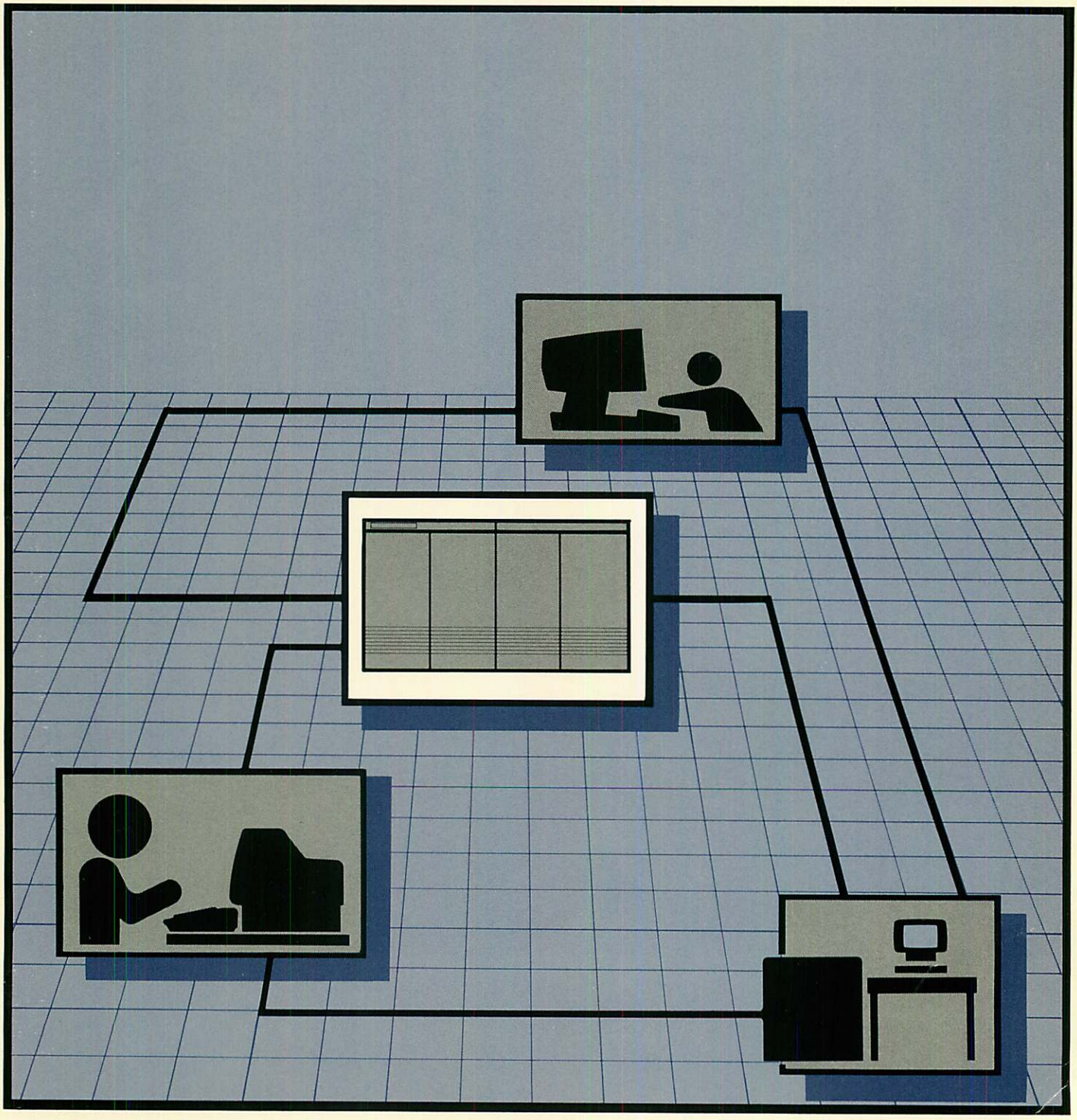
HP AdvanceNet

HEWLