlong = ntohl(netlong)

   u_long hostlong, netlong;
```

hostlong	32-bit integer in host order, returned by `ntohl()`.
netlong	32-bit integer in network order.

ntohs()

Converts a 16-bit quantify from network order to host order.

```
   hostshort = ntohs(netshort)

   u_short hostshort, netshort;
```

hostshort	16-bit integer in host order, returned by `ntohs()`.
netshort	16-bit integer in network order.

sethostent()

Opens and rewinds the `/etc/hosts` file.

```
   result = sethostent (stayopen)

   int  result, stayopen;
```

result	`0` if the call is successful. `-1` if a failure occurs.
stayopen	A zero value closes the `/etc/hosts` file after each call to the file by one of the following calls: `gethostbyaddr()`, `gethostbyname()`, and `gethostent()`. A non-zero value leaves the `/etc/services` file open after a `gethostbyaddr()`, `gethostbyname()`, or `gethostent()` call. This allows the next `gethostent()` to read from the next line of the `/etc/hosts` file rather than from the beginning of the file.

setnetent()

Opens and rewinds the `/etc/networks` file.

```
   result = setnetent (stayopen)

   int  result, stayopen;
```

result	`0` if the call is successful. `-1` if a failure occurs.
stayopen	A zero value closes the `/etc/networks` file after each call to the file by one of the following calls: `getnetbyaddr()`, `getnetbyname()`, and `getnetent()`. A non-zero value leaves the `/etc/services` file open after a `getnetbyaddr()`, `getnetbyname()`, or `getnetent()` call. This allows the next `getnetent()` to read from the next line of the `/etc/networks` file rather than from the beginning of the file.

setprotoent()

Opens and rewinds the `/etc/protocols` file.

```
   result = setprotoent (stayopen)

   int  result, stayopen;
```

result	`0` if the call is successful. `-1` if a failure occurs.
stayopen	A zero value closes the `/etc/protocols` file after each call to the file by one of the following calls: `getprotobyname()`, `getprotobynumber()`, and `getprotoent()`. A non-zero value leaves the `/etc/protocols` file open after a `getprotobyname()`, `getprotobynumber()`, or `getprotoent()` call. This allows the next `getprotoent()` to read from the next line of the `/etc/protocols` file rather than from the beginning of the file.

setservent()

Opens and rewinds the `/etc/services` file.

```
   result = setservent (stayopen)

   int  result, stayopen;
```

result	`0` if the call is successful. `-1` if a failure occurs.
stayopen	A zero value closes the `/etc/services` file after each call to the file by one of the following calls: `getservbyname()`, `getservbyport()`, and `getservent()`. A non-zero value leaves the `/etc/services` file open after a `getservbyname()`, `getservbyport()`, or `getservent()` call. This allows the next `getservent()` to read from the next line of the `/etc/services` file rather than from the beginning of the file.

Socket Descriptor Utilities

BSD IPC socket descriptor utilities operate on socket descriptor bitmasks. Socket descriptor bitmasks are used by the `select()` call to specify which sockets are ready for reading, writing, or have exceptional conditions pending.

The socket descriptor utilities are summarized here in alphabetical order.

Note The bitmasks are stored in a special data type defined as `fd_set`, which is defined in the header files for C (`types.h`), Pascal (`SOCKET.PASI`), and FORTRAN (`SOCKET.FTNI`).

FD_CLR()

Clears the specified socket descriptor's bit in the bitmask.

```
   FD_CLR (socket, bitmask)

   int     socket;
   fd_set  *bitmask;
```

socket	Socket descriptor of a local socket.
bitmask	Pointer to a variable of `fd_set` type which contains the bitmask of the socket descriptors.

FD_ISSET()

Tests whether the specified socket descriptor's bit is set in the specified bitmask.

```
   result = FD_ISSET (socket, bitmask)

   int     result, socket;
   fd_set  *bitmask;
```

result	Result of `FD_ISSET()` call. *result* = `1` means the bit is set for the specified socket descriptor, *socket*. Otherwise, *result* = `0`.
socket	Socket descriptor of a local socket.
bitmask	Pointer to a `fd_set` variable type which contains the bitmask of the socket descriptors.

FD_SET()

Sets the specified socket descriptor's bit in the bitmask.

```
   FD_SET (socket, bitmask)

   int     socket;
   fd_set  *bitmask;
```

socket	Socket descriptor of a local socket.
bitmask	Pointer to a variable of type `fd_set` which contains the bitmask of the socket descriptors.

FD_ZERO()

Clears the entire bitmask.

```
   FD_ZERO (bitmask)

   fd_set  *bitmask;
```

bitmask	Pointer to a variable of type `fd_set` which contains the bitmask of the socket descriptors.

Error Messages

Errno is a standard error variable used in UNIX* programming. For portability, the C library (`HPC.LIB`) also returns error values in a global variable called *errno*. The following is a list of the error messages for HP 1000 BSD IPC.

Table 4-3. Error Messages

Value of *errno*	Error Mnemonic	Meaning
1	[ENFILE]	Currently there are no resources available.
13	[EINVAL]	One of the following occurred: The value of a specified parameter is invalid. The socket is not a BSD IPC socket. The socket has already been shut down. The socket is not ready to accept connections yet. A `listen()` call must be done before an `accept()` call. The socket is already bound to an address.
202	[EAGAIN]	Nonblocking I/O is enabled and: 1. for `accept()`, no connection is present to be accepted. 2. for `send()`, the socket does not have space to accept any data at all.
203	[EEFAULT]	For `getsockopt()` and `setsockopt()`, the *optval* or *optlen* parameter is not valid.
204	[EMFILE]	The maximum number of socket descriptors for this process are already currently open. This could happen since sockets need to be created for internal use.
205	[EPIPE]	An attempt was made to send on a socket whose connection has been shut down by the remote peer process.
215	[EMSGSIZE]	In nonblocking mode, the socket requires that messages be sent atomically, and the message size exceeded the outbound buffer size.
216	[ENOTSOCK]	The socket descriptor, *socket*, is not a valid socket descriptor.
220	[ENOPROTOOPT]	The requested socket option is currently not set.
221	[EPROTONOSUPPORT]	The specified protocol is not supported.
222	[ESOCKTNOSUPPORT]	The specified socket type is not supported in this address family.
223	[EOPNOTSUPP]	The socket descriptor, *socket*, does not support this call or a parameter in this call.

*UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Value of *errno*	Error Mnemonic	Meaning
225	[EAFNOSUPPORT]	Addresses in the specified address family cannot be used with this socket.
226	[EADDRINUSE]	The specified address is already in use.
227	[EADDRNOTAVAIL]	The specified address is invalid or not available.
232	[ECONNRESET]	Connection has been aborted by the remote process.
233	[ENOBUFS]	No buffer space is available. The call cannot be completed.
234	[EISCONN]	The socket is already connected.
235	[ENOTCONN]	The socket has not been connected yet.
236	[ESHUTDOWN]	The network software on the system is not running. Or the socket has already been shut down for send or receive.
238	[ETIMEDOUT]	Connection establishment timeout without establishing a connection.
239	[ECONNREFUSED]	The attempt to connect was rejected by the server.
240	[EREMOTERELEASE]	The remote side has done a send shutdown; hence, there will be no more data to receive.
241	[EHOSTDOWN]	The network software on the local host is not running.
242	[EHOSTUNREACH]	The network software was unable to determine a route to the destination host.
245	[EINPROGRESS]	Nonblocking I/O is enabled and the connection has been initiated. This is not a failure. Use `select()` to find out when the connection is complete.
299	[EINTERR]	This error requires HP notification.

5

Network File Transfer

Table of Contents

File Copying Formats

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

Transparent Format is invoked by default when files are copied between NS-ARPA/1000 systems. Transparent Format does not alter a file's attributes, but simply copies the file.

You must invoke Interchange Format explicitly by specifying one or more of the Interchange Format options. (These options are explained later in this section.) Interchange Format is useful when you want to change certain source file attributes, such as record length, at the target file.

Interactive Network File Transfer

You can use NFT interactively by running the program DSCOPY.

```
         [ ,copydescriptor ]
  DSCOPY [ ,dscopycommand  ]
```

copydescriptor	A copy descriptor. May be a maximum of 256 characters long. The syntax of *copydescriptor* is provided below.
dscopycommand	A DSCOPY command. The DSCOPY commands are described later in this section.

Copy Descriptor

The copy descriptor allows you to specify the files or directories you wish to copy.

```
                              { ΔTOΔ }
  sfile[[slogon]][>snode]     { ,    }  tfile[[tlogon]][,option][,option]...]
```

sfile	The source file; the name of the file to be copied.
[*slogon*]	The logon and password, if any, at the node where the source file resides. Must be enclosed in brackets (`[ ]`). This parameter is *required* if the source node is a remote multiuser HP 1000. *Default*: If *slogon* is omitted and the source node is the local node, the account under which DSCOPY was scheduled is used.
>*snode*	The name of the source node. Must be preceded by ">." Must be in the form *node*[.*domain*[.*organization*]]. *Default*: You may omit the organization, organization and domain, or all parts of the node name. If the organization, or organization and domain, are omitted, the local organization and/or domain will be used. If the entire node name is omitted, it will default to the local node.
tfile	The target file; the name the source file will acquire at the target node.

[*tlogon*]		The logon and password, if any, at the target node. Must be enclosed in brackets (`[ ]`). This parameter is *required* if the target node is a remote multiuser HP 1000. *Default*: If `tlogon` is omitted and the target node is the local node, the account under which DSCOPY was scheduled is used.
>*tnode*		The name of the target node. Must be preceded by "`>`." Must be in the form *node*[.*domain*[.*organization*]]. *Default*: You may omit the organization, organization and domain, or all parts of the node name. If the organization, or organization and domain, are omitted, the local organization and/or domain will be used. If the entire node name is omitted, it will default to the local node name.
option		May be one or more of the options described below; there is no limit to the number of options you can specify. Each option must be separated by a comma, semicolon, or space, but different delimiters cannot be used in the same copy descriptor. If conflicting options are given (for example, `ASCII` and `BINARY`), DSCOPY will issue a warning and the *last option given will take precedence*. The first eight options described below cause Interchange Format to be used.
	`AS[CII]`	Specifies that records contain printable ASCII characters and that spaces should be used as padding when creating fixed length records. This option may be used in conjunction with the `STRIP` option to indicate that spaces should be stripped from the ends of records. *Default*: If the source file is ASCII, the target file will be ASCII.
	`BI[NARY]`	Specifies that records contain binary information and that null characters (numeric zeros) should be used as padding when creating fixed length records. This option may be used in conjunction with the `STRIP` option to indicate that nulls should be stripped from the ends of records. *Default*: If the source file is binary, the target file will be binary.
	`FI[XED]`	Specifies that source file records should be formed into fixed length records. (Record size can be specified using the `RSIZE` option and the type of padding used can be specified using the `ASCII` or `BINARY` options.) *Default*: For HP 1000 type 1 and 2 source files, the target file will have fixed length records. For other types of HP 1000 files, the target file will have variable length records.

`FS[IZE]=` *`filesize`*	Specifies how much space (*`filesize`*) to allocate for the target file. If the target file has fixed length records, *`filesize`* is in records. If the target file has variable length records, *`filesize`* is the number of maximum size records. This option can be used instead of the HP 1000 file descriptor size parameter to specify the size of an HP 1000 target file. *Default*: The target file will be the same size as the source file.
`IN[TERCHANGE]`	Overrides the default copy format and causes the file or files to be copied using Interchange Format. *Default*: DSCOPY will use Transparent Format. Interchange Format is also used if the `ASCII`, `BINARY`, `FIXED`, `FSIZE`, `RSIZE`, `STRIP`, or `VARIABLE` options are specified.
`RS[IZE]=` *`recordsize`*	Specifies the record size (*`recordsize`*) in bytes. If fixed length records are being produced, *`recordsize`* is the size of each record. If variable length records are being produced, *`recordsize`* limits the size of the largest record. DSCOPY will issue a warning if it must truncate records in order to execute this option. *Default*: The target file will have the same record size as the source file.
`ST[RIP]`	Strips any record padding from the ends of records. You can use this option to create variable length records from fixed length records. (Also see the `VARIABLE` option.) The type of padding to strip is based on the type of the source file. For HP 1000 type 4 files, spaces are stripped. In other HP 1000 file types, null characters are stripped. You can use this option in conjunction with `RSIZE` to truncate records. Records will be truncated before padding is stripped. *Default*: Padding is not stripped.
`VA[RIABLE]`	Specifies that source file records should be formed into variable length records. The maximum size of a variable length record may be given using the `RSIZE` option. *Default*: For HP 1000 type 1 or 2 files, the target file will have fixed length records. For all other HP 1000 file types, the target file will have variable length records.

The next four options can be used when a file is copied using Transparent Format. They can also be used in conjunction with Interchange Format options.

`MO[VE]`	Purges the source file after it has been successfully copied to the target system. DSCOPY will issue a warning if the file cannot be purged. You must have proper access rights, including any security code, to purge the file. If a directory is copied, the files within the directory and any subdirectories will be purged, but the directory and subdirectories will not be purged. *Default*: The source file is not purged.
`QU[IET]`	Suppresses the printing of warnings and file names to the list file or device. Error messages cannot be suppressed. *Default*: Warnings, file names, and error messages are printed to the list file.
`RE[PLACE]`	If the target file exists, this option causes it to be purged and a new file created by the same name. The original file is purged only after the new file is copied successfully to the target system. *Default*: The target file is not replaced and an error message is returned if it already exists.
`SI[LENT]`	Suppresses the printing of warnings, file names, and error messages to the list file or device. Same as the `QUIET` option, except that error messages are also suppressed. *Default*: Warnings, file names, and error messages are printed to the list file.

Interrupting the Copy Process

To interrupt DSCOPY, hit any key to enter breakmode and then type one of the following commands: `A` to Abort, `C` to Cancel, `S` for Status information, or `H` for Help.

- *`Abort`*. Terminates DSCOPY and saves the portion of the target file that has been created thus far. You can also use the `A` command to exit an active transfer file and return control to the scheduling terminal. Although you can abort a copy descriptor at any time, the target file may be in an inconsistent state if aborted prematurely.
- *`Cancel`*. Terminates DSCOPY and purges the target file. You can also use the `C` command to exit an active transfer file and return control to the scheduling terminal. You can cancel a copy descriptor at any time.
- *`Status`*. Indicates the percentage of the file that has been transmitted to the target node; not all of this data may actually have been *received* at the target. This number is not exact and should be considered an estimate.
- *`Help`*. Provides an explanation of the `A`, `C`, and `S` commands.

DSCOPY Commands

+CLEAR

Clears all currently active copy descriptor defaults that have been set with the `+DEFAULT` command.

```
    +CL[EAR]
```

+DEFAULT

Sets defaults for portions of subsequently issued copy descriptors.

```
    +DE[FAULT],copydescriptor
```

copydescriptor	A copy descriptor.

+ECHO

Causes commands to be echoed, or not echoed, to the list file or device.

```
            ┌        ┐
    +EC[HO] │ ,ON    │
            │ ,OFF   │
            └        ┘
```

`ON`	Causes commands to be echoed to the list file or device. This is the default if `+ECHO` is issued without a parameter.
`OFF`	Turns echo off. (DSCOPY does not echo commands to the list file or device by default.)

+EX

Exits DSCOPY.

```
    +EX
```

+LL

Changes the list file or device.

```
    +LL,lfiledev
```

lfiledev	The name of a list file or the LU of a device.

+RU

Runs a program from within DSCOPY.

```
    +RU,progname
```

progname	The name of the program to be run.

+SHOW

Shows all currently active copy descriptor defaults set with the `+DEFAULT` command.

```
    +SH[OW]
```

+TRANSFER

Transfers control to a command file or device.

```
    +TR[ANSFER],cmdfiledev
```

cmdfiledev	The name of the command file or the LU of the device that will have control.

+WD

Displays or changes the current working directory.

```
    +WD[,directoryname]
```

directoryname	The name of the new working directory. May be a subdirectory.

? (HELP)

Requests help information for any command or copy descriptor option.

```
    ?[,commandoption]
```

commandoption	Any DSCOPY command or copy descriptor option.

Programmatic Network File Transfer

Two calls are provided to copy files programmatically: `DscopyBuild` and `Dscopy`. The `DscopyBuild` call creates a copy descriptor that can be used by the `Dscopy` call to copy the file or files specified.

DSCOPY

Copies a file or files.

```
   DSCOPY(builtdescriptor,result)
```

builtdescriptor	*Character array (FORTRAN); String (PASCAL).* A buffer of variable length that contains a copy descriptor or a DSCOPY command. The *builtdescriptor* parameter may be created programmatically by calling `DscopyBuild`. (`DscopyBuild` is described later in this section.)
result	*Array of 16-bit integers.* A five-word array returned by `Dscopy`. The first word contains the number of errors that occurred while the file, or files, were being copied. The second word returns the error code, if any; zero is returned if the file or files are copied successfully. (If multiple files are copied, the error code is the result of the last attempted file copy.) The last three words of this parameter are reserved for future use. The `DSCOPY` error codes are described in the *NS/1000 Error Message and Recovery Manual*.

Note If your program is written in Pascal/1000, Version 2, you must set the `FIXED_STRING` option before declaring `Dscopy`. In addition, the routines `SetStrLen` and `StrMax` must be used to initialize the *builtdescriptor* string. `FIXED_STRING`, `SetStrLen` and `StrMax` are described in the *Pascal/1000 Reference Manual*. If your program is written in Pascal/1000, Version 1, you must use the routine `StrDsc` to convert the *builtdescriptor* string to a format that can be processed by both the calling program and DSCOPY. This routine is described in the *RTE-A Programmer's Reference Manual*.

DSCOPYBUILD

Builds a copy descriptor to be used in the `Dscopy` call.

```
  DSCOPYBUILD(builtdescriptor,sfile,slogon[,snode],tfile[,tlogon]
           [,tnode],options,rsize,fsize)
```

builtdescriptor	*Character array (FORTRAN); String (PASCAL).* The returned copy descriptor to be used in the `Dscopy` call. Will be blank-padded if less than the length declared.
sfile	*Character array (FORTRAN); String (PASCAL).* The source file; the name of the file to be copied.
slogon	*Character array (FORTRAN); String (PASCAL).* The logon and password, if any, at the node where the source file resides. Do *not* enclose in brackets. This parameter is *required* if the source node is a remote multiuser HP 1000. *Default*: If this parameter omitted and the source node is the local node, the account under which the program is running is used.
snode	*Character array (FORTRAN); String (PASCAL).* The name of the source node. Must be in the form *node*[.*domain*[.*organization*]]. *Default*: You may omit the organization, organization and domain, or all parts of the node name. If the organization, or organization and domain, are omitted, the local organization and/or domain will be used. If the entire node name is omitted, it will default to the local node name.
tfile	*Character array (FORTRAN); String (PASCAL).* The target file; the name the source file will acquire at the target node.
tlogon	*Character array (FORTRAN); String (PASCAL).* The logon and password, if any, at the target node. Do *not* enclose in brackets. This parameter is *required* if the source node is a remote multiuser HP 1000. *Default*: If this parameter is omitted and the target node is your local node, the account under which the program is running is used.
tnode	*Character array (FORTRAN); String (PASCAL).* The name of the target node. Must be in the form *node*[.*domain*[.*organization*]]. *Default*: You may omit the organization, organization and domain, or all parts of the node name. If the organization, or organization and domain, are omitted, the local organization and/or domain will be used. If the entire node name is omitted, it will default to the local node name.
options	*32-bit integer*. A two-word (32-bit) parameter which identifies specific options. An option is included if its corresponding bit is set. If no bits are set, no options are specified. The options and their corresponding bits are listed below (zero represents the least significant bit). These options are equivalent to those that can be used with DSCOPY interactively. For an explanation of the meaning of the

following options, refer to the "Copy Descriptor" discussion in this section.

`0`	Reserved for future use.
`1`	`ASCII`
`2`	`BINARY`
`3`	Reserved for future use.
`4`	`FIXED`
`5`	`INTERCHANGE`
`6`	`MOVE`
`7`	`OVERWRITE`
`8`	`QUIET`
`9`	`REPLACE`
`10`	`STRIP`
`11`	`VARIABLE`
`12`	`SILENT`
`13` through `31`	Reserved for future use.

`rsize` — *`32-bit integer`*. Appends the `RSIZE` option to the *`builtdescriptor`*. The value in *`rsize`* is in bytes. If fixed length records are being produced, *`rsize`* is the size of each record. If variable length records are being produced, *`rsize`* limits the size of the largest record and records may be padded or truncated. If *`rsize`* is zero, the `RSIZE` option is not appended to the *`builtdescriptor`* and the target file will have the same record size as the source file. You cannot copy files with records longer than 4400 bytes to or from an HP 1000.

`fsize` — *`32-bit integer`*. Appends the `FSIZE` option to the *`builtdescriptor`*. The value in *`fsize`* specifies how much space to allocate for the target file. If the target file has fixed length records, *`fsize`* is in records. If the target file has variable length records, *`fsize`* is the number of maximum size records. You can use this option instead of the HP 1000 file descriptor size parameter to specify the size of an HP 1000 target file. If *`fsize`* is zero, the `FSIZE` option is not appended to the *`builtdescriptor`* and the target file will be the same size as the source file.

If your program is written in Pascal/1000, Version 2, you must set the `FIXED_STRING` option before declaring `DscopyBuild`. In addition, the Pascal routines `SetStrLen` and `StrMax` must be used to initialize the *`builtdescriptor`* string prior to calling `DscopyBuild`. `FIXED_STRING`, `SetStrLen`, and `StrMax` are described in the *Pascal/1000 Reference Manual*. If your program is written in Pascal/1000, Version 1, you must use the routine `StrDsc` to convert the *`builtdescriptor`* string to a format that can be processed by both the calling program and DSCOPY. This routine is described in the *RTE-A Programmer's Reference Manual*.

Error Messages

The following error messages are returned to the current list device when DSCOPY is run interactively.

DSCOPY User Error Messages

Message	Meaning
APPEND overrides REPLACE or OVERWRITE (NFT/1000 WARN -1)	Only APPEND, OVERWRITE, or REPLACE can be in effect at one time.
option overrides previous data type setting (NFT/1000 WARN -2)	Only ASCII or BINARY may be in effect at one time. One of these options was given while the other was in effect.
OVERWRITE overrides APPEND or REPLACE (NFT/1000 WARN -4)	Only APPEND, OVERWRITE, or REPLACE may be in effect at one time. OVERWRITE was given while APPEND or REPLACE was in effect.
option overrides previous setting (NFT/1000 WARN -5)	An option (*option*) was given a value while a previous value was in effect.
option overrides previous record type setting (NFT/1000 WARN -6)	Only FIXED or VARIABLE may be in effect at one time. One of these options was given while the other was in effect.
REPLACE overrides APPEND or OVERWRITE (NFT/1000 WARN -7)	Only REPLACE, APPEND, or OVERWRITE may be in effect at one time. REPLACE was given while APPEND or OVERWRITE was in effect.
Source logon or node name overrides previous setting (NFT/1000 WARN -8)	The source logon or node name given in the copy descriptor will override the default setting for the current copy descriptor. Subsequent copy descriptors will not be affected.
Target logon or node name overrides previous setting (NFT/1000 WARN -9)	The target logon or node name given in the copy descriptor will override the default setting for the current copy descriptor. Subsequent copy descriptors will not be affected.
Input line too long (NFT/1000 ERR -10)	The input line was greater than 80 characters.
Cannot find closing quotation mark (NFT/1000 ERR -13)	An opening quotation mark was given in the command but the closing quotation mark could not be found.
No source file was given (NFT/1000 ERR -14)	No source file was specified in the copy descriptor.
Cannot find last square bracket for logon string (NFT/1000 ERR -15)	A logon string must be terminated by a square bracket (]).
Logon string is too long (NFT/1000 ERR -16)	A logon string must not be greater than 60 characters.
Node name is too long (NFT/1000 ERR -17)	A node name must not be greater than 50 characters.
File name is too long (NFT/1000 ERR -18)	A file name must not be greater than 256 characters.
Illegal command (NFT/1000 ERR -19)	The command issued is illegally formed.
Unknown command (NFT/1000 ERR -20)	DSCOPY does not recognize the command.
option is an unknown copy descriptor option (NFT/1000 ERR -21)	DSCOPY does not recognize the copy descriptor option.

Message	Meaning
`Error in option value for` *`option`* `option` `(NFT/1000 ERR -22)`	The value associated with *`option`* is invalid.
`File names cannot be defaulted` `(NFT/1000 ERR -23)`	An attempt was made to default a file name.
`Input command is too big` `(NFT/1000 ERR -24)`	The input command is greater than 256 characters.
`Cannot open file` *`filename`* `(NFT/1000 ERR -25)`	DSCOPY was unable to open the given file due to an unknown error.
`Unable to initialize DSCOPY` `(NFT/1000 ERR -26)`	This error returned because: (1) DSCOPY was unable to acquire sufficient resources; or (2) an error occurred in accessing DSAM or DSAM tables.
`Cannot create a sparse/variable length record file` `(NFT/1000 ERR -27)`	The target computer does not support sparse files with variable length records.
`Cannot schedule PRDC1 helper program` `(NFT/1000 ERR -28)`	PRDC1 could not be scheduled. Either it could not be found, or there are insufficient resources.
`Cannot schedule DSCOPY program` `(NFT/1000 ERR -29)`	The `DSCOPY` call could not schedule the DSCOPY program.
`DSCOPY aborted` `(NFT/1000 ERR -30)`	The `DSCOPY` call noticed that the DSCOPY program was terminated for some reason.
`Cannot schedule Producer program on source computer` `(NFT/1000 ERR -31)`	The Producer program could not be scheduled.
`Cannot schedule Consumer program on target computer` `(NFT/1000 ERR -32)`	The Consumer program could not be scheduled.
`Record size is too large` `(NFT/1000 ERR -33)`	The source computer or the target computer could not accept the interchange file copy request because of buffer space limitations.
`File is of inappropriate type` `(NFT/1000 ERR -34)`	Either the `TR` or `LL` command was given a file which is of an unacceptable type.
`Read from input file failed` `(NFT/1000 ERR -35)`	An unknown error occurred in reading from the input file. The file may be corrupt or the record read was too long.
`Transfer succeeded` `(NS/NFTERR 0)`	The file copy process was successful.
`Internal NFT error` `(NS/NFTERR 1)`	An internal NFT error has occurred.
`Unable to logon to source computer` `(NS/NFTERR 2)`	An error occurred in logging on to the source computer, or no logon string was given when one was required.
`Unable to logon to target computer` `(NS/NFTERR 3)`	An error occurred in logging on to the target computer, or no logon string was given when one was required.
`Unable to open or access source file or device` `(NS/NFTERR 4)`	A file system error occurred in opening or accessing the source file because of protection violation or unsupported device.
`Unable to connect to source computer` `(NS/NFTERR 5)`	A connection could not be established to the computer where the source file resides.
`Unable to connect to target computer` `(NS/NFTERR 6)`	A connection could not be established to the computer where the target file resides.
`Insufficient resources at source computer` `(NS/NFTERR 7)`	There are insufficient resources at the source computer to copy the file or files. Refer to the qualifying error string.

Message	Meaning
`Insufficient resources at target computer (NS/NFTERR 8)`	There are insufficient resources at the target computer to copy the file or files. Refer to the qualifying error string.
`Source file not found (NS/NFTERR 9)`	The source file does not exist or the name was misspelled.
`Target file not found (NS/NFTERR 10)`	The target file does not exist or the name was misspelled.
`Transfer terminated by user (NS/NFTERR 11)`	Acknowledges the user's abort or cancel request.
`Requested data type refused (NS/NFTERR 14)`	The user-defined data type (`ASCII` or `BINARY`) was refused.
`Requested record type refused (NS/NFTERR 15)`	The user-defined record type (`FIXED` or `VARIABLE`) was refused.
`Requested record size refused (NS/NFTERR 17)`	The user-defined record size value (`RSIZE`) was refused.
`Requested file size refused (NS/NFTERR 18)`	The user-defined file size value (`FSIZE`) was refused.
`Conflicting attributes or options (NS/NFTERR 19)`	An attempt was made to send a sparse file with variable length records to a target computer that does not support this file type, or that contains conflicting attributes.
`Target record size is invalid (NS/NFTERR 20)`	The user-defined record size (`RSIZE`) value is out of the acceptable range.
`Target file size is invalid (NS/NFTERR 21)`	The user-defined file size (`FSIZE`) value is out of the acceptable range.
`Target file already exists (NS/NFTERR 22)`	The target file exists and neither `APPEND`, `REPLACE`, or `OVERWRITE` was specified.
`Need password to access source file (NS/NFTERR 23)`	The source file could not be accessed without the proper password.
`Need password to access target file (NS/NFTERR 24)`	The target file could not be accessed without the proper password.
`Out of disk space (NS/NFTERR 25)`	The target computer is out of disk space.
`Connection to source computer went down (NS/NFTERR 26)`	An error was detected on the connection to the computer where the source file or files reside.
`Connection to target computer went down (NS/NFTERR 27)`	An error was detected on the connection to the computer where the target file or files are to be created.
`Unable to purge target file (NS/NFTERR 28)`	The existing target file could not be purged for some reason.
`Invalid target file name (NS/NFTERR 29)`	The target file name is not valid for the target computer.
`Unable to purge source file (NS/NFTWARN 30)`	The source file could not be purged for some reason after the file copy succeeded (the `MOVE` option was given).
`Read from source file failed (NS/NFTERR 31)`	An unexpected source file system error occurred when reading from the source file.
`Write to target file failed (NS/NFTERR 32)`	An unexpected file system error occurred when writing data to the target file.
`Unable to create or open target file (NS/NFTERR 33)`	An unexpected file system error occurred in creating or opening the target file.
`Invalid or unsupported source device (NS/NFTERR 34)`	A request was made to copy a file from a non-disk device. This is not supported.

Message	Meaning
`Invalid or unsupported target device (NS/NFTERR 35)`	A request was made to copy a file to a non-disk device. This is not supported.
`Unable to close target file (NS/NFTERR 36)`	An unexpected file system error occurred in closing the target file.
`Incorrect source file password (NS/NFTERR 38)`	The given source file password was incorrect.
`Incorrect target file password (NS/NFTERR 39)`	The given target file password was incorrect.
`Removed invalid characters in target file name (NS/NFTWARN 41)`	The target file name contained some characters which were invalid for the target computer, and they were removed from the target file name before it was created.
`Target file name was truncated (NS/NFTWARN 42)`	The target file name was too large for the target computer and was truncated.
`Source and target file attributes differ (NS/NFTWARN 43)`	The source file attributes had to be modified so that the file could be copied to the target computer.
`Records were truncated to fit in target file (NS/NFTWARN 44)`	The user-defined record size value (`RSIZE`) was smaller than the size of the largest record in the source file and one or more records in the source file were truncated.
`Not compressing for this transfer (NS/NFTWARN 45)`	Either the source and/or the target computer does not support data compression (the `COMPRESS` option) or the file is being copied locally.
`Unable to turn on tracing (NS/NFTWARN 46)`	Indicates that an internal error has occurred.
`Cannot strip padding from fixed length records (NS/NFTWARN 47)`	The source file has fixed length records but the `VARIABLE` option was not given, or the target computer requested that the target file should have fixed length records.
`Unable to access target file or device (NS/NFTERR 48)`	An unexpected file system error occurred in accessing the target file.
`Invalid source file name (NS/NFTERR 49)`	The source file name is invalid on the source computer.
`No files matched source specification (NS/NFTERR 50)`	The source file name contained one or more wildcard characters, or was a directory, but no files or directories were matched by it.
`APPEND option is not supported (NS/NFTERR 51)`	The `APPEND` option is not supported on the target computer.
`OVERWRITE option is not supported (NS/NFTERR 52)`	The `OVERWRITE` option is not supported on the target computer.
`Unable to create directory (NS/NFTERR 53)`	An unexpected file system error occurred in creating a directory on the target computer.
`Error creating or accessing scratch file on source node (NS/NFTERR 54)`	An unexpected file system error occurred in reference to a scratch file on the source computer.
`Unable to start NFT service on the source node (NS/NFTERR 55)`	An NFT server program could not be initialized on the source computer. Either the server program could not be scheduled or the server could not enter the proper session.
`Unable to start NFT service on the target node (NS/NFTERR 56)`	An NFT server program could not be initialized on the target computer. Either the server program could not be scheduled or the server could not enter the proper session.
`Incoming connection has gone down (NS/NFTERR 57)`	An incoming connection to an NFT server program has gone down for an unknown reason.
`MOVE option is not supported (NS/NFTWARN 58)`	The source computer does not support the `MOVE` option.

6

Network Interprocess Communication

Table of Contents

NetIPC Common Parameters

The `flags`, `opt`, `data`, `result`, `socketname`, and `nodename` parameters are common to many NetIPC calls. The `flags`, `opt`, and `result` parameters are also common to the Remote Process Management (RPM) calls. RPM calls are explained in the "Remote Process Management" section in the *NS-ARPA/1000 User/Programmer Reference Manual*. These calls can be used to schedule remote NetIPC programs.

The `opt` parameter provides functionality for NetIPC and RPM calls; `opt` usually has `data` associated with it. The `flags` parameter enables or disables certain functions for NetIPC calls. The `result` parameter returns error codes for NetIPC calls. The `socketname` and `nodename` parameters identify sockets and nodes, respectively.

Refer to the *NS-ARPA/1000 User/Programmer Reference Manual* for detailed information on NetIPC calls and their parameters.

Flags Parameter

The `flags` parameter is a bit map of 32 *special request bits*. By setting bits in the `flags` parameter, you can invoke various services in some NetIPC calls.

NetIPC and RPM calls assume that the bits in the `flags` parameter are numbered from left to right with the most significant bit being *one* and the least significant bit being bit *32*. In the NetIPC and RPM sections, whenever the `flags` parameter is discussed, bit *one* is the most significant bit:

```
MSB
1 2 3 4 5 6 ...           32          Pascal, NetIPC, and RPM

MSB
31 30 29 28 ...            0          FORTRAN
```

Opt Parameter

The `opt` parameter allows you to request optional services when invoking certain NetIPC and RPM calls. It enables calls that include the `opt` parameter to accept an arbitrary number of arguments that are either protocol or operating system specific. To help you distinguish between the `opt` parameter and a `flag` parameter, remember that the `opt` parameter is an array and usually has data associated with it.

Data Parameter

The data parameters reference data vectors or data buffers.

Result Parameter

Every NetIPC call has a `result` parameter. If an error occurs when a program uses a NetIPC call, an error code is returned to this parameter. The *NS-ARPA/1000 Error Message and Recovery Manual* lists and explains the NetIPC error codes.

Socketname Parameter

A *socket name* (the `socketname` parameter) may be a maximum of 16 characters long and may consist of any ASCII character. Upper and lower case characters are not considered distinct.

Nodename Parameter

A *node name* (the `nodename` parameter) refers to a remote node and has a hierarchical structure as follows:

```
   node[.domain[.organization]]
```

Each `node`, `domain`, and `organization` name may be a maximum of 16 characters long. The maximum total length of a fully-qualified node name is 50 characters. All alphanumeric characters are allowed, including the underscore (_) and dash (−) characters, but the first character of each parameter must be alphabetic.

Loading NetIPC Programs

HP 1000 NetIPC programs should be compiled and linked as CDS programs. Refer to the *RTE-A Programmer's Reference Manual* and *RTE-A Link Manual* for more information on CDS programs. After the program is linked, an RTE executable file (type 6) is ready to be scheduled.

Process Scheduling

There are at least six different ways (listed below) to schedule a remote HP 1000 NetIPC process from another HP 1000 node. A remote HP 1000 NetIPC process must be ready to execute by being an RTE type 6 file.

- *Remote Process Management (RPM)*. The `RPMCreate` call programmatically schedules a program.
- *Program-to-Program communication (PTOP)*. The `POPEN` call programmatically schedules a program.
- *Distributed EXEC (DEXEC)*. One of the DEXEC scheduling calls, such as DEXEC 9, 10, 12, 23, 24, programmatically schedules a program.
- *REMAT.* The REMAT `QU` (queue schedule a program without wait) command interactively schedules a program.
- *TELNET virtual terminal*. Logon remotely with TELNET and use the RTE `XQ` (schedule a program without wait) command to interactively schedule a program.
- *RTE WELCOME file*. The WELCOME file can have RTE run commands to schedule programs after system boot up.

You cannot use any of the above NS-ARPA and DS/1000-IV compatible services to schedule a remote HP 1000 process from a non-HP 1000 node. These services are not provided with cross-system support.

Remote HP 1000 processes that are to work with non-HP 1000 processes can be manually started or can be programs that are started at system start up.

- To manually start up a NetIPC program, simply logon to the HP 1000 system and run the NetIPC program with the RTE `XQ` (run program without wait) command.
- To have the NetIPC program execute at system start up, put the RTE `XQ` command in the WELCOME file.

The `XQ` command is explained in the *RTE-A User's Manual*.

Remote HP 9000 NetIPC processes can be manually started or can be scheduled by daemons that are started at system start up. To manually start up a NetIPC program, simply logon to the HP 9000 system and run the NetIPC program. HP recommends that you write a NetIPC daemon to schedule your NetIPC programs. You can start the daemon at system start up by placing it in your `/etc/netlinkrc` file. Refer to the HP 9000 LAN software installation documentation for more information about this file and system start up.

To manually startup an HP 3000 NetIPC program, log on to the HP 3000 and run the NetIPC program with the `RUN` command. You can schedule the program to start at a particular time by writing a job file to execute the program, and then including time and date parameters in the `:STREAM` command that executes the job file.

To manually start up a PC NetIPC program, enter the NetIPC name at the MS-DOS prompt.

To execute from within MS-Windows, copy the NetIPC program files to your Window directory and double click with the mouse on the executable file.

IPCCONNECT

Requests a connection to another process.

`IPCCONNECT(`*`calldesc`*`,`*`pathdesc`*`,`*`flags`*`,`*`opt`*`,`*`vcdesc`*`,`*`result`*`)`

calldesc	*32-bit integer, by value in Pascal, by reference in FORTRAN*. Call socket descriptor. Refers to a call socket owned by the calling process.
pathdesc	*32-bit integer, by value in Pascal, by reference in FORTRAN*. Path report descriptor. Refers to the path report which indicates the location of the destination call socket (this is the call socket to which the connection request will be sent). A path report descriptor can be obtained by calling `IPCLookUp` or `IPCGet`.
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. The following option is defined for this call: `flags [22]`—CHECKSUMMING (input). When set, this flag causes TCP to enable checksumming. However, *not* setting this bit does not ensure that checksumming will not occur. TCP checksum will always be performed if: the peer process calls `IPCRecvCn` with the checksumming bit set. TCP checksum is performed in addition to data link checksum. If TCP performs checksumming, increased overhead is required and real-time integrity cannot be guaranteed.
opt	*Byte array (Pascal); Integer array (FORTRAN), by reference*. An array of options and associated information. The following options are defined for this call: maximum send size (*optioncode* = `3`, *datalength* = `2`). A two-byte integer that specifies the maximum number of bytes you expect to send with a single `IPCSend` call on this connection. *Range*: 1 to 8,000 bytes. *Default*: 100 bytes. If this option is not specified, `IPCSend` will return errors if a call attempts to send greater than 100 bytes. maximum receive size (*optioncode* = `4`, *datalength* = `2`). A two-byte integer that specifies the maximum number of bytes you expect to receive with a single `IPCRecv` call on this connection. *Range*: 1 to 8,000 bytes. *Default*: 100 bytes. If this option is not specified, `IPCRecv` will return errors if a call attempts to receive greater than 100 bytes.
vcdesc	*32-bit integer, by reference*. VC socket descriptor. Refers to a VC socket that is the endpoint of the virtual circuit connection at this node. May be used in subsequent NetIPC calls to reference the connection.
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCCONTROL

Performs special operations on sockets.

IPCCONTROL

```
   IPCCONTROL(descriptor,requestfd,wrtdata,wlen,readdata,rlen,
            flags,result)
```

descriptor — *32-bit integer, by value in Pascal, by reference in FORTRAN.* The descriptor that refers to the socket to be manipulated. May be a call socket descriptor or VC socket descriptor depending on the request code specified in the request parameter.

requests — *32-bit integer, by value in Pascal, by reference in FORTRAN.* Request code. Defines which operation is to be performed. May be one of the following:

- 1 = Place the socket referenced in the *descriptor* parameter in asynchronous mode. For `IPCSend` and `IPCRecv` calls, this is the VC socket described by the VC socket descriptor in the *vcdesc* parameter. For `IPCRecvCn`, it is the call socket described by the call socket descriptor in the *calldesc* parameter.
- 2 = Place the socket referenced in the *descriptor* parameter in synchronous mode. For `IPCSend` and `IPCRecv` calls this is the VC socket described by the VC socket descriptor in the *vcdesc* parameter. For `IPCRecvCn`, it is the call socket described by the call socket descriptor in the *calldesc* parameter.
- 3 = Change the referenced socket's synchronous timeout. The default timeout value is 60 seconds. For `IPCSend` and `IPCRecv` calls, this is the VC socket described by the VC socket descriptor in the *vcdesc* parameter. For `IPCRecvCn`, it is the call socket described by the call socket descriptor in the *calldesc* parameter. The timeout value is given in tenths of seconds. (For example, a value of 1200 would indicate 120 seconds.) The new timeout value must be placed in the *wrtdata* parameter. The timeout value must be in the range of zero to 32767. Negative values have no meaning and will result in error. A value of zero sets the timeout to infinity. The timeout will not be reset if the referenced socket is switched to asynchronous mode and then back to synchronous mode.
- 1000 = Change the read threshold of the VC socket referenced in *descriptor* parameter. (Read thresholds are one byte by default.) The *descriptor* parameter must reference a VC socket descriptor. The new read threshold value must be placed in the *wrtdata* parameter.
- 1001 = Change the write threshold of the VC socket referenced by the *descriptor* parameter. (Write thresholds are one byte by default.) The *descriptor* parameter must reference a VC socket descriptor. The new write threshold value must be placed in the *wrtdata* parameter.

wrtdata	*16-bit integer, by reference*. A data buffer or data vector used to pass timeout and threshold information. If a *request* of 3, 1000, or 1001 is specified, the *wrtdata* and *wlen* parameters are required.
wlen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. Length in bytes of the *wrtdata* parameter. Must be set to 2 bytes.
readdata	*Array, by reference*. This parameter is reserved for future use.
rlen (input/output)	*32-bit integer, by reference*. This parameter is reserved for future use.
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero).
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCCREATE

Creates a call socket.

```
    IPCCREATE(socketkind,protocol,flags,opt,calldesc,result)
```

socketkind	*32-bit integer, by value in Pascal, by reference in FORTRAN*. Indicates the type of socket to be created. Must be 3 to indicate a call socket. (Other values are reserved for future use.) *Default*: If zero is specified, a call socket will be created.
protocol	*32-bit integer, by value in Pascal, by reference in FORTRAN*. Indicates the protocol module that the calling process wishes to access. Must be 4 to indicate Transmission Control Protocol (TCP). (Other values are reserved for future use.) *Default*: If zero is specified, TCP will always be chosen for call sockets.
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero).
opt	*Byte array (Pascal); Integer array (FORTRAN), by reference*. An array of options and associated information. The following options are defined for this call: • maximum connection requests backlog (*optioncode* = 6, *datalength* = 2). A two-byte integer that specifies the maximum number of unreceived connection requests that may be queued to a call socket. The value can be from 0 to 10. *Default*: Three requests. (NOTE: A queue limit of three may be too few if many processes attempt to initiate connections to the call socket simultaneously. If this occurs, some connection requests may be ignored.) • protocol address (*optioncode* = 128, *datalength* = 2). A two-byte integer that specifies a TCP protocol address to be used by the newly created call socket. The valid range for IPC address is 1 to 32767. If this option is not specified, NetIPC will dynamically allocate an address. *Recommended Range*: The recommended range of TCP addresses for user applications is from 30767 to 32767 decimal.
calldesc	*32-bit integer, by reference*. Call socket descriptor. Refers to the newly created call socket.
result	*32-bit integer, by reference*. The returned error code; zero if no error.

IPCDEST

Creates a path report descriptor.

```
   IPCDEST(socketkind,nodename,nodelen,protocol,protoaddr,protolen,
           flags,opt,pathdesc,result)
```

socketkind	*32-bit integer, by value in Pascal, by reference in FORTRAN.* Defines the type of socket. Must be 3 to specify a call socket. Other values are reserved for future use.
nodename	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference.* A variable length array of ASCII characters identifying the node on which the path report descriptor is to be created. The syntax of the node name is *node*[.*domain*[.*organization*]], which is further described in the *NS-ARPA/000 User/Programmer Reference Manual*.
	Default: You may omit the organization, organization and domain, or all parts of the node name. When organization or organization and domain are omitted, they will default to the local organization and/or domain. If the *nodelen* parameter is set to zero, *nodename* is ignored and the node name defaults to the local node.
nodelen	*32-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of the *nodename* parameter. If this parameter is set to zero, the *nodename* parameter is ignored and the node name defaults to the local node. A fully-qualified node name length may be 50 bytes long.
protocol	*32-bit integer, by value in Pascal, by reference in FORTRAN.* Defines the Transport Layer protocol to be used. Must be 4 to indicate the Transmission Control Protocol (TCP). Other values are reserved for future use.
protoaddr	*integer array, by reference.* A data buffer that contains a TCP protocol address. *Recommended Range*: The recommended range of TCP addresses for user applications is from 30767 to 32767 decimal.
protolen	*32-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of the protocol address. TCP protocol addresses are two bytes long.
flags	*32-bit integer, by reference.* A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero).
opt	No options are defined for this call.
pathdesc	*32-bit integer, by reference.* Path report descriptor. Describes the location of named call socket. May be used in a subsequent `IPCConnect` call to establish a connection to another process.
result	*32-bit integer, by reference.* The error code returned; zero if no error.

IPCGET

Receives a descriptor that has been given away via `IPCGive`.

`IPCGET(`*`givenname`*`,`*`nlen`*`,`*`flags`*`,`<u>*`descriptor`*</u>`,`<u>*`result`*</u>`)`

givenname	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. An array containing the ASCII-coded socket name that was assigned to the descriptor when it was given away via a call to `IPCGive`. Upper and lower case characters are not considered distinct.
nlen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The length in characters of *givenname*. Maximum length is 16 bytes.
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero). If `IPCGet` is called repeatedly, this field must be cleared before each successive call.
descriptor	*32-bit integer, by reference*. The descriptor that was given away via a call to `IPCGive`. May be a call socket descriptor, path report descriptor, or VC socket descriptor.
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCGIVE

Gives up a descriptor so that another process may obtain it.

```
   IPCGIVE(descriptor,givenname,nlen,flags,result)
```

descriptor	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The descriptor to be given up. May be a call socket descriptor, VC socket descriptor or path report descriptor.
givenname (input/output)	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. An array containing the ASCII-coded socket name to be temporarily assigned to the specified descriptor. Upper and lower case characters are not considered distinct. NetIPC can also return a randomly generated, eight-character name to this parameter (see *nlen*).
nlen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The length in characters of *givenname*. Maximum length is 16 bytes.
	Default: If zero is specified, NetIPC will generate a random eight byte name and return it in the *givenname* parameter. (The length eight is not returned through *nlen*.)
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero).
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCLOOKUP

Obtains a path report descriptor.

```
    IPCLOOKUP(socketname,nlen,nodename,nodelen,flags,pathdesc,
              protocol,socketkind,result)
```

socketname	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. An array containing the ASCII-coded name of the call socket to be "looked up." Upper and lower case characters are not considered distinct.
nlen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The length of the socket name in characters. Maximum length is 16 characters.
nodename	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. A variable length array of ASCII characters identifying the node where the socket specified in the *socketname* parameter resides. The syntax of the node name is *node*[.*domain*[.*organization*]], which is further described in the *NS-ARPA/1000 User/Programmer Reference Manual*.
	Default: You may omit the organization, organization and domain, or all parts of the node name. When organization or organization and domain are omitted, they will default to the local organization and/or domain. If the *nodelen* parameter is set to zero, *nodename* is ignored and the node name defaults to the local node.
nodelen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of the *nodename* parameter. If this parameter is zero (0), the *nodename* parameter is ignored and the node name defaults to the local node. A fully-qualified node name length may be 50 bytes long.
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero).
pathdesc	*32-bit integer, by reference*. Path report descriptor. Refers to the path report descriptor which indicates the location of the named call socket. May be used in subsequent NetIPC calls (`IPCConnect`, `IPCName`, `IPCGive`, etc.).
protocol	*32-bit integer, by reference*. Identifies the protocol module with which the "looked up" socket is associated. May be used in an `IPCCreate` call to create a call socket with the appropriate protocol binding.
socketkind	*32-bit integer, by reference*. Identifies the socket's type.
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCNAME

Associates a name with a call socket descriptor.

```
   IPCNAME(descriptor,socketname,nlen,result)
```

descriptor	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The call socket descriptor to be named.
socketname (input/output)	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. An array containing the ASCII-coded socket name to be associated with the descriptor. Upper and lower case characters are considered equivalent. NetIPC can also return a randomly-generated name in this parameter (see *nlen*).
nlen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The length in characters of *socketname*. Maximum length is 16 characters. *Default*: If zero is specified, NetIPC will return a random, eight-byte name in the *socketname* parameter. The eight-byte length is not returned in the *nlen* parameter.
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCNAMERASE

Deletes a name associated with a call socket descriptor.

```
   IPCNAMERASE(socketname,nlen,result)
```

socketname	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. An array containing an ASCII-coded name that was previously associated with a call socket descriptor via `IPCName`. Upper and lower case characters are considered equivalent.
nlen	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of the specified name. Maximum length is 16 bytes.
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCRECV

Establishes a virtual circuit connection by receiving a response to a connection request, or receives data on a previously established connection.

```
   IPCRECV(vcdesc,data,dlen,flags,opt,result)
```

vcdesc	*32-bit integer, by value in Pascal, by reference in FORTRAN*. VC socket descriptor. Refers to a VC socket that: (1) is the endpoint of a virtual circuit connection that has not yet been established, or (2) is the endpoint of a previously established virtual circuit on which data will be received.
data	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference*. A data buffer that will hold the received data, or a data vector describing the location where the data is to be placed.
dlen (input/output)	*32-bit integer, by reference*. When the *data* parameter is a data buffer, *dlen* is the maximum number of bytes you are willing to receive. When the *data* parameter is a data vector, *dlen* refers to the length of the data vector in bytes. As a return parameter (output), *dlen* indicates how many bytes were actually received. If `IPCRecv` is used to establish a connection (not to receive data), the *dlen* parameter is meaningless on input and a value of 0 is returned on output. If the DATA_WAIT flag (see `flags [21]` below) is zero, then *dlen* returns with the length of whatever data there is. If the DATA_WAIT flag is set to one, then either *dlen* returns with the same amount requested or a "WOULD BLOCK" error occurs.
flags (input/output)	*32-bit integer, by reference*. A 32-bit map of special request bits. The first `IPCRecv` call establishes a virtual connection and *flags* has no meaning. For subsequent `IPCRecv` calls, *flags* will then be invoked for the established connection. *Flags* must be initialized each time it is used by *any* NetIPC call. The following flags are defined for this call:

- `flags [21]`—DATA_WAIT (input). When this flag is set, `IPCRecv` waits until all the data that it requested in the *dlen* parameter has been received. If this bit is set to zero, `IPCRecv` may complete receiving less data than it requested in *dlen*.

Note User programs written prior to software Revision 5.0 that wait on the `IPCRecv` call until *dlen* amount of data has been received *must* change to set the DATA_WAIT flag to continue operating as they did before.

- `flags [31]`—PREVIEW (input). When set, this flag allows you to preview the data queued on the connection. Data is placed in the *data* parameter but not dequeued from the connection. Because the data is not dequeued, another `IPCRecv` call is needed to delete the same data.

- `flags [32]`—VECTORED (input). When set, this flag indicates that the *`data`* parameter is a data vector and not a data buffer.

`opt`	*`Byte array (Pascal); Integer array (FORTRAN), by reference`*. An array of options and associated information. The following option is defined for this call: • data offset (*`optioncode`* = `8`, *`datalength`* = `2`). A two-byte integer that defines a byte offset from the beginning of a data buffer where NetIPC is to begin placing the data. This option is valid only if the data parameter is a *data buffer* and not data vector.
`result`	*`32-bit integer, by reference`*. The error code returned; zero if no error.

IPCRECVCN

Receives a connection request on a call socket.

```
   IPCRECVCN(calldesc,vcdesc,flags,opt,result)
```

calldesc	*32-bit integer, by value in Pascal, by reference in FORTRAN*. Socket descriptor. Refers to a call socket owned by the calling process.
vcdesc	*32-bit integer, by reference*. VC socket descriptor. Refers to a VC socket that is the endpoint of the newly-established virtual circuit connection.
flags	*32-bit integer, by reference*. A 32-bit map of special request bits. The following flags are defined for this call: • `flags [22]`—CHECKSUMMING (input). When set, this flag causes TCP to enable checksumming. However, *not* setting this bit does not ensure that checksumming will not occur. TCP checksum will always be performed if: the peer process calls `IPCConnect` with the checksumming bit set. TCP checksum is performed in addition to data link checksum. If TCP performs checksumming, increased overhead is required and real-time integrity cannot be guaranteed.
opt	*Byte array (Pascal); Integer array (FORTRAN), by reference*. An array of options and associated information. The options are: • maximum send size (*optioncode* = `3`, *datalength* = `2`). A two-byte integer that specifies the maximum number of bytes you expect to send with a single call to `IPCSend` on this connection. *Range*: 1 to 8,000 bytes. *Default*: 100 bytes. If this option is not specified, `IPCSend` will return an error if a call attempts to send greater than 100 bytes. • maximum receive size (*optioncode* = `4`, *datalength* = `2`). A two-byte integer that specifies the maximum number of bytes you expect to receive with a single call to `IPCRecv` on this connection. *Range*: 1 to 8,000 bytes. *Default*: 100 bytes. If this option is not specified, `IPCRecv` will return errors if a call attempts to receive greater than 100 bytes.
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCSELECT

Determines the status of a call socket or VC socket.

    IPCSELECT(sdbound,readmap,writemap,exceptionmap,timeout,result)

sdbound (input/output)	*32-bit integer, by reference*. Specifies the upper ordinal bound on the range of descriptors specified in the *readmap*, *writemap*, and *exceptionmap* parameters. An `IPCSelect` call will be most efficient if this parameter is set to the maximum ordinal value of the sockets specified in these parameters. Because a NetIPC process may have concurrent access to a maximum of 32 descriptors, *sdbound* may be given a maximum value of 32. As an output parameter, *sdbound* contains the upper ordinal boundary of all of the descriptors that met the select criteria. If none of the criteria were met, *sdbound* will be set to zero.
readmap (input/output)	*32-bit integer, by reference*. A bit map indexed by VC socket descriptors. When *readmap* is an input parameter, this map should have bits set for all of the VC sockets from which you would like to receive data. As an output parameter, *readmap* is a bit map describing all of the read-selected VC sockets that are readable.
writemap (input/output)	*32-bit integer, by reference*. A bit map indexed by either call socket descriptors or VC socket descriptors. When *writemap* is an input parameter, this map should have bits set for all of the call sockets on which you would like to initiate connections, or all of the VC sockets to which you would like to send data. As an output parameter, *writemap* is a bit map describing all of the write-selected sockets that are writeable.
exceptionmap (input/output)	*32-bit integer, by reference*. A bit map indexed by either call socket descriptors or VC socket descriptors. When *exceptionmap* is an input parameter, this map should have bits for all of the sockets for which notification of exceptional conditions is desired. As an output parameter, *exceptionmap* is a bit map describing all of the exception-selected sockets that are exceptional. For call sockets, an exceptional condition is present if a connection request is queued to the socket; for VC sockets, an exceptional condition is present if the connection referenced by the socket has been aborted.
timeout	*32-bit integer, by value in Pascal, by reference in FORTRAN*. The number of tenths of seconds the calling process is willing to wait for some event to occur which would cause `IPCSelect`'s report to change. This timeout is put into effect only when none of the sockets referenced can immediately satisfy the select criteria (i.e., none are readable, writeable or exceptional). If this value is set to zero, the call will not block. If it is set to -1, the timeout will be set to infinity (i.e., the call will block).
result	*32-bit integer, by reference*. The error code returned; zero if no error.

IPCSEND

Sends data on a virtual circuit connection.

IPCSEND

```
IPCSEND(vcdesc,data,dlen,flags,opt,result)
```

vcdesc	*32-bit integer, by value in Pascal, by reference in FORTRAN.* VC socket descriptor. Refers to the VC socket endpoint of the connection through which the data will be sent. A VC socket descriptor can be obtained by calling `IPCConnect`, `IPCRecvCn`, and `IPCGet`.
data	*Packed array of characters (Pascal); Integer array (FORTRAN), by reference.* A buffer that will hold the data to be sent, or a data vector describing where the data to be sent is located.
dlen	*32-bit integer, by value in Pascal, by reference in FORTRAN.* If *data* is a data buffer, *dlen* is the length of the data in the buffer. If *data* is a data vector, *dlen* is the length of the data vector.
flags	*32-bit integer, by reference.* A 32-bit map of special request bits. The following flags are defined for this call: • `flags [27]`—HIGH THROUGHPUT (input). Indicates that you would prefer high throughput to low delay. • `flags [32]`—VECTORED (input). Indicates that the data parameter refers to a data vector and not to a data buffer.
opt	*Byte array (Pascal); Integer array (FORTRAN), by reference.* An array of options and associated information. The following option is defined for this call: • data offset (*optioncode* = `8`, *datalength* = `2`). A two-byte integer that indicates a byte offset from the beginning of the data buffer where the data to be sent actually begins. Only valid if the data parameter is a data buffer.
result	*32-bit integer, by reference.* The error code returned; zero if no error.

IPCSHUTDOWN

Releases a descriptor and any resources associated with it.

```
   IPCSHUTDOWN(descriptor,flags,opt,result)
```

descriptor	*32-bit integer, by value in Pascal, by reference in FORTRAN.* The descriptor to be released. May be a call socket descriptor, VC socket descriptor, or path report descriptor.
flags	*32-bit integer, by reference.* A 32-bit map of special request bits. This parameter is reserved for future use. All bits must be clear (set to zero).
opt	*Byte array (Pascal); Integer array (FORTRAN), by reference.* An array of options and associated information. This parameter is reserved for future use. You must initialize the *opt* parameter to contain zero arguments.
result	*32-bit integer, by reference.* The error code returned; zero if no error.

ADDOPT

Adds an argument and its associated data to the *opt* parameter.

```
   ADDOPT(opt,argnum,optioncode,datalength,data,error)
```

opt	*Byte array (Pascal); Integer array (FORTRAN), by reference.* The *opt* parameter to which you want to add an argument.
argnum	*16-bit integer, by value in Pascal, by reference in FORTRAN.* The number of the argument to be added. The first argument is number zero.
optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN.* The option code of the argument to be added. These codes are described in each NetIPC call *opt* parameter description.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of the data to be included. This information is provided in each NetIPC call *opt* parameter description.
data	Packed array of characters (Pascal); Integer array (FORTRAN), by reference. An array containing the data associated with the argument.
error	*16-bit integer, by reference.* The error code returned; zero if no error.

ADROF

Obtains the byte address of any byte within a data object.

```
   ADROF(firstobjword,offset,byteaddress)
```

firstobjword	*16-bit integer, by reference*. The name of the first (16-bit) word of the data object.
offset	*16-bit integer, by value in Pascal, by reference in FORTRAN*. An offset from the beginning of the data object. May be positive or negative. (The first byte of a data object resides at offset zero.)
byteaddress	*16-bit integer, by reference*. The byte address of the byte that is *offset* bytes away from the first object word.

INITOPT

Initializes the *opt* parameter so that arguments can be added.

```
   INITOPT(opt,optnumarguments,error)
```

opt	*Byte array (Pascal); Integer array (FORTRAN), by reference*. The *opt* parameter to be initialized.
optnumarguments	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The number of arguments that will be placed in the *opt* parameter. If this parameter is zero, the *opt* parameter will be initialized to contain zero arguments.
error	*16-bit integer, by reference*. The error code returned; zero if no error.

READOPT

Obtains the option code and argument data associated with an `opt` parameter argument.

```
   READOPT(opt,argnum,optioncode,datalength,data,error)
```

opt	*Byte array (Pascal); Integer array (FORTRAN), by reference*. The *opt* parameter to be read.
argnum	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The number of the argument to be obtained. The first argument is number zero.
optioncode	*16-bit integer, by reference*. The option code associated with the argument. These codes are described in each NetIPC call *opt* parameter description.
datalength (input/output)	*16-bit integer, by reference*. The length of the data buffer into which the argument should be read. If the data buffer is not large enough to accommodate the argument data, an error will be returned. On output, this parameter contains the length of the data actually read. (The length of the data associated with a particular option code is provided in each NetIPC call *opt* parameter description.)
data	*Array, by referenc*e. An array which will contain the data read from the argument.
error	*16-bit integer, by reference*. The error code returned; zero if no error.

NetIPC Error Codes

These error codes are returned to the `result` parameter of Network Interprocess Communication (NetIPC) calls. Refer to the *NS-ARPA/1000 Error Message and Recovery Manual* for more complete explanations of these error codes.

NetIPC Error Codes

Code	Meaning
0	The call was successful.
4	The network is down.
5	Illegal socket type.
6	Illegal protocol.
7	Illegal flags.
8	Illegal option.
10	Protocol type mismatch.
11	No memory.
12	Messages queued option error.
14	Illegal TCP address.
15	Socket limit exceeded.
16	No path records.
19	Message size option error.
20	Data offset error.
21	Duplicate option.
24	Connection queued option error.
28	Illegal name length.
29	Illegal descriptor.
30	Cannot name VC socket.
31	Duplicate name.
34	Aborted locally.
35	Name limit exceeded.
36	Name table full.
37	Name not found.
38	No ownership.
39	Illegal registry name.
40	Unknown registry.
44	No registry response.
46	Could not interpret path.
50	Bad length.
51	Not a path report descriptor.
52	Protocol mismatch.
53	Socket type mismatch.

Code	Meaning
54	Not a call socket.
55	No sockets available.
56	Would block error.
59	Timed out.
62	`IPCRecv` expected.
64	Aborted by peer.
65	Connection aborted.
66	Not a VC socket.
68	Remote has gracefully released this socket.
70	Cannot give.
74	Illegal request.
76	Illegal timeout.
98	Bad vector address.
99	Bad vector data length.
106	Address in use.
107	NS-ARPA is not initialized; NS-ARPA is going up or down; or the NS-ARPA memory area is corrupt.
109	Remote has gracefully released this socket.
111	An NS-ARPA internal software error has been encountered.
116	No useable paths.
122	Too many users.
123	No resource numbers.
124	Bad entry number in option parameter.
125	Bad option data length.
126	Bad option total.
127	Cannot read option.
128	Illegal read threshold.
129	Illegal write threshold.
130	Write threshold too big.
131	Resource error.
132	No PXP path records.
133	No IP path records.
134	No 802 path records.
135	No TCP path records.
136	Bad upper bound.
1001	Cannot read select on the socket now.

7

Remote Process Management

Table of Contents

RPM Common Parameters

The *flags*, *opt*, *result*, and *nodename* parameters are common parameters used in the RPM calls. They follow the same conventions as the NetIPC parameters. For further information on these parameters, refer to "NetIPC Common Parameters" in the "Network Interprocess Communication" section and the "Remote Process Management" section in the *NS-ARPA/1000 User/Programmer Reference Manual*.

Use the `InitOpt`, `AddOpt`, and `ReadOpt` NetIPC calls to facilitate your use of the *opt* parameter. These NetIPC calls are explained in "Special NetIPC Calls" also in the "Network Interprocess Communication" section of the *NS-ARPA/1000 User/Programmer Reference Manual*.

Flags Parameter

The *flags* parameter is a bit map of 32 *special request bits*. By setting bits in the *flags* parameter, you can invoke various services in `RPMControl` and `RPMCreate`. The *flags* parameter must be initialized to set the desired bits before it is used in these RPM calls. Make sure you also clear the unused bits.

Note

NetIPC and RPM calls assume that the bits in the *flags* parameter are numbered from left to right with the most significant bit being bit *one* and the least significant bit being bit 32. In NetIPC and RPM, whenever the *flags* parameter is discussed, bit *one* is the most significant bit:

```
MSB
1 2 3 4 5 6 ...          32
```
Pascal, NetIPC, and RPM

```
MSB
31 30 29 28 ...           0
```
FORTRAN

Opt Parameter

The *opt* parameter allows you to request optional services when invoking the `RPMCreate` call. The *opt* parameter is an array which enables a varying number of arguments to be specified.

Use the `InitOpt`, `AddOpt`, and `ReadOpt` NetIPC calls to facilitate your use of the *opt* parameter.

Result Parameter

Every RPM call has a *result* parameter. If an error occurs when a program makes an RPM call, an error code is returned in this parameter. The *NS-ARPA/1000 Error Message and Recovery Manual* lists and explains the RPM error codes.

Nodename Parameter

A *node name* (the `nodename` parameter) refers to a node and has a hierarchical structure as follows:

```
   node[.domain[.organization]]
```

The NS-ARPA node name syntax is described in “Node Names” in the “Introduction” section of the *NS-ARPA/1000 User/Programmer Reference Manual*.

Loading RPM Programs

RPM parent programs must be compiled and linked as CDS programs. RPM child programs can be either CDS or non-CDS programs. If an RPM child program makes an RPM call, then it must be a CDS program. Refer to the *RTE-A Programmer’s Reference Manual* and *RTE-A Link Manual* for more information on CDS programs.

RPMCONTROL

Controls the execution of a child program.

```
   RPMCONTROL (pd,nodename,nodelen,reqcode,wrtdata,wrtlen,
               readdata,readlen,flags,result)
```

pd — *Byte array (Pascal); Word array (FORTRAN), by reference*. An array of 16 bytes containing the program descriptor of the child program to which control requests are sent. The program descriptor is a unique value returned from the RPMCreate call. Refer to the RPMCreate description in this section for more information about *pd*.

nodename — *Packed array of characters (Pascal); word array (FORTRAN), by reference*. A variable length array identifying the node on which the child program resides. The syntax of the node name is *node*[.*domain*[.*organization*]], which is further described in the *NS-ARPA/1000 User/Programmer Reference Manual*.

Default: You may omit the organization, organization and domain, or all parts of the node name. When organization or organization and domain are omitted, they will default to the local organization and/or domain. If the *nodelen* parameter is set to zero, *nodename* is ignored and the node name defaults to the local node.

nodelen — *32-bit non-negative integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of the *nodename* parameter. If *nodelen* is zero (0), the *nodename* parameter is ignored and the node name defaults to the local node. The maximum length of a fully-qualified node name length is 50 bytes.

If *nodelen* is zero, it is assumed that the parent is either sending the RPMControl request to a dependent child program that it previously scheduled or to a child program on the parent's node (which is the local node).

reqcode — *32-bit non-negative integer, by value in Pascal, by reference in FORTRAN*. The request code for the control operation to be performed on the child program. The request codes are RTE-A specific, and you should refer to the *RTE-A User's Manual* and *RTE-A Programmer's Reference Manual* for detailed explanations of these RTE-A commands and calls. The request codes for RPMControl are as follows:

- 20001—Suspend execution of the child program. No data is required and none is returned. Therefore, *wrtlen* and *readlen* must be zero.

  This request is equivalent to the RTE-A SS (suspend program) command. As in RTE-A, if a child program is in a state that prevents it from being suspended, the program is not suspended until it is in the right state. No error is returned in *result* in this case (similarly as in RTE-A).

- 20002—Resume execution of the child program at the point it was suspended. No data is required and none is returned. Therefore, *wrtlen* and *readlen* must be zero.

  This request is equivalent to the RTE-A GO (resume program) command.

RPMCONTROL

- 23120—Set the `IFBRK` break flag in the child program's ID segment. The child program must check this flag with the RTE-A `IFBRK` system call to respond to it. No data is required and none is returned. Therefore, *wrtlen* and *readlen* must be zero.
- 23030—Change the child program's priority. The priority number is a 16-bit integer from 1 to 32767 with the smaller number representing the higher priority. The priority number is placed in the *wrtdata* parameter. This request is equivalent to the RTE-A `PR` (change program priority) command, except that you cannot request the program priority.
- 23130—Get the child program's status. RPM invokes the RTE-A `IDINFO` call to obtain the status. Refer to the *RTE-A Programmer's Reference Manual* for more information and for a list of possible status values. The status is a 16-bit integer returned in the *readdata* parameter. The *readlen* parameter must be set to at least two bytes.

Due to a network time delay, the actual execution of any of the above requests may be delayed.

wrtdata	*Byte array (Pascal); Word array (FORTRAN), by reference.* A variable length array with data to be sent to the child program for the request. When a *reqcode* of 23030 is specified, the program priority is placed in *wrtdata*. The program priority is declared as a 16-bit integer, and the *wrtlen* parameter is two bytes.
wrtlen	*32-bit non-negative integer, by value in Pascal, by reference in FORTRAN.* Length in bytes of *wrtdata*. Only *reqcode* 23030 (`PR` command) sends information from the calling parent program in *wrtdata*. The parameter *wrtlen* must be two. All other request codes must specify a zero for *wrtlen*.
readdata	*Byte array (Pascal); Word array (FORTRAN), by reference.* A variable length array with the data returned to the calling parent program. If *reqcode* of 23130 is used, then the program status is returned in *readdata*.
readlen (input/output)	*32-bit non-negative integer, by reference.* On input, *readlen* is the maximum number of bytes expected in the *readdata* parameter. On output, *readlen* is the actual number of bytes received in the *readdata* parameter. If *result* is non-zero (an error has occurred), *readlen* is set to zero, and no data is in *readdata*. Only *reqcode* 23130 (`IDINFO` call) receives information from a child program (program status) in *readdata*. The parameter *readlen* must be two. All other request codes must specify a zero in *readlen*.
flags	*32 bits, by reference.* A 32-bit map of special request bits. This parameter is reserved for future use. This parameter must contain all zeroes (cleared).
result	*32-bit non-negative integer, by reference.* The result of the `RPMControl` request; zero if no error. If *result* is not zero, an error has occurred. Errors are defined in the *NS-ARPA/1000 Error Message and Recover Manual*.

RPMCREATE

Schedules a program and, if necessary, creates a session for that program to run in.

```
    RPMCREATE (progname,namelen,nodename,nodelen,login,loginlen,
              password,passwdlen,flags,opt,pd,result)
```

progname	*`Packed array of characters (Pascal); word array (FORTRAN), by reference`*. A variable length array of ASCII characters containing the name of the child program to be scheduled. If the child program does not reside in the working directory, the full path name of the child program must be specified. The child program must be an executable file. Although `RPMCreate` may accept program names up to 256 characters, the child program name on an HP 1000 RTE-A system may not exceed 64 characters. The *`progname`* parameter is not case sensitive.
namelen	*`32-bit positive integer, by value in Pascal, by reference in FORTRAN`*. The length in bytes of the program name. This must always be a positive integer.
nodename	*`Packed array of characters (Pascal); word array (FORTRAN), by reference`*. A variable length array of ASCII characters identifying the node on which the child program resides. The syntax of the node name is *`node`*`[.`*`domain`*`[.`*`organization`*`]]`, which is described in the *NS-ARPA/1000 User/Programmer Reference Manual*. *Default*: You may omit the organization, organization and domain, or all parts of the node name. When organization or organization and domain are omitted, they will default to the local organization and/or domain. If the *`nodelen`* parameter is set to zero, *`nodename`* is ignored and the node name defaults to the local node.
nodelen	*`32-bit non-negative integer, by value in Pascal, by reference in FORTRAN`*. Length in bytes of the *`nodename`* parameter. If *`nodelen`* is zero (0), the *`nodename`* parameter is ignored, and the child program is scheduled on the same node as the parent. A fully-qualified node name length may be 50 bytes long.
login	*`Packed array of characters (Pascal); word array (FORTRAN), by reference`*. A logon sequence for the (local or remote) node on which the child program is to be scheduled. *`login`* is an RTE-A logon without the password (defined in the *`password`* parameter described below). RPM needs the logon name to logon to the local or remote node.
loginlen	*`32-bit non-negative integer, by value in Pascal, by reference in FORTRAN`*. The length in bytes of the logon sequence. The maximum length for a logon on RTE-A is 16 bytes. If *`loginlen`* is zero (0), *`passwdlen`* must be zero. When the *`loginlen`* and *`passwdlen`* are both zero and *`nodename`* is the local node, the child program is scheduled and attached to the *parent program's session*. Even if the session-sharing flag (`flags[31]`) is not set to disable session-sharing, the child program will session-share with the parent program in this case. If *`nodename`* is NOT the local node and *`loginlen`* is zero, `RPMCreate` will return an error in *`result`*.

RPMCREATE

password	*Packed array of characters (Pascal); word array (FORTRAN), by reference.* A variable length array with the password for the RTE-A logon specified in *login*. If no password is required, the *passwdlen* parameter must be zero (0).
passwdlen	*32-bit non-negative integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of the *password* parameter. If *passwdlen* is zero (0), *password* is ignored. The maximum password length in RTE-A is 14 bytes.

flags

32 bits, by reference. A 32-bit map of special request bits representing various functions. Refer to "Flags Parameter" in the "Network Interprocess Communication" section of the *NS-ARPA/1000 User/Programmer Reference Manual* for explanations of the 32 special request bits and how to use them in Pascal/1000 and FORTRAN 77. The following flags are defined on input (bit 1 is the most significant bit); all other flags must be set to zero:

- `flags[2]`—wait for child (input). When set, this flag causes the calling parent program to wait until the child program terminates.

  The default is zero (0) for no waiting. The parent program resumes execution immediately after it is notified that the child program is successfully scheduled or an error occurs. Check the *result* parameter for an error.

- `flags[31]`—session-sharing (input). When set, this flag causes the child program to share a session with other child programs. The parent must set this bit for each child that is to share the same session.

  The default is zero (0) for no session-sharing—the child program is scheduled in a new session.

  Regardless of how `flags[31]` is set, session-sharing will occur on the local node in the parent program's session if *nodename* and *loginlen* are specified as follows:
  - *nodename* specifies the local node or *nodelen* is zero.
  - *loginlen* is zero.

- `flags[32]`—dependent (input). When set, this flag causes the scheduled child program to be dependent on the parent program. When the parent program terminates, the child program terminates automatically.

  The default is zero (0) making the child program independent. The scheduled child program continues executing on its own even after the parent program terminates.

opt

Byte array (Pascal), Word array (FORTRAN), by reference. An array of options and associated information. The format of an *opt* array is the same as the NetIPC *opt*. The options are equivalent to some RTE-A commands and calls dealing with program scheduling. Refer to the *RTE-A User's Manual* and *RTE-A Programmer's Reference Manual* for more information on the RTE-A commands and calls.

A detailed description of `RPMCreate` options is given later in this section under the subsection, "RPMCREATE Options." A list of `RPMCreate` options is presented in Table 7-1.

If no options are specified, the child program is assumed to reside in the current working directory of the session to which it logged on or in the `::programs` directory. RPM causes the child program to be restored with the clone name returned by `FmpRpProgram`. The child program is then scheduled with an `EXEC 10` (immediate schedule without wait) call with no parameters.

The total length of the *`opt`* array must be 996 bytes or less.

`pd`

`Byte array (Pascal), word array (FORTRAN), by reference`. An array of 16 bytes containing a unique program descriptor returned by RPM. This program descriptor is used to identify the scheduled child program. This value, randomly generated, is presumed to be unique across all nodes. A valid program descriptor is always a non-zero value. If `RPMCreate` is unsuccessful, *`pd`* is set to all zeroes.

The program descriptor is used in subsequent RPM calls to identify the child program.

`result`

`32-bit non-negative integer, by reference`. The result of the `RPMCreate` request; zero if no error. If *`result`* is not zero, an error has occurred. Errors are defined in the *NS-ARPA/1000 Error Message and Recover Manual*.

Table 7-1. RPMCREATE Options

Numeric Code	Description	RTE-A Equivalent
Group 1:		
23000	Set working directory name	`FmpSetWorkingDir`
Group 2:		
23010	Restore program	`RP` command
Group 3:		
20000	Pass string	none
23020	Assign partition	`AS` command
23030	Set program priority	`PR` command
23040	Change working set size	`WS` command
23050	Change VMA space size	`VS` command
23060	Change CDS code size	`CD` command*
23070	Change CDS data size	`DT` command*
		*not exactly like RTE
Group 4:	If used, only one can be specified:	
23080	Time list scheduling	`EXEC 12` call
23090	Immediate schedule w/o wait	`EXEC 10` call
23100	Queue schedule w/o wait	`EXEC 24` call
23110	Run program	`FmpRunProgram`

RPMCREATE

RPMCREATE Options

At the same time that a child program is scheduled, some equivalent RTE-A commands can be sent to the child program using the `RPMCreate` options. The options are listed in Table 7-1 and are explained in the following subsections.

The options and related data are placed into the *`opt`* array by using the `AddOpt` call. This call is documented in "Special NetIPC Calls" in the "Network Interprocess Communication" section of the *NS-ARPA/1000 User/Programmer Reference Manual*. The `AddOpt` call uses all the parameters listed below. However, the following subsections present only the *`optioncode`*, *`datalength`*, and *`data`* parameters, because these are the parameters that have specific values for each `RPMCreate` option.

```
ADDOPT(opt,argnum,optioncode,datalength,data,error)
```

AddOpt Parameters

opt	*Byte array (PASCAL); Word array (FORTRAN), by reference*. The *`opt`* parameter to which you want to add an argument. Refer to "Opt Parameter" in the "Network Interprocess Communication" section for information on the structure and use of this parameter. The total length of the *`opt`* array must be 996 bytes or less.
argnum	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The number of the argument to be added. The first argument number is zero.
optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. An `RPMCreate` option code. These codes are explained in the subsequent subsections.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of the data to be included. This information is provided in each `RPMCreate` option description on the following pages.
data	*Array, by reference*. A variable length array of data to be passed to the child program. Null strings are valid.
error	*16-bit integer, by reference*. The error code returned; zero if no error. Error codes are documented in the *NS-ARPA/1000 Error Message and Recover Manual*.

RPMCREATE

RPMCreate Option 20000—Pass String

RTE-A System Equivalent: none.

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN.* 20000 to indicate the "Pass String" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of *data* which is to be included in the *opt* array. A maximum of 256 bytes can be passed.
data	*Array, by reference.* A variable length array of data to be passed to the child program. Null strings are valid.

```
  0 1 2 3 4 5 6 7 8 9 10 ...   255   bytes
+--------------------------------+
|              data              |
+--------------------------------+
```

RPMCreate Option 23000—Set Working Directory

RTE-A FMP Equivalent: `FmpSetWorkingDir` call (documented in the *RTE-A Programmer's Reference Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN.* 23000 to indicate "Set Working Directory" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of *data* which is to be included in the *opt* array. The *data* is a working directory name. The *datalength* can be up to 63 bytes for an RTE working directory name.
data	*Packed array of characters (Pascal); word array (FORTRAN), by reference.* A packed array of characters specifying the working directory. The directory name must be fully-qualified. An exception would be if it is a subdirectory of the current working directory for the session created with the *login* parameter of `RPMCreate`. In this latter case, the current directory path can be omitted.

```
  0 1 2 3 4 5 6 7 8 9 10 ...   62   bytes
+-------------------------------+
|           directory           |
+-------------------------------+
```

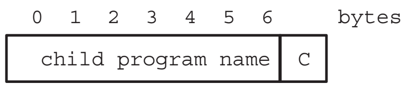
RPMCREATE

RPMCreate Option 23010—Restore Program

RTE-A FMP Equivalent: `FmpRpProgram` call (documented in the *RTE-A Programmer's Reference Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN.* 23010 to indicate the "Restore Program" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of *data* which is to be included in the *opt* array. Must be only one of the following values: 0, 6, or 7. No other values are allowed. • If the length is 0, there is no *data*. The child program is restored under a system-assigned name. The child is always restored as a permanent ID segment. • If the length is 6, it is assumed that the program name is specified in *data* and no cloning is to occur. • If the length is 7, both the program name and letter `C` are specified. The letter `C` signifies to create a clone name.
data	*Packed array of characters (Pascal); word array (FORTRAN), by reference.* A six- or seven-byte array. The first six bytes are the name under which the child program should be restored. If the name is not specified, the program will be restored under a system-assigned name. The returned name is the first five characters of the child program name (minus the directory path and file type extension). If the seventh byte contains the character `C`, a clone name is to be created. If the specified or assigned name from the first six bytes is already assigned, a clone name is created. For more information about cloning, refer to the *RTE-A User's Manual*.



RPMCREATE

RPMCreate Option 23020—Assign Partition

RTE-A System Equivalent: `AS` command (documented in the *RTE-A User's Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23020 to indicate the "Assign Partition" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be only one of the following values: 2 or 3. No other values are allowed. • If the length is 2, the 16-bit partition number is specified. The default is to assign the data section of the program to the reserved partition. • If the length is 3, the 16-bit partition number should be followed by a `C` for code section or `D` for data section.
data	*Array, by reference*. A three-byte array. The first two bytes are a 16-bit integer specifying the reserved partition number in which the child program is to run. The third byte is the character `C` or `D` to indicate either the code (`C`) or data (`D`) section. The code or data section of the program is assigned to the reserved partition. This argument applies only to CDS child programs. This argument can be in either upper or lower case.

```
     0    1    2     bytes
 ┌────────────┬────┐
 │ partition  │C or│
 │ number     │D   │
 └────────────┴────┘
```

RPMCreate Option 23030—Change Program Priority

RTE-A System equivalent: `PR` command (documented in the *RTE-A User's Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23030 to indicate the "Change Program Priority" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be a 2. No other values are allowed.
data	*16-bit integer, by reference*. An integer from 1 to 32767 specifying the child program priority.

```
   0       1     bytes
 ┌─────────────┐
 │  priority   │
 └─────────────┘
```

RPMCREATE

RPMCreate Option 23040—Modify Working Set Size

RTE-A System equivalent: `WS` command (documented in the *RTE-A User's Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23040 to indicate the "Modify Working Set Size" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be a 2. No other values are allowed.
data	*16-bit integer, by reference*. An integer from 2 to 1022 specifying the working set size in pages.

```
  0      1     bytes
+-----------+
| working   |
| set size  |
+-----------+
```

RPMCreate Option 23050—Modify VMA Size

RTE-A System equivalent: `VS` command (documented in the *RTE-A User's Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23050 to indicate the "Modify VMA Size" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be a 2. No other values are allowed.
data	*16-bit integer, by reference*. An integer from 32 to 32767 specifying the virtual EMA size in pages.

```
  0      1     bytes
+-----------+
| virtual   |
| EMA size  |
+-----------+
```

RPMCREATE

RPMCreate Option 23060—Modify Code Partition Size

RTE-A System equivalent: `CD` command (documented in the *RTE-A User's Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23060 to indicate the "Modify Code Partition Size" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be a 2. No other values are allowed.
data	*16-bit integer, by reference*. A 16-bit integer specifying the maximum number of code segments permitted to remain in memory at once. This number must be less than or equal to the actual number of code segments for the program.

```
     0    1      bytes
 ┌───────────┐
 │ code      │
 │ partition │
 │ size      │
 └───────────┘
```

RPMCreate Option 23070—Modify Data Partition Size

RTE-A System equivalent: `DT` command (documented in the *RTE-A User's Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23070 to indicate the "Modify Data Partition Size" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be a 2. No other values are allowed.
data	*16-bit integer, by reference*. A 16-bit integer specifying the size of the data partition in pages.

```
     0    1      bytes
 ┌───────────┐
 │ data      │
 │ partition │
 │ size      │
 └───────────┘
```

RPMCREATE

RPMCreate Option 23080—Time Scheduling

RTE-A System Equivalent: `EXEC 12` call (documented in the *RTE-A Programmer's Reference Manual*).

AddOpt Parameters

optioncode — *16-bit integer, by value in Pascal, by reference in FORTRAN*. 23080 to indicate the "Time Scheduling" option.

datalength — *16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array. Must be 2, 4, 6, 8, 10, or 12 bytes. No other values are allowed. The exact length depends on the parameters specified.

- If the length is 2, only *units* is specified. The call is an Absolute Start Scheduling call to schedule the child program immediately.
- If the length is 4, the *units* and *often* values are specified. The call is an Absolute Start Scheduling call to schedule the child program immediately and how often it should be scheduled.
- If the length is 6, the values for *units*, *often*, and *delay/hour* are specified. If the *delay/hour* parameter is negative, the call is an Initial Offset Scheduling call. If the *delay/hour* parameter is non-negative, the call is an Absolute Start Scheduling call.
- If the length is 8, the values for *units*, *often*, *hour*, and *min* are specified. The call is a Scheduling Absolute Starting Time call.
- If the length is 10, the values for *units*, *often*, *hour*, *min*, and *sec* are specified. The call is a Scheduling Absolute Starting Time call.
- If the length is 12, the values for *units*, *often*, *hour*, *min*, *sec*, and *msec* are specified. The call is a Scheduling Absolute Starting Time call.

data — *Array, by reference*. A 2 to 12 byte array with the following contents:

0 1	2 3	4 5	bytes
units	often	delay	

0 1	2 3	4 5	6 7	8 9	10 11	bytes
units	often	hour	min	sec	msec	

units — A resolution code that specifies the time units. In conjunction with parameter *often*, *units* specifies the time between each execution of the child program.

	`units` is one of the following values: 1 = tens of milliseconds 2 = seconds 3 = minutes 4 = hours
`often`	An integer value (0 to 4095) indicating the execution multiple or how often the program is to run.
`delay`	The initial offset. A negative number indicating the starting time of the first execution (not zero).

The following parameters collectively specify the starting time:

`hour`	The starting hour (0 to 23).
`min`	The starting minute (0 to 59).
`sec`	The starting second (0 to 59).
`msec`	The starting tens of milliseconds (0 to 99).

RPMCreate Option 23090—Program Scheduling (Immediate No Wait)

RTE-A System Equivalent: `EXEC 10` call (documented in the *RTE-A Programmer's Reference Manual*).

AddOpt Parameters

`optioncode` — *`16-bit integer, by value in Pascal, by reference in FORTRAN`*. 23090 to indicate the "Program Scheduling—Immediate No Wait" option.

`datalength` — *`16-bit integer, by value in Pascal, by reference in FORTRAN`*. The length in bytes of *`data`* which is to be included in the *`opt`* array. Must be one of the following values: 0, 2, 4, 6, 8, 10, 13, or greater than 13 bytes. Any parameter that is not specified defaults to zero. The exact length depends on the parameters specified:

- If the length is 0, all parameters are omitted and take their default value to be zero.
- If the length is 2, only *`pr1`* is specified.
- If the length is 4, *`pr1`* and *`pr2`* are specified.
- If the length is 6, *`pr1`*, *`pr2`*, and *`pr3`* are specified.
- If the length is 8, *`pr1`*, *`pr2`*, *`pr3`*, and *`pr4`* are specified.
- If the length is 10, *`pr1`*, *`pr2`*, *`pr3`*, *`pr4`*, and *`pr5`* are specified.
- If the length is 13 or more, all seven parameters are specified.

RPMCREATE

data — *Array, by reference*. A variable length array with the following contents:

```
 0  1  2  3  4  5  6  7  8  9  10  11 ... n  n + 1 bytes
+-----+-----+-----+-----+-----+----------+------+
| pr1 | pr2 | pr3 | pr4 | pr5 |   bufr   | bufln|
+-----+-----+-----+-----+-----+----------+------+
```

pr1, *pr2*, *pr3*, *pr4*, *pr5*	Five optional integer parameters to be passed to the child program. If any of the parameters *pr1*, *pr2*, *pr3*, *pr4*, or *pr5* are omitted, the remaining parameters all default to 0.
bufr	A variable length buffer containing data to be sent to the child program. The child program can recover the buffer by using the RTE `GETST` subroutine or the RTE string passage `EXEC 14` call. Refer to the RTE manual for usage. NOTE: Any string that is retrieved with `GETST` must be structured so that two leading commas exist in the string. `GETST` discards the information preceding the two commas and returns the string following them.
bufln	The length of *bufr*. If a positive integer, *bufln* indicates the number of words. If a negative integer, *bufln* indicates the number of bytes in *bufr*. If a positive integer, *bufln* indicates the number of words in *bufr*. If the *bufr* parameter is specified, the last two bytes of data are *bufln*.

RPMCREATE

RPMCreate Option 23100—Queue Program Scheduling

RTE-A System Equivalent: `EXEC 24` call (documented in the *RTE-A Programmer's Reference Manual*).

AddOpt Parameters

optioncode — *16-bit integer, by value in Pascal, by reference in FORTRAN.* 23100 to indicate the "Queue Program Scheduling" option.

datalength — *16-bit integer, by value in Pascal, by reference in FORTRAN.* The length in bytes of *data* which is to be included in the *opt* array. Must be one of the following values: 0, 2, 4, 6, 8, 10, 13, or greater than 13 bytes. Any parameter that is not specified defaults to zero. The exact length depends on the parameters specified:

- If the length is 0, all parameters are omitted and take their default value to be zero.
- If the length is 2, only *pr1* is specified.
- If the length is 4, *pr1* and *pr2* are specified.
- If the length is 6, *pr1*, *pr2*, and *pr3* are specified.
- If the length is 8, *pr1*, *pr2*, *pr3*, and *pr4* are specified.
- If the length is 10, *pr1*, *pr2*, *pr3*, *pr4*, and *pr5* are specified.
- If the length is 13 or more, all seven parameters are specified.

data — *Array, by reference.* A variable length array with the following contents:

```
 0 1  2 3  4 5  6 7  8 9  10 11 ... n  n + 1 bytes
+----+----+----+----+----+----------+------+
|pr1 |pr2 |pr3 |pr4 |pr5 |   bufr   |bufln |
+----+----+----+----+----+----------+------+
```

pr1, *pr2*, *pr3*, *pr4*, *pr5* — Five optional integer parameters to be passed to the child program. If any of the parameters *pr1*, *pr2*, *pr3*, *pr4*, or *pr5* are omitted, the remaining parameters all default to 0.

bufr — A variable length buffer containing data to be sent to the child program. The child program can recover the buffer by using the RTE `GETST` subroutine or the RTE string passage `EXEC 14` call.

`bufln`	The length of `bufr`. If a positive integer, `bufln` indicates the number of words. If a negative integer, `bufln` indicates the number of bytes in `bufr`. If a positive integer, `bufln` indicates the number of words in `bufr`. If the `bufr` parameter is specified, the last two bytes of data are `bufln`.

Caution Option 23100 should be used with extreme care and is recommended only for child programs which execute only for a very short duration. If this option is issued for a child program that is currently executing, RPM will suspend and will not be able to process other requests that arrive while waiting for the currently executing child program to terminate. If requests to RPM are frequent enough and RPM suspends for a long time, this may cause many requests to be rejected.

RPMCreate Option 23110—Program Scheduling

RTE-A FMP Equivalent: `FmpRunProgram` call (documented in the *RTE-A Programmer's Reference Manual*).

AddOpt Parameters

optioncode	*16-bit integer, by value in Pascal, by reference in FORTRAN*. 23110 to indicate the "Program Scheduling" option.
datalength	*16-bit integer, by value in Pascal, by reference in FORTRAN*. The length in bytes of *data* which is to be included in the *opt* array.
data	A variable length character string that contains the runstring. Note that the `XQ` command must be specified at the beginning of the runstring, or RPM will insert it. If the `RU` command is specified at the beginning of the runstring, RPM replaces it with an `XQ`. Also, the program name should be the same as the *progname* parameter of the `RPMCreate` call. An error is returned if this is not the case. This program name will be replaced by RPM by the name of the ID segment under which it is restored. The `IH` option of the RTE `RU` (Run Program) command is not permitted to follow the program name. Cloning can be inhibited by specifying option 23010 (Restore Program Option) beforehand in the *opt* array of the `RPMCreate` call. The last two bytes of *data* is a 16-bit integer indicating how `FmpRunProgram` is to handle the string parameter. The possible values are as follows: 1 The string is converted to uppercase and each group of one or more consecutive blanks is converted to a comma. 0 The string is not altered.

```
  0   1   2   . . .  n  n + 1     bytes
┌─────────────────────┬────────┐
│       string        │ 1 or 0 │
└─────────────────────┴────────┘
```

RPMGETSTRING

Allows the child program to retrieve strings passed to it by the parent program.

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

rpmstring	*Packed array of characters (Pascal); word array (FORTRAN), by reference.* A variable length array containing the string passed in the *opt* parameter of the `RPMCreate` call which scheduled this child program.
rpmstringlen (input/output)	*32-bit non-negative integer, by reference.* On input, *rpmstringlen* is the maximum byte length allowed for the *rpmstring*. On output, *rpmstringlen* indicates the actual length of the returned *rpmstring*. A string longer than what the buffer can accommodate will be truncated. In RTE-A the maximum string length retrieved with `RPMGetString` is 256 bytes. If there is no string received in *rpmstring*, an error is returned in *result*.
result	*32-bit non-negative integer, by reference.* The result of the `RPMGetString` request; zero if no error. If *result* is not zero, an error has occurred. Errors are defined in the *NS-ARPA/1000 Error Message and Recover Manual*.

RPMKILL

Terminates a specified child program scheduled by an `RPMCreate` call.

`RPMKILL (`*`pd`*`,`*`nodename`*`,`*`nodelen`*`,`<u>*`result`*</u>`)`

pd	*Byte array (Pascal); Word array (FORTRAN), by reference*. An array of 16 bytes containing the program descriptor returned by the `RPMCreate` call.
nodename	*Packed array of characters (Pascal); word array (FORTRAN), by reference*. A variable length array identifying the node on which the child program resides. The syntax of the node name is *node*`[.`*domain*`[.`*organization*`]]`, which is further described in the *NS-ARPA/1000 User/Programmer Reference Manual*. *Default*: You may omit the organization, organization and domain, or all parts of the node name. When organization or organization and domain are omitted, they will default to the local organization and/or domain. If the *nodelen* parameter is set to zero, *nodename* is ignored and the node name defaults to the local node.
nodelen	*32-bit non-negative integer, by value in Pascal, by reference in FORTRAN*. Length in bytes of the *nodename* parameter. If *nodelen* is zero (0), the *nodename* parameter is ignored. In this case, it is assumed that the parent is either terminating a dependent child program that it previously scheduled or the child program is on the local node.
result	*32-bit non-negative integer, by reference*. The result of the `RPMKill` request; zero if no error. If *result* is not zero, an error has occurred. Errors are defined in the *NS-ARPA/1000 Error Message and Recover Manual*.

RPM Error Codes

These error codes are returned in the `result` parameter of Remote Process Management (RPM) calls. Refer to the *NS-ARPA/1000 Error Messages and Recovery Manual* for more complete explanations of these error codes.

RPM Error Codes

Code	Meaning
0	The call was successful.
1	Network is down.
2	Illegal name length.
3	Illegal flags.
4	Illegal option or request code.
5	Illegal option format.
6	Invalid login or password.
7	Child program not found on child node.
8	Invalid program descriptor.
9	Remote process limit exceeded.
10	Insufficient memory to create a child program.
11	Security violation or device error.
12	Unknown internal error.
13	Bad RPM packet structure.
14	Network transport error.
15	Incompatible version number ID.
16	Unsupported RPM option or request.
17	`RPMCreate` request message too long.
18	Bad parameter in `opt` array parameter.
19	Invalid node name.
20	No RPM parameter string.
21	Illegal string length.
22	The child program terminated abnormally.
23	Unsupported RPM call.
302	Table is full.
304	RPM attach failed.
305	Schedule error.
306	Cannot find child program's ID segment.
307	Logoff failed.
308	Group 3 option error.

8

REMAT

Table of Contents

REMAT Operation

```
   CI>RU,REMAT
   $
```

Local RTE command processing now in effect.

```
   $SW,570,572,DS
```

Set up origin (570) and destination (572) nodes. Network user's security code (`DS`) must be specified.

```
   $EX
```

Terminate REMAT.

Note Physical node locations are designated by the notation `NODE1 = xx` and `NODE2 = xx` where *xx* is the Router/1000 node address assigned by the Network Manager. Logical node location, those locations obtained using the REMAT `SW` command, are designated by the notation `NODE1` (the origin node) and `NODE2` (the destination node).

REMAT Scheduling

REMAT can be scheduled with or without the following runstring parameters.

```
   REMAT  [ ,filedesc ]  [,log] [,list] [,severity code]
          [ ,input    ]
```

filedesc	The name of a command file that provides REMAT commands. All commands in the file must be preceded by a $.
input	The LU of the system input device. Must be less than 64. Default is your terminal.
log	The LU of the interactive message logging device. Must be less than 64. Default is input LU (if interactive) or the value returned by LOGLU.
list	The LU of the list device. Must be less than 64. Default is the log LU.
severity code	The error reporting code. If 0 is specified, all commands will be echoed and all errors will be reported (this is the default). If 1 is specified, the command echo is inhibited.

REMAT Commands

AT

Attaches to an account at a remote Session Monitor node. Do not use this command if you want the default account, if the remote node does not have Session Monitor, or if the remote node is an RTE-A system.

```
   AT[,user.group[/password]][user.group[/password]]
```

user.group/password	The account and optional password in the corresponding node for which a non-interactive session is to be created and to which subsequent REMAT commands will non-interactively attach. Previous session at this node will be released. For non-session access, this parameter is *password* only. The first parameter indicates the logon at `NODE1`; the second parameter is the logon at `NODE2`. The `AT` command must be specified with one or the other; it is meaningless without parameters. User, group, and password may each be up to 10 characters in length.

BC

Broadcasts a message to each node in your Nodal Routing Vector.

```
   BC,message
```

message	An ASCII string of up to 72 characters in length.

CL

Lists the mounted cartridges at the currently switched node (`NODE1`) on a local list LU.

```
   CL[,LU]
```

LU	The logical unit number for the local list device. Must be less than 64. If *LU* is not specified, the current list device set up by the `LL` command, or the list LU assigned by REMAT.

CR

Creates a disk file on the currently switched node (`NODE1`). No data is transferred to the file.

```
   CR,namr
```

namr	A file descriptor that describes a FMGR file. It must not be a logical unit number. All omitted subparameters (except filename) default to zero. However, the file type and file size must be specified as greater than zero or an error will be returned.

DE

Detaches from a remote account established at an RTE system with Session Monitor.

```
   DE[,N1][,N2]
```

N1	Specifies the current NODE1.
N2	Specifies the current NODE2.

DL

Lists the file directory of the currently switched node (NODE1) on the local list device.

```
   DL [ ,cartridge ] [,msc][,lu]
      [ ,namr      ]
```

cartridge	Cartridge identifier of a FMGR cartridge. Must be numeric, positive for cartridge reference number, negative for logical unit number. If omitted or zero, the directories of all cartridges mounted to the current account are listed.
msc	The system master security code. If specified correctly, the security code of each file will be listed.
namr	A file descriptor that describes a FMGR file. Minus signs may be used in the file name to specify a match with any single character.
lu	The logical unit number of the local list device. Must be less than 64. If specified, the directory will be listed on this LU. Default is the list device specified when REMAT was scheduled, or with the LL command.

DU

Transfers data from a file or logical unit at `NODE1` to a logical unit at `NODE2`.

```
   DU,namr,lu[,format]
```

namr	A file descriptor that describes a FMGR file, or logical unit (less than 64) from which the records will be dumped.
lu	The logical unit to which the records will be dumped. Must be less than 64.
format	Format of the data being transferred. Default is derived from *namr*, if *namr* is a file, otherwise default is ASCII. The format may be `AS` (ASCII), `BR` (Binary Relocatable), `BN` (Binary, no checksum is performed), or `BA` (Binary absolute, checksum is performed).

EX

Terminates REMAT.

```
   EX
```

FL

Closes a disk file at `NODE1` to users at a specified node.

```
FL,namr,node
```

namr	A file descriptor that describes a FMGR file. The *crn* subparameter of *namr* must be supplied and be non-zero.
node	The node number at which the user to whom the file is being closed resides. If *node* is −1, the file is closed to all users.

IO

Lists the system I/O configuration on RTE-A systems only.

```
IO
```

LC

Displays the local node Router/1000 node address.

```
LC
```

LI

Prints the contents of a file at `NODE1` to a logical unit at `NODE2`.

```
LI,namr[,lu]
```

namr	A file descriptor that describes a FMGR file. If the file has a negative security code, you must specify it.
lu	The optional logical unit where the file is to be listed. Must be less than 64.

LL

Changes or displays the LU of the list and/or log device.

```
LL[,list][,log]
```

list	The logical unit number of the new list device. Must be less than 64.
log	The logical unit number of the new log device. Must be less than 64.

LO

Loads an absolute program file from NODE1 into a memory-based RTE-A system at NODE2.

```
   LO,namr
```

namr	A file descriptor that describes a FMGR type 6 file (executable program file, memory-image) that contains the program to be loaded. The first five characters of the file name are used as the name of the program in building the program's ID segment.

PL

Lists all programs that are in memory at a remote memory-based RTE-A node.

```
   PL[,status]
```

status	One of the program status codes (ASCII) described under the command PL (list programs) in the *RTE-A Quick Reference Guide*. Only those programs with the specified status will be listed. If no option is specified, all programs, their status, priority, and point of suspension are listed.

PU

Removes a file from the disk at NODE1.

```
   PU,namr
```

namr	A file descriptor that describes a FMGR file.

QU

Queue schedules a program to run without wait at NODE1. If NODE1 is an RTE-A system, the program to be scheduled must have an ID segment and be loaded as a system utility.

```
   QU,pname[ [,p1][,p2][,p3][,p4][,p5] ]
           [ [,string]                 ]
```

pname	The name of the program to be scheduled.
p1,...,p5	Up to five optional parameters to be passed to the program.
string	ASCII string to be passed to the program. The command line, including the $ prompt if commands are being read from a non-interactive device or file, must not exceed 80 characters.

QW

Queue schedules a program to run with wait at NODE1. If NODE1 is an RTE-A system, the program to be scheduled must have an ID segment and be loaded as a system utility.

```
QW,pname [ [,p1] [,p2] [,p3] [,p4] [,p5] ]
         [ [,string]                     ]
```

pname	The name of the program to be scheduled.
p1,...,p5	Up to five optional parameters to be passed to the program.
string	ASCII string to be passed to the program. The command line, including the $ prompt if commands are being read from a non-interactive device or file, must not exceed 80 characters.

RN

Changes a file name to a new name. None of the file characteristics are changed except the name.

```
RN,namr,nuname
```

namr	A file descriptor that describes a FMGR file.
nuname	New file name unique to the disk cartridge. Security code and cartridge identifier cannot be altered.

RW

Schedules a program to run with wait at NODE1. If NODE1 an RTE-A system, the program to be scheduled must have an ID segment and be loaded as a system utility.

```
RW,pname [ [,p1] [,p2] [,p3] [,p4] [,p5] ]
         [ [,string]                     ]
```

pname	The name of the program to be scheduled.
p1,...,p5	Up to five optional parameters to be passed to the program.
string	ASCII string to be passed to the program. The command line, including the $ prompt if commands are being read from a non-interactive device or file, must not exceed 80 characters.

SD

Shuts down a session created by NS-ARPA at NODE1. Sessions are created at Session Monitor nodes only.

The SD command releases tables at NODE1 only, not at the node which created the session. Its purpose is to clean up when catastrophic events (such as rebooting) occur at the creating node, causing discrepancies between the creator's Process Number List (PNL) and NODE1's POOL. *Do not shut down a session with the SD command for any other reason*.

```
SD,session ID,NMSC
```

session ID	The session ID at NODE1 that you wish to shut down. (Your session ID can be found by using the NSINF utility.)
NMSC	The Network Management Security Code at the node where REMAT is running.

SL

Lists all program-to-program (PTOP) slave programs.

```
SL[,list lu]
```

list lu	The logical unit number of the local list device. Must be less than 64. Default is either the list device specified in the `LL` command or the list LU assigned by REMAT.

SO

Terminates a program-to-program (PTOP) slave program.

```
SO[,program name]
```

program name	The name of the program to be terminated. *If no program is specified, all current PTOP slaves are terminated at NODE1*. `NODE1` is specified in the `SW` command.

ST

Transfers data from `NODE1` and creates a file at `NODE2`.

```
ST,namr1,namr2[,format][,mode]
```

namr1 — A file descriptor that describes an existing FMGR file, or a logical unit number. Data is transferred from *namr1*. LU must be less than 64.

namr2 — A file descriptor that describes a FMGR file to which data will be transferred. Cannot be an LU number. The following *namr2* subparameters have non-standard default values.

type	If zero or not specified, defaults to the type of *namr* if *namr1* is a file. If *namr1* is not a file, default is type 3.
size	If given, must be positive; default is 10 blocks if *namr1* is an LU, or is equal to the size of *namr1* when *namr1* is a file.

format — The format of the data being transferred. Default is derived from *namr1* if *namr1* is a file, otherwise default is ASCII. The choices are:

`AS`	ASCII
`BR`	Binary relocatable; checksum is performed
`BN`	Binary; no checksum is performed
`BA`	Binary absolute; checksum is performed

mode — Transfer mode. May be entered when both *namr1* and *namr2* are files. A non-zero value causes both files to be opened as type 1 to increase the rate of data transfer. Extents are copied if both source and destination nodes support extendable type 1 or 2 files.

You cannot use non-zero transfer mode for storing hierarchical file system files with odd byte length records to nodes with non-hierarchical file systems.

SW

Changes or displays the origination and/or destination nodes of subsequent REMAT commands. The `SW` command can be used in four different ways.

To display the current values of `NODE1` and `NODE2`:

```
SW
```

To set `NODE1` and `NODE2`:

```
SW[,NODE1][,NODE2],security code
```

To set `NODE1` and `NODE2` to the local node address:

```
SW,LO[CAL]
```

To attach to a specific account.

```
SW[,NODE1[:user.group/password]][,NODE2[:user.group/password]],
   security code
```

NODE1	If a REMAT command requires one node it uses `NODE1`. If the command requires two nodes, then `NODE1` is the node from which action originates when a REMAT command is issued. Must be the Router/1000 node address of the desired node. (If `NODE1` is a neighboring node, may also be the negative logical unit number of that node.)
NODE2	If a REMAT command requires two nodes, `NODE2` is the node to which the results of the action taken are destined when a REMAT command is issued. Must be the Router/1000 node address of the desired node. (If `NODE2` is a neighboring node, may also be the negative logical unit number of that node.)
security code	The local node's Network User's security code assigned to the node at network initialization time.
`LO[`*CAL*`]`	A literal parameter that forces `NODE1` and `NODE2` to the local node number.
user.group/password	(Can be used to attach to specific accounts at DS/1000-IV nodes with Session Monitor only.) The account and optional password for which a non-interactive session will be created at the specified node and to which subsequent REMAT commands will non-interactively attach. The previous session at this node will be released. This parameter is not required if the default account is desired. For non-session access, this parameter is the password only. User, group, and password may each be up to 10 characters in length.

Table 8-1. Effect of SW

Command	NODE1	NODE2	Local
AT	Session created here (Session Monitor nodes only)	Session created here (Session Monitor nodes only)	
BC	Message delivered at all nodes	Message delivered at all nodes	Message delivered at all nodes
CL	Cartridges mounted here		List here
CR	File created here		
DE	Detaches session here (Session Monitor nodes only)	Detaches session here (Session Monitor nodes only)	
DL	File directory here		List here
DU	From file or lu	To lu	
EX			Only used here
FL	File closed here		
IO	From RTE-A		I/O configuration listed here
LC			Only used here
LI	From file	To lu	
LL			Only used here
LO	From file	To memory-based RTE-A	
PL	From RTE-A		Programs listed here
PU	File purged here.		
QU	Scheduled program here		
QW	Scheduled program here		
RN	File renamed here		
RW	Scheduled program here		
SD	Shut down a session here		
SL	Slave program here		List here
SO	Slave program here		
ST	From file or lu	To file or lu	
SW			Only used here
TE	Message sent here		
TR	Transfer to file at any node	Transfer to file at any node	Transfer to file at any node

TE

Sends a message to `NODE1`.

```
   TE,message
```

message	An ASCII string of up to 72 characters in length.

TR

Transfers control of REMAT to a file at any node or a logical unit.

```
      ┌ [,filedesc][,node] ┐
   TR │ [,-integer]        │
      └ [,lu]              ┘
```

filedesc	File name of transfer file containing RTE commands.
node	Positive node number or negative logical unit number of a node where the command file exists. If *filedesc* is an LU, node is ignored.
-integer	Negative integer that denotes a transfer back through the stacked transfer files. Current command file is not included in the count.
lu	Logical unit of input device.

WHZAT

Schedules the DS version of the RTE system status utility WHZAT.

```
   RW,WHZAT[,lu],option,nodeaddress
```

RW	Used to schedule WHZAT.
lu	The system LU to which the output will be printed. Must be less than 64. If this parameter is omitted, the output LU will default to system LU 1 at the node specified by *nodeaddress*.
option	The WHZAT program option. Refer to the appropriate programmer's reference manual for an explanation of the WHZAT program options.
nodeaddress	The Router/1000 node address of the node where the output is to be returned. This must be your local node to display the WHZAT output at your local system.

Note You *cannot* schedule WHZAT at `NODE1` if that node is an NS-ARPA/1000 system or a DS/1000-IV system with an RTE-A operating system.

Error Messages

The following error messages are returned to the current list device when an error is encountered by the program REMAT.

Table 8-2. REMAT Error Messages

Message	Meaning
`/LOGOFF:DSERR` *`SSEE(QQ)`*`,` `REPORTING NODE` *`NNNNN`*	This message can appear if the node being accessed by REMAT is a Session Monitor node.
`/LOGON:DSERR` *`SSEE(QQ)`*`,` `REPORTING NODE` *`NNNNN`*	This message can appear if the node being accessed by REMAT is a Session Monitor node.
`/REMAT: DSERR` *`SSEE(QQ)`*`,` `REPORTING NODE` *`NNNNN`*	See error code (*`EE`*) in specific subsystem (*`SS`*).
`/REMAT:` *`xxx`*	Numerical error message equivalent to FMGR error.

9

RMOTE

Table of Contents

RMOTE Operation

```
    CI>RU,RMOTE
    $
```

Local RTE command processing now in effect.

```
    $SW
```

Switch to the HP 3000 node.

```
    #HELLO user.group
```

Once the `HELLO` command is issued, standard HP 3000 command processing can take place.

```
    #EX
```

Exit RMOTE.

MPE Commands

RMOTE provides a remote interface to all MPE commands. In order to use RMOTE, you should be familiar with the HP 3000 and the MPE commands. These commands are described in the appropriate MPE commands reference manual.

RMOTE Scheduling

RMOTE can be scheduled with or without the following runstring parameters. Alternately, RMOTE can be scheduled from a program with DEXEC or EXEC call.

```
    REMAT [,filedesc] [,log] [,severity code]
          [,input   ]
```

filedesc	The name of the file that provides all input commands; a command file. All local commands must be preceded by the "$" prompt and all remote commands must be preceded by the "#" prompt. Comments must be preceded by an asterisk (*).
input	The LU of the input device for $STDIN. Default is your terminal, the Multiterminal Monitor LU, or the value returned by LOGLU.
log	The LU of an output device for $STDLIST requests and for logging errors. The default log LU is the input LU (if interactive) or the value returned by LOGLU.
severity code	The display code. Default is zero. If an invalid code is entered, it is defaulted to zero. The options are:

0	All commands are echoed on the log device. Any error causes an appropriate error message to be printed.
1	Inhibit command echo on log device.
2	Inhibit messages to log device unless error is severe enough to cause control to transfer to log device for command input.

RMOTE Commands

EX

Terminates RMOTE execution.

```
EX
```

LL

Changes the $STDLIST device.

```
LL,lu
```

lu	The logical unit number of the new $STDLIST device at the local node.

MO

Moves files between an HP 1000 and an HP 3000.

If input is from an interactive LU, RMOTE prompts for data with a slash ("/").
CONTROL D terminates a data entry.

HP 1000 to HP 3000 File Transfer Format

When a file is moved from an HP 1000 to an HP 3000 ("$" prompt), use the following command:

```
MO { ,filedesc } ,filename [,UN] [ :CC ]
   { ,lu       }                 [ :SP ]
```

filedesc	The file to be transferred.
lu	The logical unit that specifies the location of the file to be moved. Must be less than 64.
filename	The filename created on the HP 3000. If the length of the file to be transferred is more than 1023 records, a larger file size must be established on the HP 3000 before the move can be successfully completed. To set the file size, switch to the HP 3000 and type: `#FILE filename;DISK=number of records` Then switch back to the HP 1000 to complete the move under the HP 1000 "$" prompt. Any MPE file attribute can be changed from its default by using the `FILE` command. This equation does not alter the characteristics of an existing file.
UN	Indicates the file is to be written as fixed length records. RMOTE uses a default size of 80 characters, but this can be overridden by the MPE `FILE` command.
CC	Indicates column 1 is to be used for carriage control at MPE.
SP	Indicates the RTE file is spooling system format. I/O control embedded in the file will be translated to MPE carriage control characters.

MO

Moves files between an HP 3000 and an HP 1000.

HP 3000 to HP 1000 File Transfer Format

When a file is moved from an HP 3000 to an HP 1000 (“#” prompt), use the following command:

```
MO,filename{ ,filedesc  } [,UN]
           { ,lu[:SP]   }
```

filename	The name of the HP 3000 file to be transferred to the HP 1000.
filedesc	The name the HP 3000 file will acquire at the HP 1000.
lu	The logical unit number specifying the destination of the HP 3000 file on the HP 1000. If the LU is a magnetic tape drive, an EOF is written when the move completes. If the LU is a line printer, column one is interpreted as carriage control unless octal 200 (decimal 128) is added to set the V bit. A top of form is written when the move completes.
UN	Removes the last 8 characters of each line regardless of line length. This feature is useful for moving numbered MPE text files. Numbered files created by the MPE text editor have an 8-digit sequence number at the end of each line. The UN option would remove those eight digits on each line before moving the file.
SP DS/1000-IV and RTE-6/VM only	Indicates output is to be spooled to the RTE LU (at priority 99). If this is specified, RMOTE creates a spooling file RM*LUnn* on the spool disk, where *LU* is the logical number of the log device and *nn* is a number between 00 and 99 (e.g., *RM0900*). The file is purged when outspooling is completed.

ON, RU

Schedules a local HP 1000 program.

```
{ RU }
{ ON } ,program[,NOW][,parameters]
```

program	Name of the program to be scheduled. The program must have an ID segment and be loaded as a system utility.
NOW	If specified, schedules the program immediately. Commands with the NOW option are passed directly to RTE and are not upshifted.
parameters	Up to five parameters or a string can be passed to the program. If the first parameter is not provided, it is set to the value returned by LOGLU. If the fifth parameter is not provided, it is set to the negative session number obtained if RMOTE successfully establishes an HP 3000 session.

RW

Queues, with wait, a local HP 1000 program.

```
   RW,program[,parameters]
```

program	Name of the program to be scheduled. The program must have an ID segment and be loaded as a system utility.
parameters	Up to five parameters or a string to be passed to the program. If the first parameter is not provided, it is set to the value returned by `LOGLU`. If the fifth parameter is not provided, it is set to the negative session number obtained if RMOTE successfully establishes a session.

SV

Sets or changes the severity code.

```
   SV,severity
```

severity Severity may be:

0	Display error codes and echo commands on log device (default).
1	Inhibit command echo on log device.
2	Command echoed only if RMOTE error occurred.

SW

Establishes which HP 3000 a `HELLO` will be sent and toggles between the HP 3000 and HP 1000.

```
     ┌[,lu]                      ┐
   SW│                           │
     └,#x.25address[,x.25lu]     ┘
```

lu	The positive LU number of a BISYNC link to an HP 3000. If this parameter is omitted, the first BISYNC HP 3000 LU specified at NS-ARPA initialization is used.
x.25address	The X.25 network address assigned to the HP 3000. The pound sign (#) must appear before the address. The address can consist of up to fifteen digits. There is no default X.25 network address.
x.25lu	Specifies the physical X.25 link to use. Only necessary when more than one X.25 I/O board is in use. This is not the virtual circuit LU but is the network LU as defined by XINIT, the X.25 initialization program.

TR

Transfers command input to a file or logical unit at the local node.

```
    ┌          ┐
    │,filedesc │
 TR │,lu       │
    │,-integer │
    └          ┘
```

filedesc	The file name of transfer file where commands are to be read.
lu	Logical unit of input device where commands are to be read. Must be less than 64.
-integer	Negative integer that denotes a transfer back through the stacked command files. The current command file is not included in the count. The maximum number of command files is seven.

Error Messages

The following error messages are returned to the current list device when an error is encountered by the program RMOTE.

Error Messages

Message	Meaning
AUTO "BYE" FAILED	BYE generated automatically when EX command is entered with a HELLO outstanding has failed. May occur if the link has been disconnected since the HELLO was entered.
BAD LU	A negative LU number was specified in a MO command.
DS/1000 ERROR *nnn*	The reported numeric NS-ARPA/1000 error occurred during a file move operation.
DS/3000 ERROR *nnn*	The reported numeric DS/3000 error occurred during a file move operation.
HELLO FAILED OR LINE DOWN	HELLO command was not correct or could not be transmitted due to a line error.
ILLEGAL STATUS	RTE returned an SC03 scheduling error for an RU, ON, or RW command.
INVALID INPUT	Wrong or missing parameter or wrong prompt on transfer file input.
INVALID REMOTE LU	Either LU is not in the 3000 LU table when NSINIT was executed, or it is not the LU of the currently active session on the 3000.
LINK IS DISCONNECTED	The link to the HP 3000 is not functioning.
MPE FILE ERROR *nnn*	FS/3000 error occurred during a file move operation.
NEED "HELLO"	Attempt to send a command to the HP 3000 before issuing HELLO.
NEED TO RUN "DINIT"	Attempt to switch to remote node before local node has been initialized for communications to HP 3000.
NO BUFFER SPACE	Less than 256 words of memory are available for the PTOP file move buffer used with the MO command.
NO SLAVE AT 3000	Slave program does not exist as COPY3K.PUB.SYS.
NO SUCH PROGRAM	RTE returned an SC05 scheduling error for a RU, ON, or RW command.
NOT ENOUGH SAM	RTE returned an SC10 scheduling error for an RU, ON, or RW command.
NOT LOCAL COMMAND	HELLO or BYE under the $ prompt from RMOTE.
OLD COPY3K VERSION ON 3000 MOVE FAILED. LOAD NEW VERSION	Slave program on HP 3000 for MO command is not most recent version.
OLD RMOTE VERSION ON 1000 MOVE FAILED. LOAD NEW VERSION	Version of RMOTE is incompatible with slave program on HP 3000 for RMOTE MO command.
OVERWRITE?	Asked when the "to" file in a file move already exists.
PROGRAM BUSY	RU or ON specified a non-dormant program.
REQUEST FAILED	The HP 3000 rejected the last request.
RMOTE IO*xx*	RTE-reported I/O errors.
RMOTE SC*xx*	Indicates bad parameters.
RTE FILE ERROR *nnn*	An FMP error occurred during a file access.
TIMEOUT: NO REPLY FROM REMOTE	The HP 3000 did not respond to the last command: try again.
TR STACK OVERFLOW	The transfer stack is more than seven levels deep.
UNINITIALIZED @ READ	Local and/or remote ID sequences do not match the HP 3000.
WARNING--ILLEGAL OPTION	Printed only if severity = 0. SP specified with input from RTE LU or an RTE file in non-spooled format.
WARNING: RMOTE BUFFER TOO SMALL!	Printed only if severity = 0. RMOTE has insufficient buffer space at the end of the partition to hold some of the messages from the HP 3000.

10

Remote File Access

Table of Contents

The RFA calls that do not begin with DX can define files up to 16383 blocks. Those that do begin with DX (for example, DXCRE) can define larger files up to 32767 times 128 blocks in size.

RFA Common Parameters

The following list defines frequently used RFA parameters. Refer to this list when using calls that employ these parameters.

dcb (input/output)	The RFA Data Control Block. A 4-word array of any type. The *dcb* array resides in the same partition as your program and must not be modified. Programs having the standard 144-word *dcb* are accepted although 4-words is the minimum required. This *dcb* is used to store parameters when the file is opened so that any calls that reference this *dcb* will obtain the same file. The true *dcb* (144 words) is maintained by the RFA Monitor at the node where the file resides.
err (input/output)	Error code parameter. Produces an error code if an error condition is encountered during execution of a RFA call. Must be defined as a 16-bit integer. Upon successful completion of a DCRET call, the number of sectors allocated to the file is returned. Upon successful completion of a DOPEN call, a value representing the file type is returned. If a value less than zero is returned, DSERR may be called to obtain an ASCII message describing the problem.
len (input)	Length parameter. A value that declares the length of data for a read or write request (DREAD or DWRIT). Must be defined as a 16-bit integer. This value is limited to a maximum of 128 words. In extended file calls, *len* can be up to 512 words long.
cr (input)	Cartridge identifier parameter. Must be defined as an array of two 16-bit integers. Word 1 declares the type of cartridge reference. It is an integer value that can be positive, negative, or zero. If positive, the file search is restricted to the cartridge reference declared by the specified integer value. The Session Monitor nodes, any private, group, or system cartridge available to this specific user's account will be searched. If negative, the file search is restricted to the logical unit number declared by the specified integer value. If zero, the file search is not restricted to any particular cartridge. For DCRET, if *cr* is zero, the file will be allocated on the first cartridge encountered that has enough room for the file; calls other than DCRET search the cartridges in the order in which they were mounted until the file is found. Word 1 is the cartridge identifier. If positive, it specifies the crn; if negative, it specifies the LU. Word 2 declares the address of the node at which the call is to be executed. If this value matches the address of the local node, or is −1, the call is executed locally. This parameter may also be specified as the negative value of the communications line logical unit number. If this parameter is omitted, word 1 defaults to zero and word 2 defaults to −1 (local node).
erlc (output)	An error condition location variable. Must be defined as a 16-bit integer. If an error condition is encountered, the address of the node at which the error occurred is returned.

DAPOS/DXAPO

DAPOS and DXAPO set the address of the next record to be accessed within a file by calling these routines. The record position set may be defined via a prior call to `DLOCF`. `DAPOS` is equivalent to the FMGR routine `APOSN`; `DXAPO` is equivalent to `EAPOS`. This routine cannot be used with Type 0 (non-disk) files.

The *rb* and *ioff* parameters must be present for all files having variable record length to ensure correct operation.

```
    DAPOS(dcb,err,rec[,rb][,off][,erlc])

    DXAPO(dcb,err,rec[,rb][,off][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See “RFA Common Parameters” for more information.)
err	*16-bit integer*. Error return variable. (See “RFA Common Parameters” for more information.)
rec	*16-bit integer/32-bit integer*. Next record. A variable set to the number of the next sequential record in the file. Can be determined by a prior call to `DLOCF` or `DXLOC`. For `DXAPO`, *rec* is a double-word variable with the same meaning.
rb	*16-bit integer/32-bit integer*. Relative block address of the next record. A variable set to the block number of the next record. This parameter is required for files of type 3, 4, 5, 7, or greater (variable length records). For `DXAPO`, *rb* is a double-word variable containing the block number of the next record.
off	*16-bit integer*. Block offset of next record. A variable set to the offset in the block of the next record. This parameter is required for files of Type 3 or greater.
erlc	*16-bit integer*. Optional error condition location. (See “RFA Common Parameters” for more information.)

DCLOS/DXCLO

Following access operations, you may close a file using a call to `DCLOS` or `DXCLO`. `DCLOS` is equivalent to the FMGR call `CLOSE`; `DXCLO` is equivalent to `ECLOS`.

```
    DCLOS(dcb,err[,trun][,erlc])

    DXCLO(dcb,err[,trun][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See “RFA Common Parameters” for further information.)
err	*16-bit integer*. Error return variable. (See “RFA Common Parameters” for further information.)
trun	*16-bit integer/32-bit integer*. File truncation variable. Contains an integer value that defines the number of blocks to be deleted from the file upon closing. If zero or not specified, no truncation occurs. If negative, only file extents are truncated. If greater than the number of blocks in the file, no action occurs. If equal to the number of blocks in the file, the file is purged. For `DXCLO`, *trun* is a double-word variable containing a 32-bit number of blocks to be deleted from the main file at closing.
erlc	*16-bit integer*. Error condition location. (See “RFA Common Parameters” for further information.)

DCONT

The RTE input/output control requests to Type 0 files are transmitted using this routine. DCONT is equivalent to the FMGR call FCONT.

```
   DCONT(dcb,err,con1[,con2][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
con1	*16-bit integer*. Function code. A variable containing a numeric value defining an input/output function (see the following table).
con2	*16-bit integer*. Function sub-code. A variable containing a numeric value. This sub-code is required for some functions. (See the following discussion.)
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)

Function Code

Bits 6 through 10 of parameter *con1* are used for the function code.

BITS:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0						0	0	0	0	0	0

The function codes are defined in Table 10-1. These function codes are driver dependent. Consult the appropriate RTE driver reference manual for more information.

Function Sub-Code

Function sub-codes are required for line spacing (function 11) and finding files (function 27). If the function code is 11 octal, then DCONT expects a value in *con2*. This value controls output line spacing on the line printer or a keyboard display device:

- 0 to suppress line spacing on the next line
- >0 to indicate the number of lines to space before the next line
- <0 to page eject the line printer; space specified lines on keyboard device

If the function code is 27 octal, then DCONT expects a value in *con2*. This value declares the absolute file number to be located in the range 1 through 255.

Table 10-1. DCONT Function Codes

FUNCTION CODE (OCTAL)	FUNCTION	DEVICE
00	Unused	Magnetic Tape Cartridge Tape Unit
01	Write end-of-file	Magnetic Tape Cartridge Tape Unit
02	Backspace one record	Magnetic Tape Cartridge Tape Unit
03	Forward space one record	Magnetic Tape Cartridge Tape Unit
04	Rewind	Magnetic Tape Cartridge Tape Unit
05	Rewind Standby Rewind	Magnetic Tape Cartridge Tape Unit
06	Actual Device Status	Magnetic Tape Cartridge Tape Unit
07	Set end-of-tape	Paper tape/TTY
10	Generate leader/Write end-of-file if not just written or not at load point	Paper tape/TTY Cartridge Tape Unit
11*	List output line spacing	Line printer
12	Write 3 inch inter-record gap	Magnetic Tape
13	Forward space one file	Magnetic Tape Cartridge Tape Unit
14	Backspace one file	Magnetic Tape Cartridge Tape Unit
15	Conditional top-of-form	Line Printer or Display Device
20	Enable terminal—allows terminal to schedule its program with any keystroke	Codes 20-27 are defined for a keyboard terminal (DVR00). Refer to the DVR00 manual, 29029-60001 for other uses.
21	Disable terminal—inhibits scheduling of terminal program	
22	Set timeout—sets new timeout interval	
23	Ignore all further requests until the request queue is empty, an input request is received, or a restore control request is received	
24	Restore output processing (this request is usually not necessary)	
26	Write end-of-data	Cartridge Tape Unit
27**	Locate file number	Cartridge Tape Unit

* When function code 11 is specified in `con1`, then `con2` must be included in the parameter list to specify the particular line spacing.

** When function code 27 is specified in `con1`, then `con2` must be included in the parameter list to specify the particular file number.

DCRET/DXCRE

The `DCRET` and `DXCRE` routines define a new file. `DCRET` is equivalent to the FMGR call `CREAT`; `DXCRE` is equivalent to `ECREA`.

```
DCRET(dcb,err,name,size,type[,secu][,cr][,erlc])

DXCRE(dcb,err,name,size,type[,secu][,cr][,xsize][,erlc])
```

dcb	*Four-word array*. Data Control Block. (See “RFA Common Parameters” for more information.)
err	*16-bit integer*. Error return variable. (See “RFA Common Parameters” for more information.)
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. File name. A 3-word array containing the ASCII-coded name of the file to be created. Must be a FMGR file.
size	*Array of 16-bit integers/Array of 32-bit integers*. File size. A 2-word array in which word 1 contains a positive value declaring the number of blocks to be allocated for this file. Word 2 is used only for Type 2 files. It contains the record length of the file specified in number of words. For `DXCRE`, a two-entry array where each entry is a 32-bit integer. The first entry contains the file size in double-word number of blocks. The second entry is used only for Type 2 files and is the double-word record length.
type	*16-bit integer*. File type (0 − 32767).
secu	*16-bit integer*. File security code. A variable in the range 0 through −32767 or +32767. A positive value declares file write protection. A negative value declares file read and write protection. A value of 0 declares no file protection.
cr	*Array of 16-bit integers*. Cartridge reference label. A 2-word array which defines cartridge search and node destination conditions. (See “RFA Common Parameters” for more information.)
xsize	*Array of 16-bit integers*. Actual created file size in sectors. In `DXCRE`, an optional two-word array that contains the actual file size if `DXCRE` is successful.
erlc	*16-bit integer*. Error condition location. (See “RFA Common Parameters” for more information.)

DLOCF/DXLOC

These calls are used to retrieve the location of the current record pointer within disk files. `DLOCF` is equivalent to the FMGR routine `LOCF`; `DXLOC` is equivalent to `ELOCF`.

```
DLOCF(dcb,err,rec[,rb][,off][,sec][,lu][][,recsz][,erlc])

DXLOC(dcb,err,rec[,rb][,off][,sec][,lu][,ty][,recsz][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See “RFA Common Parameters” for more information.)
err	*16-bit integer*. Error return variable. (See “RFA Common Parameters” for more information.)
rec	*16-bit integer/32-bit integer*. Next record. A variable to which the number of the next sequential record in the file is returned. For `DXLOC`, *rec* is a double-word variable containing the next sequential record number.
rb	*16-bit integer/32-bit integer*. Relative block number containing the next record. A variable to which the number of the next block is returned. For Type 0 files, nothing is returned. For Type 1 files, *rb*=*rec*−1. If the file is extended, extents are accounted for within the value returned. For `DXLOC`, *rb* is a double-word variable with the same meaning.
off	*16-bit integer*. Offset of the next record in the block. A variable to which the location of the next word within the record is returned. For Type 0 files, nothing is returned.
sec	*16-bit integer/32-bit integer*. File size in sectors. A variable to which the number of sectors allocated to the file at creation is returned. To determine the block count, divide the sector count by 2. For Type 0 files, nothing is returned. For `DXLOC`, *sec* is a double-word variable.
lu	*16-bit integer*. Logical unit number. A variable to which the logical unit number of the device upon which the file resides is returned.
ty	*16-bit integer*. File type. A variable to which the file type determined at opening is returned.
recsz	*16-bit integer*. Record size. A variable to which the record length is returned for Type 1 and Type 2 files or the read/write access code for Type 0 files. This parameter is not applicable to a file type equal to or greater than Type 3.
erlc	*16-bit integer*. Error condition location. (See “RFA Common Parameters” for more information.)

DNAME

Renames an existing file. `DNAME` is equivalent to the FMGR routine `NAMF`.

```
DNAME(dcb,err,name,nname[,secu][,cr][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. File's current name. A 3-word array containing the ASCII coded current name of the file. Must be a FMGR file.
nname	*Integer array (FORTRAN); Packed character array (PASCAL)*. The file's new name. A 3-word array containing the ASCII coded new name of the file. Must be a FMGR file.
secu	*16-bit integer*. File security code. A variable in the range 0 through +32767 or −32767. This parameter must be declared if the file specified in the *name* parameter was created having a security code.
cr	*Array of 16-bit integers*. Cartridge reference label. A 2-word array which defines cartridge search and node destination conditions. (See "RFA Common Parameters" for more information.)
erlc	*16-bit integer*. Optional error condition location. (See "RFA Common Parameters" for more information.)

DOPEN

Opens defined files to access by your programs. DOPEN is equivalent to the FMGR call OPEN.

```
   DOPEN(dcb,err,name[,optn][,secu][,cr][,erlc])
```

dcb	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. File name. A 3-word array containing the ASCII coded name of the file to be opened. Must be a FMGR file.
optn	*16-bit integer*. File access specifications. A variable containing a numeric value that specifies non-standard conditions upon opening a file. For a detailed description of the options available, refer to the *NS-ARPA/1000 DS/1000-IV Compatible Services Reference Manual*.
secu	*16-bit integer*. If the file was created (see DCRET/DXCRE call description) with a non-zero security code, this parameter must be specified and it must match the original security code assigned to the file.
cr	*Array of 16-bit integers*. Cartridge reference label. A 2-word array which defines cartridge search and node destination conditions. (See "RFA Common Parameters" for more information.)
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)

DPOSN/DXPOS

These calls are used to position any type of file to a point relative to its current position or to a specific record number. `DPOSN` is equivalent to the FMGR call `POSNT`; `DXPOS` is equivalent to `EOPSN`.

```
DPOSN(dcb,err,nur[,rec][,erlc])

DXPOS(dcb,err,nur[,rec][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See “RFA Common Parameters” for more information.)
err	*16-bit integer*. Error return variable. (See “RFA Common Parameters” for more information.)
nur	*16-bit integer/32-bit integer*. Record count or record number. A variable containing either the number of records to skip forward (positive value) or backwards (negative value), or the absolute record number to which the file is to be positioned (positive integer only). The *rec* parameter determines how *nur* is interpreted. For DXPOS, *nur* is a double-word variable with the same meaning.
rec	*16-bit integer*. A variable that, if zero, declares that *nur* is the number of records to skip. If *rec* is not zero, *nur* is interpreted as an absolute record number.
erlc	*16-bit integer*. Error condition location. (See “RFA Common Parameters” for more information.)

DPURG

A call to `DPURG` removes a file from the file directory. `DPURG` is equivalent to the FMGR routine `PURGE`.

```
   DPURG(dcb,err,name[,secu][,cr][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. File name. A 3-word array containing the ASCII coded name of the file to be purged. Must be a FMGR file.
secu	*16-bit integer*. If the file was created (refer to `DCRET` earlier) with a non-zero security code, this parameter must be specified and it must match the original security code assigned to the file.
cr	*Array of 16-bit integers*. Cartridge reference label. A 2-word array which defines cartridge and node destination conditions. (See "RFA Common Parameters" for more information.)
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)

DREAD/DXREA

These routines read data from your file (currently open to the Data Control Block) into your program's data buffer. `DREAD` is equivalent to the FMGR call `READF`; `DXREA` is equivalent to `EREAD`.

```
   DREAD(dcb,err,buf,len[,rlen][,num][,erlc])

   DXREA(dcb,err,buf,len[,rlen][,num][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
buf	*Integer array (FORTRAN); Packed character array (PASCAL)*. Data buffer. An array, with a size equal to or greater than the value of the *len* parameter, into which the requested data is placed by the system.
len	*16-bit integer*. Data length. A variable specifying the number of words to be read. (See "RFA Common Parameters" for more information.)
rlen	*16-bit integer*. Actual length (in words) of data read. A variable to which the actual count of words transferred is returned. Set to −1 if end-of-file is read.
num	*16-bit integer/32-bit integer*. Record number. A variable that, if positive, contains the record number from which data is to be read. If negative, *num* is the number of records to backspace. If omitted, the record at the current position is read. Meaningful for type 1 and type 2 files only. For `DXREA`, *num* is a double-word variable >32767.
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)

DSTAT

This routine is called to obtain status information for all mounted cartridges in the cartridge directory. `DSTAT` is equivalent to the FMGR routine `FSTAT`.

```
DSTAT(stat,err,dest[,erlc][len][,form][,op][,add])
```

stat	*Integer array*. Status buffer. An array to which the cartridge directory status is returned. Default is 125 words.
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
dest	*16-bit integer*. Router/1000 node address of the destination node. A variable that contains a decimal value specifying the nodal address for which the status information is to be obtained. The value of this parameter may either be positive to denote the node number or negative to denote the communications line logical unit number.
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)
len	*16-bit integer*. Length in words of buffer *stat*. Default is 125 words.
form	*16-bit integer*. If *form* is zero, the disk directory will be written in FORMAT I (default). If it is non-zero, the disk directory will be written into the buffer in FORMAT II. This is exactly as it appears on disk. (Refer to the following tables for information on the two formats.)
op	*16-bit integer*. A one-word variable specifying the type of cartridges about which information is to be returned. If equal to 1, all disks mounted to the system are returned in *stat*. If equal to 0, system disks and non-session disks are returned in that order. (If *op* is equal to 0 and the remote node is an RTE-6/VM system with Session Monitor, private and group disks mounted to the session and system disks are returned, in that order.)
add	*16-bit integer*. Returned non-zero by the system if not all of the cartridge list could be returned in *stat* (buffer not large enough).

DSTAT

The formats that can be used with the `DSTAT` call are shown below.

Table 10-2. *stat* Format I

Word	Contents	Cartridge
1	Logical Unit Number	First Cartridge
2	Last FMP Track	
3	Cartridge Reference Number	
4	Lock Word	
5	Logical Unit Number	Second Cartridge
6	Last FMP Track	
7	Cartridge Reference Number	
8	Lock Word*	
9	Logical Unit Number	Third Cartridge
.	.	
.	.	
.	.	
	0 = no more disks	
* Lock word is the ID segment address of the locking program. If 0, the cartridge is not locked.		

Table 10-3. *stat* Format II

Word	Contents	Cartridge
1	Lock Word | Logical Unit#	First Cartridge
2	Last FMP Track	
3	Cartridge Reference Number	
4	ID**	
5	Lock Word* | Logical Unit#	Second Cartridge
6	Last FMP Track	
7	Cartridge Reference Number	
8	ID**	
9	Lock Word* | Logical Unit#	Third Cartridge
.	.	
.	.	
.	.	
	0 = no more disks	
* Lock word is the offset of the ID segment in the Keyword Table or 0 (not locked). If 0, the cartridge is not locked.		
** ID identifies who mounted the cartridge (user or group ID).		

DWIND

DWIND may be used to rewind (position to the beginning of first record) a Type 0 file or set disk files so that the next record in the file is the first record (position to beginning of first record). `DWIND` is equivalent to the FMGR call `RWNDF`.

```
   DWIND(dcb,err[,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)

DWRIT/DXWRI

These routines write data from a data buffer to your file (currently open to the Data Control Block). `DWRIT` is equivalent to the FMGR call `WRITF`; `DXWRI` is equivalent to `EWRIT`.

```
   DWRIT(dcb,err,buf,len[,num][,erlc])

   DXWRI(dcb,err,buf,len[,num][,erlc])
```

dcb (input/output)	*Four-word array*. Data Control Block. (See "RFA Common Parameters" for more information.)
err	*16-bit integer*. Error return variable. (See "RFA Common Parameters" for more information.)
buf	Data buffer; an array of a size equal to or greater than the value of the *len* parameter.
len	*16-bit integer*. Data length variable. (See "RFA Common Parameters" for more information.)
num	*16-bit integer/32-bit integer*. Record number. A variable containing the record number to which data is to be transferred (if positive), or the number of records to backspace (if negative). Used only for type 1 and 2 files. If omitted, record at current file position is written to. For `DXWRI`, *num* is a double-word variable up to (2**31)−1.
erlc	*16-bit integer*. Error condition location. (See "RFA Common Parameters" for more information.)

HP 3000 RFA Calls

You can issue calls from your NS-ARPA/1000 application programs to a set of compatible HP 3000 RFA intrinsics. These intrinsics provide file access at NS/3000 and DS/3000 nodes. Only a brief description of the HP 3000 intrinsics is shown in the following pages. Refer to the appropriate MPE intrinsics manual for a complete description of these calls. In addition, HP 3000 to HP 1000 RFA is discussed in the *DS/3000 HP 3000 to HP 1000 Reference Manual*.

Table 10-4. HP 3000 Intrinsics

HP 1000 Intrinsic	Equivalent MPE Intrinsic	Description
FCHEK	FCHECK	Obtains file I/O error information. It can be used when a call to a file intrinsic returns a condition code value that indicates an I/O error.
FCLOS	FCLOSE	Terminates access to a file by your program. When FCLOS is executed, buffers and control blocks through which you accessed the file are deleted and the device on which the file resides is deallocated. If you do not issue a call to the FCLOS intrinsic for each file opened during your session, MPE will issue FCLOS calls automatically when your session is terminated.
FCNTL	FCONTROL	Performs special control operations on a file or device.
FINFO	FGETINFO	Obtains file access and status information. Once a file is opened, a call to FINFO can be issued to obtain this information.
FLOCK	FLOCK	Dynamically locks a file for exclusive access by your program.
FOPEN	FOPEN	Opens an HP 3000 file for access by your program. When FOPEN is executed, a file number is returned. This file number must be used by your program in all subsequent file references in order to access the proper file. You must explicitly declare FOPEN as an integer function in your FORTRAN 77 and Pascal/1000 programs.
FPOIN	FPOINT	Sets the logical record pointer to any record within a disk file. The file must have only fixed-length records.
FRDIR	FREADDIR	Performs a read operation on a specified logical record from a disk file to the user's buffer. The file must have only fixed-length or undefined-length records.
FRDSK	FREADSEEK	Seeks out and performs a transfer of a specific logical record from a disk file to a buffer prior to a call to the FRDIR intrinsic. The file referenced must allow I/O buffering and have fixed-length or undefined-length records.
FREAD	FREAD	Performs a read operation of a logical record from a file on any device to the user's buffer. The record read is determined by the current position of the logical record pointer. This intrinsic returns an integer value representing the number of words or bytes read. You must explicitly declare FREAD as an integer function in your FORTRAN 77 and Pascal/1000 programs.
FRLAB	FREADLABEL	Performs a read operation on your disk file label.
FRLAT	FRELATE	Obtains information about the interactive/duplicative attributes of a specified file pair. A value is returned that represents the current attributes of the files. You must explicitly declare FRLAT as an integer function in your FORTRAN 77 and Pascal/1000 programs.
FRNAM	FRENAME	Renames a disk file.
FSPAC	FSPACE	Performs forward or backward spacing over a specified number of logical records on a disk file, or physical records on a magnetic tape file. The file must have fixed-length or undefined-length records.
FSTMD	FSSETMODE	Sets or resets file access modes such as automatic error recovery, critical output verification, and user terminal control. Any file access mode set by a call to FSTMD remains in effect until either reset by another call to FSTMD or until the file is closed.
FUNLK	FUNLOCK	Dynamically unlocks a file that was previously locked via a call to FLOCK.
FUPDT	FUPDATE	Performs a write update of a logical record to a disk file.
FWDIR	FWRITEDIR	Performs a write operation of a specified logical record to a disk file from a user's buffer. The file must have only fixed-length or undefined-length records.
FWLAB	FWRITELABEL	Performs a write operation of your label to a disk file. FWLAB overwrites any existing label.
FWRIT	FWRITE	Performs a write operation of a logical record from a user's buffer to a file on any device. Following FWRIT execution, the logical record pointer is set to the record immediately following the record written.

11

DEXEC

Table of Contents

DEXEC Call Syntax

Parametric compatibility is maintained between DEXEC calls and EXEC calls with one additional parameter: DEXEC calls include a destination node parameter, `dest`, that declares the network node at which the call to DEXEC is to be processed. The `dest` parameter may be either a positive value declaring the Router/1000 node address of the destination node, or a negative value declaring the communication line logical unit number. A value of negative one (−1) indicates the local address.

The DEXEC call syntax in this section applies to destination nodes with RTE-A operating systems only. The syntax used when addressing DS/1000-IV nodes with RTE-6/VM operating systems may be different. It is therefore recommended that you refer to the appropriate RTE reference manual for the equivalent EXEC call syntax.

DEXEC 1 (Remote Read)

Transfers one record from an I/O device to a buffer. (Read or write requests that reference disk LUs are not supported.)

```
   DEXEC(dest,code,cnwd,bufr,bufl[,prm1][,prm2][,0,0,keywd])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* equals the local node address.
code	*16-bit integer*. Read request code = 1.
cnwd	Control word. Unlike an EXEC 1 call, bit 11 of *cnwd* in a DEXEC 1 call cannot be used to specify control information to the driver in RTE-A systems. Refer to the "Interactive Write/Read" call description for more information on this restriction. (For further information on this parameter, refer to the *RTE-A Programmer's Reference Manual*.) Do not specify the logical unit number of any NS-ARPA/1000 communication line in the control word. Doing so may destroy the integrity of your network.
bufr	Read buffer.
bufl	*16-bit integer*. Read buffer length (+ for words, − for characters). The maximum buffer size for a remote DEXEC read operation is 512 words (−1024 characters). If the Z-bit in the control word is equal to 1 (indicating double word buffering), the combined buffer length, *bufl* + *prm2*, must be less than or equal to 512 words.
prm1	See the EXEC call descriptions in the *RTE-A Programmer's Reference Manual* for an explanation of this parameter. If the Z-bit in the control word is equal to 1 (indicating double buffering), this parameter specifies the address of the second buffer.
prm2	See the EXEC call descriptions in the *RTE-A Programmer's Reference Manual* for an explanation of this parameter. If the Z-bit in the control word is equal to 1 (indicating double buffering), this parameter specifies the length of the second buffer (+ for words, − for characters). The combined buffer length, *bufl* + *prm2*, must be less than or equal to 512 words.
0,0	Formal place holders that must be included whenever *keywd* is specified.
keywd	The locked LU's keyword number. (See the *RTE-A Programmer's Reference Manual* for more information.)

Interactive Write/Read

The interactive WRITE/READ uses the same request code as a Remote Read call and is useful for programs that must communicate with an operator using a question/answer format for the exchange of information. (Read or write requests that reference disk LUs are not supported.)

```
   DEXEC(dest,code,cnwd,bufr,bufl[,prm1][,prm2][,0,0,keywd])
```

dest	`16-bit integer`. The Router/1000 node address of the node where the call is executed. A value of −1 indicates local node. Local execution will also be performed whenever `dest` equals the local node address.
code	`16-bit integer`. Read request code = 1.
cnwd	Control word. Set bit 11 equal to 1 to indicate interactive write/read. Define the read buffer in *bufr*/*bufl* and write buffer in *prm1*/*prm2*. Bit 12 must equal zero. (See the *RTE-A Programmer's Reference Manual* for more information on this parameter.) Because DEXEC 1 uses bit 11 to indicate an interactive write/read, this bit cannot be used to send control information to the driver in RTE-A systems. (Bit 11 can be specified to the driver in DEXEC 2 and 3 calls.) Do not specify the logical unit number of any NS-ARPA/1000 communication line in the control word. Doing so may destroy the integrity of your network.
bufr (input/output)	Read buffer.
bufl	`16-bit integer`. Read buffer length (+ for words, − for characters). The maximum combined write/read buffer size (`bufl + prm1`) is 512 words (−1024 characters).
prm1	Write buffer.
prm2	`16-bit integer`. Write buffer length (+ for words, − for characters). The write buffer length should be less than or equal to the read buffer length. The maximum combined write/read buffer size (`bufl + prm1`) is 512 words (−1024 characters).
`0,0`	Formal place holders which must be included whenever `keywd` is specified.
keywd	The locked LU's keyword number. (See the *RTE-A Programmer's Reference Manual* for more information on this parameter.)

DEXEC 2 (Remote Write)

A call to DEXEC with request code 2 results in the transfer of one record from a buffer to an I/O device. (Read or write requests that reference disk LUs are not supported.)

```
   DEXEC(dest,code,cnwd,bufr,bufl[,prm1][,prm2][,0,0,keywd])
```

dest	`16-bit integer`. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever `dest` is equal to the local node address.
code	`16-bit integer`. Write request code = 2.
cnwd	Control word. (See the *RTE-A Programmer's Reference Manual* for an explanation of this parameter.) Do not specify the logical unit number of any NS-ARPA/1000 communication line in the control word. Doing so may destroy the integrity of your network.
bufr	Write buffer.
bufl	`16-bit integer`. Write buffer length (+ for words, − for characters). The maximum buffer size for a remote DEXEC write operation is 512 words (−1024 characters). If the Z-bit in the control word is equal to 1 (indicating double buffering), the combined buffer length, `bufl + prm2`, must be less than or equal to 512 words.
prm1	See the EXEC call descriptions in the *RTE-A Programmer's Reference Manual* for an explanation of this parameter. If the Z-bit in the control word is equal to 1 (indicating double buffering), this parameter specifies the second buffer.
prm2	See the EXEC call descriptions in the *RTE-A Programmer's Reference Manual* for an explanation of this parameter. If the Z-bit in the control word is equal to 1 (indicating double buffering), this parameter specifies the length of the second buffer (+ for words, − for characters). The combined buffer length, `bufl + prm2`, must be less than or equal to 512 words.
0,0	Formal place holders that must be included whenever `keywd` is specified.
keywd	The locked LU's keyword number.

DEXEC 3 (Remote I/O Control)

DEXEC request code 3 performs remote I/O control operations such as backspace, write end-of-file, and rewind.

```
   DEXEC(dest,code,cnwd][,p1[,p2][,p3][,p4][,0,0,keywd])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* equals the local node address.
code	*16-bit integer*. I/O control request code = 3.
cnwd	Control word. (See the *RTE-A Programmer's Reference Manual* for an explanation of this parameter.)
p1,...,p4 (input/output)	Required for list output line spacing and various other functions. (See the *RTE-A Programmer's Reference Manual* for details of the contents of these parameters.)
0,0	Formal place holders that must be included whenever *keywd* is specified.
keywd	The locked LU's keyword number.

Note DEXEC 3 calls to HP-IB devices that do double buffering are not supported. Use PTOP to implement I/O control operations to HP-IB devices.

DEXEC 6 (Remote Program Termination)

Using this DEXEC call your program can inform an RTE operating system that it wants to terminate execution of itself or another program. DEXEC 6 works only on programs that were originally started by a DEXEC 9, 10, 23, or 24 call.

```
    DEXEC(dest,code,name,numb)
```

If the program is to terminate itself (local operation only), optional parameters can be used as follows:

```
    DEXEC(dest,code,name,numb[,op1][,op2][,op3][,op4][,op5])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* equals the local node address.
code	*16-bit integer*. Remote program termination code = 6.
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. A 3-word array containing the program name to be terminated in the first five bytes (the sixth byte is not significant). *name* can be the value zero which causes a program to terminate itself (local operations only). Contrary to usage in an EXEC call, this parameter is not optional.
numb	*16-bit integer*. Completion type code. Contrary to usage in an EXEC call, this parameter is not optional. The completion type codes follow. Note that only 0 and 1 can be executed remotely.

-1	Serial reusability completion. When rescheduled, the program is not reloaded if it remained resident in memory. If this call is a "father" program's request, it is the same as *numb* = 0.
0	Normal completion.
1	Place program in dormant state; save current suspension point and resources.
2	Terminate and remove named program from the time list. If program is in I/O suspend state, system waits until I/O completes before terminating program. However, this call does not wait. The program's disk tracks are not released.
3	Terminate named program immediately, remove it from the time list, and release program's disk tracks. If program is in I/O suspend state, a system-generated clear request is issued to the driver. An abort message is printed at the system console.

op1,...,*op5* (input/output)	Up to five parameters that can be passed to the caller the next time the program executes. Works when *name* = 0 only.

DEXEC 9, 10, 23, 24

A call to DEXEC 9, 10, 23, or 24 schedules a dormant program for execution.

```
DEXEC(dest,code,name[,prm1][,prm2][,prm3][,prm4][,prm5][,bufr,bufl])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* equals the local node address.
code	*16-bit integer*. Schedule request code:
	9 Immediate schedule, wait
	10 Immediate schedule, no wait
	23 Queue schedule, wait
	24 Queue schedule, no wait
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. A 3-word array containing the program name to be scheduled in the first five bytes (the sixth byte is not significant). This parameter must reference an ASCII-coded program name. If a value of zero is referenced for a remote schedule call or if the program named does not exist, an error results.
prm1,...,prm5 (input/output)	Up to five optional one-word parameters may be passed to the named program.
bufr	Data string that can be passed to the scheduled program.
bufl	*16-bit integer*. Buffer length (+ for words, − for characters). The maximum length of this string buffer is 512 words (−1024 characters). If you specify *bufr* you must also specify *bufl*.

Note Programs scheduled with DEXEC 9, 10, 23, or 24 calls at RTE-A systems must be loaded as system utilities.

DEXEC 11 (Remote Time Request)

You can use this DEXEC (request code 11) to obtain the current time from the real-time clock at a specific node within your network.

```
   DEXEC(dest,code,time[,year])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* is equal to the local node address.
code	*16-bit integer*. Remote time request code = 11.
time	*Array of 16-bit integers*. 5-word time value array where *time(1)* is tens of milliseconds, *time(2)* is seconds, *time(3)* is minutes, *time(4)* is hours, and *time(5)* is the day of the year.
year	*16-bit integer*. Year value (1 word).

DEXEC 12 (Remote Timed Program Schedule)

This DEXEC call (request code 12) schedules a program for execution at specific time intervals. Execution may be scheduled either at an absolute start time or following a specified initial offset value.

Programs scheduled on RTE-A systems with a DEXEC 12 call must be loaded as system utilities.

Schedule execution at an absolute start time with the following syntax:

```
   DEXEC(dest,code,name,resl,mtple,hrs,mins,secs,msecs)
```

Schedule execution following a specified initial offset value with the following syntax:

```
   DEXEC(dest,code,name,resl,mtple,ofst)
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* is equal to the local node address.
code	*16-bit integer*. Remote timed schedule request code = 12.
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. A 3-word array containing the program name to be added to the time list in the first five bytes (the sixth byte is not significant). This parameter must reference an ASCII-coded program name. If a value of zero is referenced for a remote schedule call, if the program named does not exist, or if the program named is not dormant, an error will result.
resl	*16-bit integer*. Resolution code. 1 = tens of milliseconds; 2 = seconds; 3 = minutes; 4 = hours.
mtple	*16-bit integer*. Execution multiple.
hrs,...,msecs	*16-bit integer*s. Absolute start time in hours, minutes, seconds, and tens of milliseconds on a 24-hour clock.
ofst	*16-bit integer*. A negative value that declares the initial execution offset time based upon the content of the *resl* (resolution code) parameter.

DEXEC 13 (Remote I/O Status)

You can use this DEXEC call (request code 13) to obtain the status and type of a device identified by a logical unit number.

```
   DEXEC(dest,code,cnwd,sta1[,sta2][,sta3][,sta4])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* is equal to the local node address.
code	*16-bit integer*. Remote I/O status request code = 13.
cnwd	Control word containing the LU of the I/O device.
sta1	First status word. Device Table word 6 is returned here.
sta2	Second status word. Interface Table word 6 is returned here.
sta3	Third status word. If the Z-bit in the control word is equal to 0, driver parameter word 1 is returned here. If the Z-bit is equal to 1, the buffer address for return of driver information is specified here.
sta4	Fourth status word. If the Z-bit in the control word is equal to 0, driver parameter word 2 is returned here. If the Z-bit is equal to 1, the buffer length for return of driver information is specified here. When the Z-bit is equal to 1, the *sta4* length specification must be less than or equal to 512 words.

DEXEC 25 (Remote Partition Status)

Obtains the current status of a specific partition at an RTE-6/VM node. This call is not supported to RTE-A systems.

```
   DEXEC(dest,code,part,page,pnum,stat)
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. Must be a DS/1000-IV node with an RTE-6/VM operating system. Cannot be −1 or the local node number to indicate the local node.
code	*16-bit integer*. Remote partition status request code = 25.
part	*16-bit integer*. A decimal value that declares the partition number whose status is desired.
page	*16-bit integer*. The starting page number (plus 1) of the partition is returned here. This value represents the ordinal page number (i.e., first, second, etc.). A zero is returned if *part* contains an invalid partition number.
pnum	*16-bit integer*. The size of the user area available in the partition (in pages, minus 1 for the base page) is returned here. A value of −1 is returned if *part* contains an invalid partition number.
stat	*16-bit integer*. The status of the partition is returned here. Upon return, the parameter *stat* has the following status format: bit 15 will equal zero if the partition is reserved for programs requesting its; 1 if the partition is not reserved; bit 14 will equal zero if it is a real-time partition, 1 if it is a background partition.

DEXEC 99 (Remote Program Status)

A call to DEXEC with request code 99 results in the return of status information for a specific program. The program status codes are defined in the appropriate operating system reference manual. Programs controlled by this call on RTE-A systems must be loaded as system utilities and have an ID segment.

```
   DEXEC(dest,code,name[,stat])
```

dest	*16-bit integer*. The Router/1000 node address of the node where the call is executed. A value of −1 indicates the local node. Local execution will also be performed whenever *dest* equals the local node address.
code	*16-bit integer*. Remote program status code = 99.
name	*Integer array (FORTRAN); Packed character array (PASCAL)*. A 3-word array containing the program name for which status is requested in the first five bytes (the sixth byte is not significant). This parameter must reference an ASCII-coded program name. If a value of zero is referenced for a remote program status call, an error will result.
stat	Return parameter for program status. Status is returned in both this parameter and the A-register. The status word returned has the following format: bit 15 is zero if the program named is not a segment, 1 if it is a segment; bit 14 is zero if the program named is not in the time list, 1 if it is dormant but in the time list; bits 3 through 0 contain the actual program status code. The status word will contain −1 if the program named does not exist. The status codes returned in bits 3 through 0 are listed in the following table.

The RTE-A expanded status code are mapped to the RTE-6/VM equivalent in Table 11-1.

Table 11-1. Status Codes

Original RTE-A Status	Mapped RTE-6/VM Returned Equivalent
0	0
1	ERROR
2	2
3	3
4	ERROR
5	ERROR
6	6
7	6
47	0
50	3
51	3
52	3
53	3
54	3
55	3
56	2
57	1
60	1
61	4

12

Program-to-Program Communication (PTOP)

Table of Contents

PTOP Common Parameters

Several parameters are common to more than one PTOP call. These common parameters are described in detail in the following paragraphs. Parameters that are unique to a specific call are described within the parameter description area of that call.

pcb (input/output)	PTOP control block. A 4-word array of any type. The contents of the *pcb* parameter serve as a control block for the data link. The *pcb* array resides within the calling program and must not be modified.
err (output)	Error code. Produces an error code if an error condition is encountered during execution of a PTOP call. Must be defined as a 16-bit integer. If a slave program responds with a `REJCT` during execution, a "1" is returned in *err*. If the slave program responds with an `ACEPT`, the value "0" will be returned. Any other value returned in this parameter represents an error in the handling of the message.
tag (input/output)	Tag field. A 20-word array of any type. You can define information to be exchanged between the master and slave programs within the tag field. This parameter is passed from a master program via a `POPEN`, `PREAD`, `PWRIT`, or `PCONT` call. A slave program obtains the *tag* via a call to `GET`. A slave program may then pass *tag* data back to the `GET`. A slave program may then pass *tag* data back to the master program via an `ACEPT` or `REJCT` call.

Master PTOP Calls

POPEN

A `POPEN` call directed to a remote node causes the named slave PTOP program at that node to be scheduled.

HP 1000 to HP 1000 Calling Sequence

For PTOP communications between HP 1000 nodes, you must call `POPEN` within a master program to initiate the communication link with a slave program.

```
   POPEN(pcb,err,name,node,tag[,clon])
```

pcb	PTOP control block.
err	*16-bit integer*. Error return.
name	Slave program name. An array of up to 14 words containing the ASCII-coded slave program name and terminated by a blank or non-ASCII character.
node	*16-bit integer*. The Router/1000 node address of the node where the slave program resides and where it is to be scheduled for execution. A value of -1 indicates the local node.
tag (input/output)	Tag field. A 20-word array.
clon	Slave cloning parameter. Ignored when the slave is running in an RTE-A system. (Can be used to clone a slave program if the slave resides on an RTE-6/VM node with Session Monitor. Refer to the *DS/1000-IV User's Manual* for more information on the use of this parameter.)

HP 1000 to HP 3000 Calling Sequence

A POPEN call directed to a remote HP 3000 node causes the named slave PTOP program at that node to be scheduled.

For PTOP communications between HP 1000 and HP 3000 nodes you must call the subroutine `HELLO` to open communication before issuing the POPEN call. `BYE` must be called after the last PTOP call to terminate communication.

```
POPEN(pcb,err,name,node,tag[,enam][,pram][,flag][,bfsz])
```

pcb	PTOP control block.
err	*16-bit integer*. Error return.
name	Slave program name. An array of up to 28 bytes containing the ASCII-coded slave program name and terminated by a blank or non-ASCII character.
node	*16-bit integer*. The negative value of the logical unit (`LU`) number associated with the link to the HP 3000. For BISYNC connections, the LU number of the BISYNC link. For X.25 connections, the LU number of the virtual circuit assigned to the connection by DSN X.25/1000. Before using the POPEN call, use the subroutine `LU3K` to obtain this LU number of the virtual circuit.
tag (input/output)	Tag field. 20-word array.
enam	HP 3000 program ASCII entry point name. Refer to the *DS/3000 HP 3000 to HP 1000 Reference Manual for HP 3000 Users* for more information.
pram	HP 3000 program control information. Refer to the *DS/3000 HP 3000 to HP 1000 Reference Manual for HP 3000 Users* for more information.
flag	HP 3000 program loading options. The *flag* parameter passes loading option information to the scheduled slave program at the HP 3000 node. The loading options are declared by the value of bits within the *flag* parameter word. These bit settings and their defaults are described in the *DS/3000 HP 3000 to HP 1000 Reference Manual for HP 3000 Users*. If the *flag* parameter is omitted, the default values are used.
bfsz	*16-bit integer*. HP 3000 program communications buffer size. Refer to the *DS/3000 HP 3000 to HP 1000 Reference Manual for HP 3000 Users* for more information.

PREAD

Called from a master program to read data from a slave program.

```
   PREAD(pcb,err,buf,len,tag)
```

pcb (input/output)	PTOP control block.
err	*16-bit integer*. Error return.
buf	Data buffer. An array of a size equal to or greater than the value of the *len* parameter.
len	*16-bit integer*. Data length in words. A positive decimal value that declares the length of data available for the PREAD request. This value is passed to the *len* parameter in the slave's GET call as an indication of the maximum amount of data to send to the master. A maximum of 4096 words may be transferred between an HP 1000 and another HP 1000, or between an HP 1000 master program and an HP 3000 slave program. The maximum number of words transferred between an HP 3000 master program and an HP 1000 slave program depends on which size buffer module was appended to the request and reply converts (RQCNV and RPCNV). Maximum is 4096 words. The HP 3000 to HP 1000 buffer size defaults to the buffer size specified when the HP 3000 to HP 1000 link was configured. This default is overridden by supplying the optional buffer size parameter in the POPEN call at the HP 3000. Between the 1000 and the 3000, a negative length may be used to indicate the length is in bytes. When an odd number of bytes are to be transferred, an extra byte is appended to the end of the data.
tag (input/output)	Tag field.

PWRIT

Transfers data from a master program to a slave program.

`PWRIT(pcb,err,buf,len,tag)`

pcb (input/output)	PTOP control block.
err	*16-bit integer*. Error return.
buf	Data buffer. An array of a size equal to or greater than the value of the *len* parameter.
len	*16-bit integer*. Data length in words. A positive decimal value that declares the actual length of data being transferred to the slave for the PWRIT request. This value is passed to the *il* parameter in the slave's GET call as an indication of how much data to expect from the master. A maximum of 4096 words may be transferred between an HP 1000 and another HP 1000, or between an HP 1000 master program and an HP 3000 slave program. The maximum number of words transferred between an HP 3000 master program and an HP 1000 slave program depends on which size buffer module was appended to the request and reply converts (RQCNV and RPCNV). Maximum is 4096 words. Between the 1000 and the 3000, a negative length may be used to indicate the length is in bytes. When an odd number of bytes are to be transferred, an extra byte is appended to the end of the data.
tag (input/output)	Tag field.

PCONT

Provides for the exchange of a tag field between a master and a slave program.

`PCONT(pcb,err,tag)`

pcb (input/output)	PTOP control block.
err	*16-bit integer*. Error return.
tag (input/output)	Tag field.

PCLOS

Called from a master program to terminate a slave program. If the slave resides on an HP 1000 node, logical communication is also terminated.

`PCLOS(pcb,err)`

pcb (input/output)	PTOP control block.
err	*16-bit integer*. Error return.

PNRPY

Called from a master program to eliminate the need for the master to wait for a reply from the slave (an `ACEPT` or `REJCT`) before continuing processing. This call is honored only for `PWRIT`, `PCONT`, and `PCLOS`. `POPEN` and `PREAD` calls will be forced to wait for the reply.

PTOP communication with no reply is available only between HP 1000s. No Message Accounting is provided on PTOP calls with no reply.

```
   PNRPY([mode][,tto])
```

mode	*16-bit integer*. Defines the scope of the call. The options are: `0` No-reply only applies to the next PTOP call (default). `<0` No-reply applies to all PTOP calls that follow, until explicitly turned off. `>0` Turns off the no-reply option.
tto	*16-bit integer*. Transaction Timeout override value. If specified, it must be positive and less than 256. If not specified, the transaction timeout value which is used for all other requests for that node is used. Negative values or values greater than 256 are truncated to the value of the lower eight bits of the stated value. *tto* is specified in five second units. For example, *tto* = 2 is the same as 10 seconds.

Slave PTOP Calls

GET

Called from a slave program to obtain the next outstanding master request.

```
   GET(clas,err,func,tag,len[,bufr,bufr])
```

clas	Slave PTOP class number. A value for *clas* is assigned to the slave program as the first parameter in the runstring when the slave program is scheduled. Every slave program should first issue a call to `RMPAR` to obtain the value of *clas*. Only one value for *clas* is created by the initial `POPEN` call. All further `POPEN` calls to the same slave are assigned the same *clas* value. The value for *clas* must be saved and referenced in each `GET` call. Be careful that you do not alter the content of *clas*.
err	*16-bit integer*. Error return.

func	*16-bit integer*. A value is returned to this parameter upon completion of a call to `GET`. This value indicates the type of master program request obtained via `GET`. Request types are:
	`1 POPEN`
	`2 PREAD`
	`3 PWRIT`
	`4 PCONT`

tag	Tag field received from master program.
len	*16-bit integer*. A value is returned here at the completion of the GET call when *func* = 2 or 3 (PREAD or PWRIT). *len* indicates the actual number of words to be transferred if the master request is accepted. (See the *bufr* parameter.)
bufr	Data buffer. An array greater than the value of the *bufz* parameter. This parameter is only required when the master request is a PWRIT. Data from the master is transferred to this buffer (see the following table.) For POPEN, PREAD, and PCONT requests, the buffer contents are unmodified. On PWRIT requests, the data is copied to *bufr* up to the limit specified in *bufz*. *len* contains the actual size of the data buffer as sent by the master. Among its other error checking, the slave should check to see if the master sent too much data by comparing the values in *len* and *bufz*.
bufz	*16-bit integer*. Defines buffer size of *bufr*.

The following tables show how buffers are affected when a GET call is issued after either a PWRIT or a PREAD call. In the case of a PWRIT call, either the *bufr* or *buf* parameter may be used. If *bufr* (GET call) is used, *buf* (ACEPT call) should not be used. For a PREAD call, only *bufr* (ACEPT call) should be used.

Table 12-1. PWRIT Master Call

len (GET call)	*bufr* (GET call)	*buf* (ACEPT call)	*bufz* (GET call)
Actual number of words transferred here.	Data from slave is transferred to this buffer.	Data from master may be optionally transferred to this buffer.	Size of *bufr* in GET call. If *len* > *bufz*, an error will occur.

Table 12-2. PREAD Master Call

len (GET call)	*bufr* (GET call)	*buf* (ACEPT call)	*bufz* (GET call)
Size of master buffer specified in master.	Not used.	*len* words from this buffer are transmitted to master.	Not used.

ACEPT

Called from a slave program to accept and complete a master request obtained via a GET call.

```
ACEPT(tag,err[,buf])
```

tag	Tag field to be returned to master program. In a PREAD request, the actual length of the buffer being transferred should be passed in this parameter.
err	*16-bit integer*. Error return.
buf (input/output)	Data buffer containing the data to be transferred in a PREAD request and optionally in a PWRIT request.

REJCT

Called from a slave program to reject a master request obtained via a `GET` call.

```
   REJCT(tag,err)
```

tag	Tag field.
err	*16-bit integer*. Error return.

FINIS

Called from a slave program to terminate communication with a master program. Cannot be used if the master program resides on an HP 3000.

```
   FINIS
```

Error Codes

The following error codes are returned to the *err* parameter of PTOP master or PTOP slave calls.

Table 12-3. PTOP Error Codes

Message	Meaning
−40	Not enough parameters.
−41	Remote program not defined.
−42	No remote system room to initiate communication.
−44	Remote program not open correctly (PCB was destroyed).
−45	A `PWRIT`, `PREAD`, or `PCONT` call has been issued to a slave program which is dormant.
−46	Sequence error.
−47	May occur in PTOP slave subroutine calls when a communications line error, system table validity check failure or request timeout error occurs.
−48	Abortive error: indicates something seriously wrong.
−49	A `PCLOS` terminated a shared slave program while one or more requests were pending from other master programs.
−50	Local node not initialized or, for requests made to itself, local node is quiescent (same as `DS00`).
−51	Communications line parity or other line error.
−52	Communications line timeout error.
−53	Illegal record size.
−54	Illegal nodal address.
−55	Request timeout.
−56	Illegal request.
−57	System table error.
−58	Remote busy.
−59	Illegal or missing parameters.
−103	Illegal PCB (PTOP)

13

Remote I/O Mapping

Table of Contents

Remote I/O Mapping *maps* or redirects I/O requests destined for an LU on an HP 1000 node to an LU at a remote HP 1000 node. The node at which the mapped LU (or *source LU*) resides is known as the *source node*; I/O requests to the source LU are redirected to the *destination LU* at the *destination node*.

Using IOMAP

To set up a map to an LU at a destination node, you execute the program IOMAP at the source node. *You must initialize NS-ARPA/1000 between the source and destination node before running IOMAP.*

IOMAP can perform seven functions. These functions are summarized in Table 13-1, along with the runstrings.

Table 13-1. Remote I/O Mapping Runstrings

Function	Runstring
Initialize I/O Mapping	`[RU,]IOMAP,`*`mappableLU`*`,-1`
Establish a map using the first unused mappable LU	`[RU,]IOMAP,-1,`*`destLU`*`,`*`destNode`*`,`*`securityCode`*
Establish/change a map	`[RU,]IOMAP,`*`sourceLU`*`,`*`destLU`*`,`*`destNode`*`,`*`securityCode`*
Disable map	`[RU,]IOMAP,`*`sourceLU`*`,0, ,`*`securityCode`*
Obtain information on specific map	`[RU,]IOMAP,`*`sourceLU`*`,-1`
Obtain error values	`[RU,]IOMAP,-2`
Obtain mapping information	`[RU,]IOMAP,`*`sourceLU`*`,-2`

Caution There is no protection against re-assigning a map that is in use. You should establish a map by allowing IOMAP to select the first unused mappable LU. You should also clear maps after you have used them.

Initialize I/O Mapping

```
[RU,]IOMAP,mappableLU,-1
```

Parameters

mappableLU	Mappable LU (system LU associated with a mappable DVT).

Return Parameters

First Parameter	IOMAP returns the specified *mappableLU* if it is mappable, otherwise, it returns the first mappable LU.

Establish a Map Using First Unused Mappable LU

```
[RU,]IOMAP,-1,destinationLU,destinationNode,securityCode
```

Parameters

destinationLU

Destination system LU of map.

Options:

Set bit 15 to enable the header flag. Remote I/O Mapping will write a message of the following form to the destination LU preceding each record sent to that LU:

```
MESSAGE FROM NODE # nnnnn PRGM ppppp AT DAY dddd, hh :mm :ss
```

Where: *nnnnn* is the source node (node where the message originated); *ppppp* is the name of the program sending the message (or `SYS I/O` if the I/O request is from the system or class buffered); and *ddddd hh mm ss* is the day and time that the request was made at the source node.

Set bit 14 to enable the prompt flag. Remote I/O Mapping will write a prompt of the following form to the destination LU preceding each read request mapped from the source LU:

```
(nnnnn)
```

Where: *nnnnn* is the source node.

Set bit 13 with bit 14 to change the read request timeout to 20 minutes but suppress the prompt flag.

destinationNode

Destination node number of the map (the node of the destination LU).

securityCode

Network Management security code.

Return Parameters

First Parameter	The source LU if the map was successfully established, otherwise one of the following negative error codes: • −1 (177777B) if `securityCode` was incorrect • −2 (177776B) if `sourceLU` is not a mappable LU • −3 (177775B) if Remote I/O Mapping is not set up correctly or if Remote I/O Mapping has not been initialized • −4 (177774B) if `destinationLU` is mappable • −5 (177773B) if `destinationNode` is not known to the local node • −6 (177772B) if `destinationLU` is invalid or if attempts to obtain information about the destination LU failed • −7 (177771B) if LUMAP is not present and active • −8 (177770B) if LUQUE is not present and active
Second Parameter	`destinationLU`
Third Parameter	`destinationNode`
Fourth Parameter	If IOMAP did not establish the map, it returns the specified `securityCode`. If IOMAP successfully established the map, it returns one of the following values: • 0 (00000B) normal completion • −1 (177777B) if `destinationLU` has either an associated infinite device timeout or a timeout larger than the user requested master timeout • −2 (177776B) if `destinationLU` is associated with the bit bucket • −3 (177775B) if `destinationLU` is not a unit-record device

Establish or Change a Map

```
[RU,]IOMAP,sourceLU,destinationLU,destinationNode,securityCode
```

Parameters

sourceLU	Mappable LU on source system.

destinationLU	Destination system LU of map. The *destinationLU* must be associated with a unit-record device (such as a terminal, printer, or mag tape unit).
	
	Options:
	
	Set bit 15 to enable the header flag. Remote I/O Mapping will write a message of the following form to the destination LU preceding each record sent to that LU:
	
	`MESSAGE FROM NODE # nnnnn PRGM ppppp AT DAY dddd, hh :mm :ss`
	
	Where: *nnnnn* is the source node (node where the message originated); *ppppp* is the name of the program sending the message (or `SYS I/O` if the I/O request is from the system or class buffered); and *ddddd hh mm ss* is the day and time that the request was made at the source node.
	
	Set bit 14 to enable the prompt flag. Remote I/O Mapping will write a prompt of the following form to the destination LU preceding each read request mapped from the source LU:
	
	`(nnnnn)`
	
	Where: *nnnnn* is the source node.
	
	Set bit 13 with bit 14 to change the read request timeout to 20 minutes but suppress the prompt flag.

destinationNode	Destination node number of the map (the node of the destination LU).
securityCode	Network Management security code.

Return Parameters

First Parameter — *sourceLU* if map was successfully established, otherwise one of the following negative error codes:

- −1 (177777B) if *securityCode* was incorrect
- −2 (177776B) if *sourceLU* is not a mappable LU
- −3 (177775B) if Remote I/O Mapping is not set up correctly
- −4 (177774B) if *destinationLU* is mappable
- −5 (177773B) if *destinationNode* is not known to the local node

	• −6 (177772B) if *destinationLU* is invalid or if attempts to obtain information about the destination LU failed • −7 (177771B) if LUMAP is not present and active • −8 (177770B) if LUQUE is not present and active
Second Parameter	*destinationLU*
Third Parameter	*destinationNode*
Fourth Parameter	If IOMAP did not establish the map, it returns the specified *securityCode*. If IOMAP successfully established the map, it returns one of the following values: • 0 (000000B) normal completion • −1 (177777B) if *destinationLU* has either an associated infinite device timeout or a timeout larger than the user requested master timeout • −2 (177776B) if *destinationLU* is associated with the bit bucket • −3 (177775B) if *destinationLU* is not a unit-record device

Disable Map

```
[RU,]IOMAP,sourceLU,0, ,securityCode
```

Parameters

sourceLU	System LU of map to be disabled.
securityCode	Network Management security code.

Return Parameters

First Parameter	*sourceLU* if map was successfully disabled, otherwise one of the following negative error codes: • −1 (177777B) if *securityCode* was incorrect • −2 (177776B) if *sourceLU* is not a mappable LU • −3 (177775B) if Remote I/O Mapping is not set up correctly • −7 (177771B) if LUMAP is not present and active • −8 (177770B) if LUQUE is not present and active
Second Parameter	0 (the destination LU is the bit bucket)
Third Parameter	0
Fourth Parameter	specified *securityCode*

Return LU Mapping Information

```
    [RU,]IOMAP,sourceLU,-1
```

Parameters

sourceLU	The mappable LU for which you want IOMAP to display information. If you specify a non-mappable LU, IOMAP will display map information for the first mappable LU greater than *sourceLU*.

Return Parameters

First Parameter	The *sourceLU*. If *sourceLU* is not mappable, this will be the first mappable LU greater than *sourceLU*. Otherwise, it is one of the following negative error codes: • −2 (177776B) if the *sourceLU* is not a "mappable" LU and there are no mappable LUs greater than *sourceLU* • −3 (177775B) if Remote I/O Mapping is not set up correctly
Second Parameter	destination LU of map for the LU returned in the first parameter.
Third Parameter	destination node number of map for the LU returned in the first parameter.
Fourth Parameter	The type of remote operating system (`$OPSYS` value, as documented in the RTE-A manual set).

Return Mapping Information for a Specific LU

```
   [RU,]IOMAP,sourceLU,-2
```

Parameters

sourceLU	The mappable LU for which you want IOMAP to display information.

Return Parameters

First Parameter	*sourceLU*, or the following negative error code: • −3 if I/O mapping is not initialized correctly.
Second Parameter	destination node, or the following negative error code: • −1 (177777B) if *sourceLU* is not mappable
Third Parameter	destination LU

If the first return parameter is not a mappable LU, IOMAP returns one of the following negative error codes:

- −1 if LU is an invalid LU number
- −2 if LU is associated with the bit bucket
- negative of the LU's timeout if not −1 or −2

Return Error Values

```
[RU,]IOMAP,−2
```

Return Parameters

First Parameter	first word (two characters) of the ASCII four-character error code (for example, “DS” of DS04). For serious errors, this is a numeric error code, and the second and third parameters (below) are meaningless.
Second Parameter	second word (two characters) of the ASCII four-character error code.
Third Parameter	number of node reporting the error.

Remote Interactive Sessions

Remote I/O Mapping allows you to gain access to the remote command interpreters (CI or FMGR). This “virtual terminal” capability allows you to establish a session at a remote node and interact with the command interpreter as if the terminal were connected directly to the remote node.

To set up a remote interactive session, you must perform the following tasks:

- Run IOMAP from REMAT to establish an LU map at the remote node.
- Run SYSAT to get the remote system’s attention and logon prompt.
- Logon to the remote node.

These steps are described in the following subsections.

Establishing the Map

Use REMAT to switch to the node at which you want to establish an interactive session. Then, run IOMAP at the remote node, mapping an LU at that node to your terminal’s LU at your node.

Get the Remote System’s Attention

The next step is to get the remote system’s attention. You can use SYSAT to do this, or to set the break bit of a program on a remote node. In gaining system attention on a particular LU, SYSAT has the effect of someone striking a key on a terminal to get a logon (or break mode) prompt.

```
RU,SYSAT,attentionLU,remoteNode
```

SYSAT causes the remote node to give its breakmode prompt or the logon prompt on the *attentionLU*. The *attentionLU* on the remote node must be a mappable LU. At least one map must have been set up at the specified node for this request to work.

To set the break flag of a program in a remote node, enter:

```
RU,SYSAT,programName,remoteNode
```

Logon

If you are running under a command interpreter at the local node, then you will want to keep this command interpreter "out of the way" while interacting with the remote system. One way to keep your local command interpreter from interfering with your remote session is to suspend (`SS`) it. To restart CI, enter the system command `GO`. Refer to *RTE-A User's Guide* (92077-90002) for more information on these commands.

You must enter the password, if any, along with the account name (on the same line) in response to the logon prompt.

Obtaining a CM or Breakmode Prompt

While logged on to a remote node, there may be times where you want the CM's attention (or breakmode prompt). Typically, the CM (or breakmode) prompt you obtain will be that of the local system. To get the remote node's CM, obtain the local node's CM prompt and run SYSAT, specifying the LU mapped to this node as the attention LU (or, if running a program, you can also specify the program name). SYSAT will cause the remote node's CM (or breakmode) prompt to appear.

Error Messages

The following are "serious" Remote I/O Mapping error codes. When these errors occur, they are returned to IOMAP in the first return parameter (the second and third parameters will be meaningless). These error codes are obtained when IOMAP is run with a -2 parameter.

Table 13-2. Remote I/O Mapping Error Codes

Message	Meaning
0	No error.
4	NS-ARPA quiescent resource number is corrupt.
5	Source node is not completely initialized.
6	Class number set up for LUMAP has been corrupted.
8	Program LUMAP is not present in system.
9	Destination node number is not in the NRV.
10	NS-ARPA not initialized.
11	Error on class number allocation.
12	No class numbers available.

14

Maintenance Utilities

Table of Contents

NSINF

NSINF is the NS-ARPA/1000 information utility.

NSINF Scheduling

```
[RU,]NSINF
```

NSINF Operation

NSINF is an interactive program. When you run NSINF, it will prompt you for a command as follows:

```
NSInf>
```

Enter `?` and NSINF will print its main menu with a list of valid commands. Table 14-1 is a summary of the NSINF commands.

To exit NSINF, enter `E` in response to the `NSInf>` prompt.

If NSINF has more than a screenful of information to print, it will print one screen and then ask you if you want to print more information as follows:

```
--- More ---
```

The following replies are allowed:

- Enter RETURN, a space, or a plus (+) to display the next screen of information.
- Enter `A` or `Q` to abort the display.
- Enter `Z` to suspend NSINF. Entering any character will then invoke the RTE system prompt `CM>`. Enter `GO` at the RTE system prompt to continue the display.
- All other characters will have no affect on continuing the display.

Commands

Table 14-1 is a summary of the NSINF commands.

Table 14-1. NSINF Commands

Command	Description
`?`	Prints the NSINIF main menu.
`A`	Prints the local node's name, addresses, and Gateway Table (GT) as configured via NSINIT. Used to check the gateway or routing configuration.
`B`	Prints Buffer and Memory Manager statistics.
`C`	Prints the configured resources for the NS and ARPA Services. Used to check if there are enough resources available.
`I`	Prints transmit LUs of NS-ARPA/1000 link interface cards generated in the system. Used to verify the card configuration.
`L`	Reads the NS-ARPA link interface card statistics and IFT extent word information. NSINF will prompt you for a link interface card LU. Used to verify the card configuration.
`M`	Prints Message Accounting (MA) information for each node in the NRV with MA enabled.
`N`	Prints the Nodal Routing Vector (NRV).
`P`	Prints socket record and protocol path record information for the sockets owned by a program. NSINF will prompt you for a program name. Used to troubleshoot a user program.
`R`	Prints information about the local node's Router/1000 links.
`S`	Prints information about the open VC and CALL sockets.
`T`	Prints the number entries in the Master Transaction Control Block (TCB) list, Slave TCB list, and Process Number Lists (PNL).
`U`	Prints information about the NS-ARPA message tracing utility (NSTRC) and the NS-ARPA event logging monitor (EVMON). Used to check the status of tracing and logging.
`V`	Prints information about NS-ARPA/1000 resource numbers, information about DS/1000-IV Compatible Services system resources and configuration values, and LU table and links used for DS/1000-IV Compatible Services (RTE-MPE).
`W`	Prints information about TELNET virtual terminal sessions, both initiated at and remotely connected to the local node. Use to troubleshoot TELNET.
Display Commands	When more than one screenful of information needs to be displayed, NSINF prompts you to continue or to end the display. Enter one of the following commands. `+` or RETURN or space—displays the next screen of information. `A` or `Q`—aborts the display. `Z`—suspends NSINF display.

NSINIT

NSINIT uses a dialogue to prompt the user for initialization information. You can specify an answer file that contains the responses to the dialogue or you can respond to the dialogue interactively.

Scheduling NSINIT

To initialize NS-ARPA, you can schedule NSINIT from the system WELCOME file.

```
    [RU,]NSINIT[,inputDevice][,outputDevice][,logDevice]
```

inputDevice	Device (file or interactive device LU) that provides input for NSINIT dialogue. The *inputDevice* can be an answer file previously created by running NSINIT. If *inputDevice* is an interactive device, NSINIT will also print prompts and error messages to the device. *Default*: local LU 1 (scheduling terminal).
outputDevice	Device (file name or device LU) to which NSINIT will write the dialogue. If this is a file, it can later be used as an answer file. If *outputDevice* and *inputDevice* are files, they cannot be the same file. If you enter the name of an existing file, NSINIT will ask you if you want to overwrite the file. If you specify a device, HP recommends that the device be spooled. If you enter LU 0, NSINIT does not create any output. *Default*: If *inputDevice* is interactive, NSINIT will prompt you for a file name. If *inputDevice* is not interactive, the default is LU 0 (no output). During an interactive NSINIT dialogue, NSINIT prompts you again for an output file name.
logDevice	Device (file name or device LU) to which NSINIT will log any errors. *Default*: local LU 1 (scheduling terminal).

NS-ARPA Message Tracing

There are three NS-ARPA message tracing utilities:

- NSTRC, which enables message tracing
- BRTRC, which terminates message tracing and allows you to format the trace file
- FMTRC, which formats the trace file and allows you to select trace records according to nodes, Link Interfaces, and sockets

Using Message Tracing

The procedure for using message tracing is as follows:

1. Enable message tracing.
2. Run the user program or generate NS-ARPA activity (for example, use NFT or FTP to copy a file). If you are debugging a user NetIPC program, use the NSINF `P` command to get the program's Global Socket Descriptor (GSDs).
3. When the program or activity terminates, halt tracing by running BRTRC.
4. Format and examine the trace file using FMTRC. If you are debugging a user NetIPC program, examine the socket level trace records for the program's GSDs. If you are debugging a program that used DS/1000-IV Compatible Services, format RTR messages.
5. If you suspect a link problem, run tracing at two nodes and examine the network trace records at both ends of the link for symmetry.

NSTRC Scheduling

```
                                                      ┌,N[ETWORK ]┐
[XQ,]NSTRC[,traceFile][,errorLog][,recordLength]│,S[OCKET]  │
                                                      └,B[OTH]    ┘
```

`traceFile`	File to which you want NSTRC to write the trace records. While NSTRC is executing, it writes the trace records to its VMA partition. NSTRC writes to the partition in a circular manner; if NSTRC has filled the partition, it overwrites the earliest records. NSTRC uses approximately one-fourth of the trace file for accounting. For each remaining page in the partition, NSTRC can write 10 trace records. Therefore, if NSTRC's VMA partition is *`n`* pages long, NSTRC can write approximately `( n * 3/4) * 10` trace records before it must overwrite previous records. If you enter the name of an existing file, NSTRC overwrites the existing file. If you use the default *`traceFile`* and it already exists, NSTRC prints an error message and terminates. *Default*: `NS_TRACE.TRC` in the current working directory.
`errorLog`	Device (file or device LU) to which you want NSTRC to report run-time errors. *Default*: local LU 1 (scheduling terminal).
`recordLength`	The trace record length, in bytes. For network level tracing, the trace record size is approximately the total size of all protocol headers. If a message has more bytes than *`recordLength`*, NSTRC truncates the message. *Range*: 48 to 120 bytes. *Default*: 60 bytes.

`NETWORK`	(*Default*) Trace messages at the network level. NSTRC records messages at the LI software level as they enter or leave the node. In addition, NSTRC records any Probe, Socket Registry, NFT, TCP, and IP control messages (control messages are messages sent by the protocol modules as part of the protocol and do not contain user data), and HDLC/Multidrop "Link Up" messages. You can use FMTRC to format the IP or LI header (OSI layer 3s−−IEEE 802.3/ETHERNET/LAN or Router/1000). FMTRC does not format higher-level protocol headers, such as TCP or PXP unless the `nice` format option is used. *Useful for analyzing network problems*.
`SOCKET`	Trace messages at the socket level. NSTRC records messages as they are sent and received on user sockets. (You can select messages according to Global Socket Descriptors when you format the trace file via FMTRC.) *Useful for debugging user applications*.
`BOTH`	Trace messages at both the network and socket level. *Default*: `NETWORK`.

BRTRC Scheduling

```
[RU,]BRTRC
```

Caution If you abort NSTRC instead of terminating it with BRTRC, FMTRC will not be able to format the trace file.

FMTRC Scheduling

```
[RU,]FMTRC[,inputDevice][,rawTraceFile][,formattedDevice]
          [,titleField][,logDevice]
```

inputDevice	Device (answer file name or interactive device LU) that provides input for FMTRC dialogue. *Default*: local LU 1 (scheduling terminal).
rawTraceFile	The raw trace file for FMTRC to format. This must be a disk file; if NSTRC posted trace messages to a tape, you must restore the tape's contents to a disk file before formatting it. NSTRC must not have the file open for tracing. *Default*: `NS_TRACE.TRC`.
formattedDevice	Device (file name or device LU) to which FMTRC outputs the formatted trace file. *Default*: If you specified a file for *rawTraceFile*, FMTRC takes that file name, strips any existing type extension, and adds the type extension FMT to form the name for the formatted (output) file. If you did not specify a *rawTraceFile*, the default is the file `NS_TRACE.FMT`.

titleField	An ASCII string of up to 72 characters to label the output file. FMTRC prints *titleField* in a heading at the beginning of the output file.
logDevice	Device (file name or device LU) to which FMTRC logs any errors. *Default*: local LU 1 (scheduling terminal) if you are running FMTRC interactively, `FMTRC.LOG` if you specified an answer file. If `FMTRC.LOG` already exists, FMTRC overwrites it.

NS-ARPA Event Logging

There are three NS-ARPA event logging utilities:

- EVMON, the event monitor. EVMON receives log records from NS-ARPA protocol handlers and services, and writes the records to a log file or device.
- BREVL, which terminates EVMON, allowing you to purge the log file.
- LOGCHG, which allows you to change the event classes for EVMON to log.

EVMON Scheduling

```
[XQ,]EVMON[,logDevice][,logMask]
```

logDevice	The device (file or device LU) to which EVMON will log the events. If *logDevice* is an existing file, EVMON will append to it. EVMON opens the log file with shared read access and writes logging records to a file until the LU that the file resides on is full, or until you halt EVMON (via BREVL), or until you shut down NS-ARPA/1000. *Default*: The file `/SYS/NS_EVENT.LOG`.

logMask	The octal representation of the bits set in the following mask. The *logMask* must be octal; as an option, you may append a `B` to the end of it.
	Log Mask:
	`7             0`
	`O R D E W M P L`
	The bits set select the event classes as described in Table 14-2.
	Default: the current value of *logMask*. NSINIT initially sets this value to 161 octal to log logging statistics, severe errors, disasters, and resource depletion.

Table 14-2. Log Mask Event Classes

Bit	Position	Logging Event Class
`L`	0	Logging Statistics (Class 0). *Must be set*. Event logging start and stop times.
`P`	1	Protocol-specific information (Class 1). *HP recommends that you do not select this class unless your HP representative specifically asks that you do so*. This logging event class generates a multitude of log file entries.
`M`	2	Event messages (Class 2). *HP recommends that you do not select this class unless your HP representative specifically asks that you do so*. This logging event class generates a multitude of log file entries.
`W`	3	Warnings (Class 3), which indicate abnormal events, but do not necessarily indicate subsystem problems. *HP recommends that you do not select this logging class*.
`E`	4	Severe errors (Class 4), which indicate that NS-ARPA/1000 is not performing as it should, but the subsystem was able to recover. *HP recommends that you select this logging class*. NSINIT selects this logging class automatically.
`D`	5	Disasters (Class 5), which indicate that the NS-ARPA/1000 software detected a severe and irrecoverable problem. *HP recommends that you select this logging class*. NSINIT selects this class automatically.
`R`	6	Resource limit (Class 6), which indicates that a user-configurable resource has been depleted. *HP recommends that you select this logging class*. The network manager can use this information to adjust the node's configuration. NSINIT selects this logging class automatically.
`-`	7	Reserved. *Must be zero*.

EVMON Output

EVMON writes log records to a file or device. EVMON output consists of an *EVMON header* and *log records*.

EVMON Header

Each time a user schedules EVMON, EVMON writes the following header to the log file or device:

```
        currentTime
                                   Event Log at nodeName
```

where *`currentTime`* is the system time at which EVMON was scheduled and *`nodeName`* is the name of the node at which EVMON is running and *`currentTime`* is the system time at which EVMON was scheduled. For example:

```
        Tue Apr 8 1986 10:13:50 pm
                                   Event Log at ANNE.LAB.HP1000
```

Log Records

The log file contains *log records*. Log records consist of two parts:

- the first part is the *log record header*;
- the second part is the *log message*.

The log record header and log message are illustrated below:

```
currentTime
  EventClass  Entity  Location   xx  yy    LogMask   ProcessName/Session
  LogMessage
```

The log record header may include the *currentTime*, which is the current system time. The *currentTime* is not printed if there is less than one second difference between it and the last *currentTime* printed.

BREVL Scheduling

```
[RU,]BREVL
```

LOGCHG Scheduling

```
                  [          logmask[B]     ]
                  [          *              ]
                  [  { + }   RESOURCELIM    ]
                  [  { - }   DISASTER       ]
[RU,]LOGCHG       [  { , }   ERROR          ]  . . .
                  [  { Δ }   WARNING        ]
                  [          EVENT          ]
                  [          PROLOG         ]
                  [          LOGSTATS       ]
```

+ – , Δ	LOGCHG evaluates the runstring parameters as an expression and sets the log mask (*logMask*) to the new value. The parameters are delimited by a plus, minus, comma, or a blank space. LOGCHG uses two operators when evaluating the expression: + and –. The + represents logical addition (bit inclusion); the – represents logical subtraction (bit exclusion). When two terms have no operator between them, the + operator is used. For example, `LOGCHG DISASTER LOGSTAT` is the same as `LOGCHG DISASTER+LOGSTAT`. When the expression begins with a + (or –), the value of the expression is added to (or subtracted from) the current log mask. For example, `LOGCHG +WARNING` is the same as `LOGCHG *+WARNING`.
logmask[B]	The octal representation of the bits set in the following mask. The *logMask* must be octal; as an option, you may append a `B` to it.

Log Mask:

```
 7             0
|O|R|D|E|W|M|P|L|
```

	The bits set select the event classes as described in Table 14-2.
`*`	The current value of *`logMask`*.
`RESOURCELIM`	The bit name for the resource limit event class (Class 6).
`DISASTER`	The bit name for the disasters event class (Class 5).
`ERROR`	The bit name for the severe errors event class (Class 4).
`WARNING`	The bit name for the warnings event class (Class 3).
`EVENT`	The bit name for the event messages event class (Class 2).
`PROLOG`	The bit name for the protocol-specific information event class (Class 1).
`LOGSTAT`	The bit name for the logging statistics event class (Class 0). You must always log this event class.

Nodal Registry List Utility (NRLIST)

NRLIST lists the contents of the Nodal Registry.

NRLIST Scheduling

```
[RU,]NRLIST
```

DS/1000-IV (RTE-MPE) Message Tracing Utilities

LOG3K and TRC3K allow you to trace and format DS/1000-IV Compatible Services (RTE-MPE) message records over Bisync and X.25 links.

LOG3K Scheduling

```
[RU,]LOG3K[,consoleLU]
```

`consoleLU`	Is the logical unit LOG3K uses for command input and output. *Default*: Local LU 1 (scheduling terminal).

LOG3K Operation

LOG3K prompts you for commands by printing the following message:

```
CHANGES?
```

Commands

LOG3K commands are summarized in Table 14-3. You must enter LOG3K commands in uppercase.

Table 14-3. LOG3K Commands

Command	Function
`??`	Print a description of LOG3K commands and options.

`EN`
`EX`
`/E`
`NO`
	Exit LOG3K.

`LU`	Enable tracing, disable tracing or specify LU of magnetic tape device to which trace records will be logged.
`TY`	Specify the type of tracing (header, appendage, and/or data).
`UP`	Restart tracing if an error is encountered while writing to the magnetic tape.

??

Print a description of LOG3K commands and options.

```
??
```

EN, EX, /E, NO

Exit LOG3K when issued in response to the `CHANGES?` prompt.

```
  ┌    ┐
  │ EN │
  │ EX │
  │ /E │
  │ NO │
  └    ┘
```

LU

Enable tracing, disable tracing, or specify LU of magnetic tape device to which trace records will be written.

```
    ┌              ┐
  LU│ ,magTapeLU   │
    │ ,0           │
    └              ┘
```

magTapeLU	The LU of the magnetic tape device. QUEX/D3KMS will write the DS/1000-IV Compatible Services (RTE-MPE) message records to the tape mounted at that device. (Cannot be a CS/80 tape device.) By specifying an LU, you enable tracing.
`0`	Disables tracing.

TY

Specify the type of information that you want to trace.

```
   ┌              ┐
   │AP            │
TY │DA:numWords   │
   │HE            │
   │NO            │
   └              ┘
```

`AP`	Trace message header and appendage.
`DA:`*`numWords`*	Trace message header, appendage, and *`numWords`* words of data. *`numWords`* must be an integer between 1 and 135.
`HE`	Trace message header.
`NO`	Trace nothing. QUEX/D3KMS will not write any records to the trace file, but the file will still be open and tracing will still be enabled.

UP

Restart tracing if an error has occurred while writing to the magnetic tape.

```
UP
```

TRC3K Scheduling

```
[RU,]TRC3K[,commandInput][,loggingInput][,outputDevice]
```

`commandInput`	The LU or FMGR file to provide TRC3K commands. *Default*: Local LU 1 (scheduling terminal).
`loggingInput`	The LU of the magnetic tape device with the tape where the traced records were logged. QUEX/D3KMS must not be writing to this LU while TRC3K is reading from it. If you enter a negative number, TRC3K will terminate. If not specified, TRC3K prompts you for a magnetic tape LU.
`outputDevice`	The LU or file to which TRC3K will print the formatted output. If *`outputDevice`* is a file, *it must be an existing FMGR file*. *Default*: *`commandInput`* if it is interactive, or local LU 1 (scheduling terminal).

TRC3K Operation

If you did not specify *`loggingInput`*, TRC3K prompts you for the LU of the magnetic tape device with the tape where the traced records were logged:

```
LOGGING INPUT:
```

Enter the magnetic tape device LU, as specified above.

Commands

TRC3K prompts you for commands by printing the following message:

```
/TRC3K:
```

TRC3K commands are summarized in Table 14-4.

Table 14-4. TRC3K Commands

Command	Function
`??`	Print a description of TRC3K commands and options.
`EXIT`	Exit TRC3K.
`FORMAT`	Specify the items (header, appendage, and/or data) to format.
`LIST`	Set the list (output) device or file.
`PRINT`	Print the formatted message records to the output device.
`SET`	Set the characteristics of the messages to be printed.

Note You must enter TRC3K commands in uppercase.

??

Print a description of the TRC3K commands.

```
??
```

EXIT

Exit TRC3K.

```
E[XIT]
```

FORMAT

Specify which portion or portions of the message record to print when you enter the `PRINT` command.

```
         ┌,H[EADER]     ┐
F[ORMAT] │,A[PPENDAGE]  │
         └,D[ATA]       ┘
```

`HEADER`	Print the message header only.
`APPENDAGE`	Print the header and an octal and ASCII representation of the appendage.
`DATA`	Print the header, and an octal and ASCII representation of the appendage and data.

LIST

Set the list (output) device or file.

```
   L[IST]=outputDevice
```

outputDevice	The LU or file to which TRC3K will print the formatted output. If *outputDevice* is a file, *it must be an existing FMGR file*. *Default*: the second parameter in the runstring (*outputDevice*); if you did not specify *outputDevice* in the runstring, the default is the scheduling terminal.

PRINT

Print the formatted records to the output device.

```
          ┌,A[LL]    ┐
   P[RINT]│,F[IRST]  │
          └,N[EXT]   ┘
```

ALL	Print all the message records that meet the characteristics specified by the SET command.
FIRST	Print the first message that meets the characteristics specified by the SET command.
NEXT	Print the next message that meets the characteristics specified by the SET command. (Default.)

SET

Set the characteristics of the messages to be printed by the PRINT command.

```
        ┌                    ┐
        │ C[LASS]=value      │
        │ E[NDREC]=value     │
S[ET]   │ R[TENO]=value      │  . . .
        │ STA[RTREC]=value   │
        │ STR[EAM]=value     │
        │ @                  │
        └                    ┘
```

CLASS	Select only the message records that belong to the class specified by *value*. For a list of message classes and the corresponding messages, refer to the *NS-ARPA/1000 Error Message and Recovery Manual*.
ENDREC	Select the records on the tape with record numbers less than or equal to the number specified by *value*. The record numbers are the tape's sequential record numbers. *Default*: the highest (last) record number in the file.
RTENO	Select only the message records generated by the RTE process number specified by *value*. The RTE process number is the user's terminal LU.
STARTREC	Select the records on the tape with record numbers greater than or equal to the number specified by *value*. The record numbers are the tape's sequential record numbers. *Default*: the first record number (number 1) in the file.
STREAM	Select only the message records that belong to the stream type specified by *value*. For a list of message streams and the corresponding messages, refer to the *NS Message Formats Manual*.
value	Integer value from x to y, or @ to clear the value. For STREAM, this value is an octal number.
@	Clear all the values set.

DSMOD

DSMOD allows you to alter DS/1000-IV Compatible Services parameters set during initialization, or change timing parameters not set through NSINIT questions.

DSMOD Scheduling

```
DSMOD [ ,lu   ] [,errordevice]
      [ ,file ]
```

lu	The logical unit number of the device from which responses to the DSMOD prompts will be entered. Default is the scheduling terminal.
file	The name of a command file that contains responses to the DSMOD prompts. If the full file path is not specified, the default directory is the current working directory.
errordevice	The logical unit number of the device where errors will be logged. Default is the scheduling terminal.

DSMOD Operation

When DSMOD is run with an interactive input device, you are prompted with the following question:

```
/DSMOD: OPERATION?
```

You can respond to this prompt with any of the commands listed in Table 14-5.

When DSMOD is scheduled with a command file, it obtains the responses to its prompts from the file. Running DSMOD with a command file can be useful if there are changes that need to made on boot-up.

If NS-ARPA is shutdown while DSMOD is running, DSMOD may abort with a request error (RN02).

Commands

The DSMOD commands are summarized in Table 14-5.

Table 14-5. DSMOD Commands

Command	Description
`??`	Prints a list of the DSMOD commands.
`/A`	Aborts DSMOD.
`CN`	Changes the Nodal Routing Vector (NRV).
`DI`	Disables a link. (You cannot use the `DI` command to disable LAN links.)
`/E`	Exits DSMOD.
`/I`	Changes local and remote ID sequences.
`/L`	Enables a link. (You cannot use the `/L` command to enable LAN links or X.25 LUs. (You can use the DSMOD `DI` command to return X.25 LUs to the pool.)
`/N`	Displays the Nodal Routing Vector (NRV).
`/S`	Schedules monitors. (Only the following monitors can be scheduled with DSMOD: CNSLM, DLIST, EXECM, EXECW, OPERM, PTOPM, PROGL, RFAM, RDBAM, VCPMN.)
`/T`	Adjusts network timing values that are used by the DS/1000-IV Compatible Services.

HP 1000 Utilities

The utility subroutines described here are provided for use in conjunction with PTOP, RFA, and DEXEC calls.

The following utility subroutines are part of the DS/1000-IV Compatible Services (RTE-RTE).

DLGNS

Allows non-session access at remote DS/1000-IV nodes with Session Monitor.

```
   DLGNS(err,node,acct,len[,oride])

   sess = DLGNS(err,node,acct,len[oride]
```

sess	*16-bit integer*. The session identifier of the remote session created, or the negative of the session identifier of the existing remote session that is being used.
err	*16-bit integer*. A variable to which error codes are returned.
node	*16-bit integer*. The positive Router/1000 node address, or, if the remote DS/1000-IV node is a neighbor node, the negative communication link LU number. An error is reported if the local Router/1000 node address is entered here.
acct	*Integer array (FORTRAN); Packed character array (PASCAL)*. The password for non-session access at the target node. The password can be up to ten characters in length and can optionally be preceded by a slash.

len	*16-bit integer*. The length of the password in positive (+) words or negative (−) bytes including the optional leading slash.
oride	*16-bit integer*. An integer specifying the degree to which you want to override session sharing. Default is zero (no override). (Refer to DLGON for more information about this parameter.)

DLGOF

Releases the session at the specified remote DS/1000-IV node with Session Monitor.

```
DLGOF(err,node)
```

err	*16-bit integer*. A variable to which logoff error codes are returned. Zero is returned if no errors occurred.
node	*16-bit integer*. The positive Router/1000 node address, or, if the remote DS/1000-IV node is a neighbor node, the negative communication link LU number. An error is returned if the local node number is entered here.

DLGON

Provides non-interactive access to a specific remote account at a DS/1000-IV node with Session Monitor.

```
DLGON(err,node,acct,len[,oride]

sess = DLGON(err,node,acct,len[oride]
```

sess	*16-bit integer*. The session identifier of the remote session created, or the negative of the session identifier of the existing remote session that is being used. DLGON is an integer function.
err	*16-bit integer*. A variable to which logon error codes are returned. Zero is returned if there were no errors.
node	*16-bit integer*. The positive Router/1000 node address, or, if the remote DS/1000-IV node is a neighbor node, the negative communication link LU number. An error is reported if the local node number is entered here.
acct	*Integer array (FORTRAN); Packed character array (PASCAL)*. The name and optional password of the remote account in the form *user.group/password*. The *user*, *group*, and *password* can each be up to ten characters in length.
len	*16-bit integer*. The length of the account name in positive (+) words or negative (−) bytes.
oride	*16-bit integer*. An integer specifying the degree to which you want to override session sharing. For most applications, the default value (0) is sufficient. The *oride* values are:

2	Absolute session sharing override. Do not try to share any existing session.

`1`	Override session sharing with non-ancestors. Share with ancestors. Request a new session even if a non-ancestor program within the calling program's process group already owns a session at the requested node.
`0`	(Default) No override. Share with ancestors or other programs within the calling program's process group.
`-1`	Slave ability. No override. The logon request is refused as above. However, slave back, if possible.

DMESG

Sends a message from a program at your local node to system LU 1 at a remote HP 1000 node.

```
DMESG(dest,msgad,msg)
```

dest	*16-bit integer*. The Router/1000 node address of the node where the message is to be sent.
msgad	*Integer array (FORTRAN); Packed character array (PASCAL)*. ASCII-coded message; less than or equal to 72 characters.
msgl	*16-bit integer*. Message length. A positive value to indicate the number of words, or a negative value to indicate the number of characters in the message.

DMESS

Transmits a system command to the message processor at a remote node.

```
DMESS(dest,bufr,bufl)
```

dest	*16-bit integer*. The Router/1000 node address of the node where the message is to be sent.
bufr (input/output)	*Integer array (FORTRAN); Packed character array (PASCAL)*. Message buffer. Less than or equal to 40 characters. Many RTE-A system messages exceed the *bufr* size limit and will be truncated.
bufl	*16-bit integer*. Message buffer length in bytes. A positive value indicating the number of bytes in the buffer (40 bytes maximum).

Upon return to a program, A-register and B-register contain status information:

A = 0	Indicates that there is no return message from the remote node.
A < 0	Indicates the negative value of the byte count for a return message from the remote node. The return message is contained in the message buffer (*bufr*).

B = −1	Indicates an error condition. In this case, the A-register contains −4 and *bufr* contains a 4-character error message (for example, DS03).

Caution Issuing the system command OF, *program name* via DMESS, where *program name* is the name of any NS-ARPA/1000 program, may result in the failure of a requested transaction and/or close files or terminate programs. DMESS is the remote processing version of the RTE subroutine, MESSS. Refer to the *RTE-A Programmer's Reference Manual*, part number 92077-90007, for more information on MESSS.

DSERR

Returns expanded error information about a reported error in your program's most recent PTOP, RFA, or DEXEC request. Do *not* use DSERR if your PTOP master calls are made to an HP 3000 slave program, if you are using HP 3000 RFA calls, or if you are calling the utility subroutines HELLO, BYE, or PRCNM.

```
DSERR(erbf[,noder][,lqlfr])
```

erbf — *Integer array (FORTRAN); Packed character array (PASCAL)*. ASCII error message return buffer. Must be an array of at least 24 words. The message is of the form DS ERROR: *xxxx*(*qq*), REPORTING NODE *nnnnn* where:

xxxx	May contain either the ASCII or numeric (converted to ASCII) error code.
qq	Is the error code qualifier.
nnnnn	Is the node number reporting the error.

noder — *16-bit integer*. A positive integer to which the reporting node number is returned.

lqlfr — *16-bit integer*. A positive integer to which the qualifier code is returned.

DS_GETNAME

Obtains a node name from /system/nodenames given the node number.

```
Error = DS_GetName(Node,Nodename)
```

Error — *16-bit integer*. An error code is returned here indicating whether a match was made with the node number in the /system/nodenames file or a negative FMP error code.

FMP −302 indicates the node number was not found in the /system/nodenames file.
0 = no error.

Node	*16-bit integer*. The node number of the remote node for which you want the node name. Use the node number from the NRV. The node number is used to search /system/nodenames to find a match and obtain the node name associated with it.
Nodename	*Integer array (FORTRAN); Packed character array (PASCAL)*. An array that contains the node name as found in /system/nodenames. Length is 16 characters.

DS_GETNODE

Obtains a node number from /system/nodenames given the node name.

```
   Node = DS_GETNODE(Nodename)
```

Node	*16-bit integer*. The node number obtained from /system/nodenames as associated with the node name or a negative FMP error code. FMP −302 is equivalent to the “node name was not found in /system/nodenames.”
Nodename	*Integer array (FORTRAN); Packed character array (PASCAL)*. An array which contains the node name of the remote node for which you wish to find the node number. This should be a node name as found in /system/nodenames. It should be left-justified and blank padded.

FCOPY

Copies FMGR files between HP 1000 nodes. Cannot copy non-FMGR files.

```
   FCOPY(fil1,cr1,fil2,cr2,err[,sec][,typ2][,siz2][,rec2][,mode][,sec2])
```

fil1	*Integer array (FORTRAN); Packed character array (PASCAL)*. Origin file name. A 3-word array containing the ASCII-coded name of the file to be copied. Must be a FMGR file.
cr1	*Array of 16-bit integers*. A two-word array. Word one is the cartridge identifier. Must be a positive or negative integer value or zero. If positive, the file search is restricted to the cartridge declared by the specified integer value. If negative, the file search is restricted to the logical unit number declared by the specified integer value. If zero, the file search is not restricted to any particular cartridge. Word two indicates the Router/1000 node address of the node where the origin file resides.
fil2	*Integer array (FORTRAN); Packed character array (PASCAL)*. Destination file name. A 3-word array containing the ASCII-coded name of the file copy. Must be a FMGR file.
cr2	*Array of 16-bit integers*. A two-word array. Word 1 contains the destination file cartridge identifier. Word 2 contains the Router/1000 node address. (See *cr1* for more information.)
err	*16-bit integer*. Error return variable.

sec1	`16-bit integer`. Origin file security code. Specify one if a security code exists for the origin file.
typ2	`16-bit integer`. Destination file type. Specify only if you want the destination file type to differ from that of the origin file.
siz2	`16-bit integer`. Destination file size in blocks. Specify only if you want the destination file size to differ from that of the origin file. If specified, this value must be greater than the block size of the origin file. A negative value should not be specified (it would cause all available remote disk space to be allocated to the destination file if a line failure occurred before the file was truncated and closed). If you attempt to specify a negative value, error code −6 will result.
rec2	`16-bit integer`. Destination file record size in words for Type 2 files only. Specify only if you want the record size of the destination file to differ from that of the origin file. It may not be a negative value.
mode	`16-bit integer`. Transfer mode. Specify a non-zero value in the parameter to cause both the origin file and the destination file to be opened as Type 1. It is usually desirable to enable this mode because variable record length files are then transferred in blocks of 128 words which is faster than record-by-record transfers.
isec2	`16-bit integer`. Destination file security code. Specify only if a different security code is desired on the destination file.

FLOAD

You may use this utility to download an absolute or memory-image program file into memory-based RTE-A nodes.

```
   FLOAD(name,cr,node1,node2,err[,secu][,prtn][,psiz][,erbf])
```

name	Absolute program file name. A 3-word array containing the ASCII-coded name of the program file to be downloaded. The first five characters of the file name are used to fill in the ID segment. Must be a FMGR file.
cr	`16-bit integer`. Program file's cartridge reference number.
node1	`16-bit integer`. The Router/1000 node address of the node at which the file exists. Must be a non-negative node address.
node2	`16-bit integer`. The Router/1000 node address of the node where the file is to be downloaded.
err	`16-bit integer`. Error return.
secu	`16-bit integer`. Program file security code.
prtn	`16-bit integer`. This is a dummy parameter.
psiz	`16-bit integer`. This is a dummy parameter.

erbf	*Integer array (FORTRAN); Packed character array (PASCAL)*. Error buffer. A 3-word array used for the return of an ASCII-coded program name under certain error conditions.

GNODE

Obtains your local Router/1000 node address.

```
   GNODE(node)
```

node	*16-bit integer*. Your Router/1000 node address is returned here. Returns `-1` if NS-ARPA is not initialized.

SEGLD

Locates, loads and executes a program segment from a disk file at an HP 1000 node.

```
   SEGLD(name,err[,p1][,p2][,p3][,p4][,p5]
```

name	A 3-word array that contains the ASCII-coded name of the program segment file to be loaded. Must be a FMGR file.
err	`16-bit integer`. Error return. A variable to which an integer value representing an error code is returned if an error condition is encountered during execution of this call.
ip1...ip5	Up to five 16-bit values that may be passed from the calling program to the called segment.

HP 3000 Utilities

The utility subroutines described here are provided for use in conjunction with PTOP, RFA, and DEXEC calls.

The following utility subroutines are part of the DS/1000-IV Compatible Services (RTE-MPE).

HELLO

Establishes communication between your NS-ARPA/1000 system and an NS/3000 or DS/3000 node. Creates a session at the remote HP 3000.

```
   HELLO(err,ldev,lstdv,nmsmp,logr,logrl[,lux.25]
```

err	*16-bit integer*. An error code is returned here if an error condition is encountered. Upon successful completion of the call to `HELLO`, the value of *err* is zero.
ldev	*16-bit integer*. The logical unit number of an HP 3000 (an integer less than 256) or an X.25 address (up to 15 ASCII-coded digits).

lstdv	*16-bit integer*. The logical unit number of the desired list device. The “logon” response generated at the HP 3000 as a result of a successful `HELLO` operation is transmitted to this device. Zero equals the scheduling terminal.
nmsmp	*16-bit integer*. The session (SMP) number is returned here from the HP 3000.
logr	*Integer array (FORTRAN); Packed character array (PASCAL)*. An array that contains the `HELLO` command logon parameters in the form of a message. The first six characters of this logon message must be the characters HELLO followed with a blank. The entire message string must also be followed (terminated) with a blank.
logrl	*16-bit integer*. The length (in characters) of the logon message contained in the *logr* array.
lux.25	*16-bit integer*. The LU associated with an X.25 network. If this parameter is omitted and an X.25 address is passed in *ldev*, `HELLO` passes zero in *lux.25*. This zero is used by the X.25 virtual circuit allocation routine which indicates that X.25 will use the first network in SAM (the last network entered from `XINIT`).

BYE

Terminates logical communication between an HP 1000 program and an HP 3000 program. Also terminates the HP 3000 session in which the program is running.

```
   BYE(err,ldev,lstdv,nmsmp)
```

err	*16-bit integer*. An error code is returned here if an error condition is encountered when issuing the `BYE` command. Upon successful completion of the call to `BYE`, the value of *err* is zero.
ldev	*16-bit integer*. The logical unit number (integer < 256 >) of an HP 3000. (If the HP 3000 has an X.25 address (up to 15 ASCII-coded digits), FORTRAN type is *integer array* and Pascal type is *packed character array*.)
lstdv	*16-bit integer*. The logical unit number of the desired local list device. The “logoff” message generated by the HP 3000 as a result of a successful `BYE` operation is transmitted to this device. Zero equals the scheduling terminal.
nmsmp	*16-bit integer*. The session (SMP) number obtained via this session’s corresponding `HELLO` command.

LU3K (for X.25 Links)

Obtains the LU number of a virtual circuit allocated for an HP 1000 to HP 3000 X.25 connection.

```
   a3klu = LU3K([dumy])
```

a3klu	`16-bit integer`. The LU number of the X.25 virtual circuit is returned here.
dumy	Dummy parameter required for FTN4x (and earlier FORTRAN compilers). Omit this parameter for Pascal and programs written in later versions of FORTRAN (FTN77/7x).

PRCNM

Establishes communication between a son program and a session (SMP) created by a father program at an HP 3000.

```
   PRCNM(parm)
```

parm	`16-bit integer`. May be any non-zero value to establish communication with the HP 3000 session. If your node is memory-based, *parm* can be set to zero to clear the communication values set by a previous run of the program.

Index

This section is a master index for the NS-ARPA/1000 manual set. The following codes identify the NS-ARPA/1000 manuals.

Code	Manual Title
USR	*NS-ARPA/1000 User/Programmer Reference Manual*, part number 91790-90020.
GEN	*NS-ARPA/1000 Generation and Initialization Reference Manual*, part number 91790-90030.
MNT	*NS-ARPA/1000 Maintenance and Principles of Operation Manual*, part number 91790-90031.
DS	*NS-ARPA/1000 DS/1000-IV Compatible Services Reference Manual*, part number 91790-90050.
FSV	*File Server Reference Guide for NS-ARPA/1000 and ARPA/1000*, part number 91790-90054.
BSD	*BSD IPC Reference Manual for NS-ARPA/1000 and ARPA/1000*, part number 91790-90060.
MSG	*NS Message Formats Reference Manual*, part number 5958-8523.

For example, *GEN* 4-35 means that the entry is on page 4-35 of the *NS-ARPA/1000 Generation and Initialization Reference Manual*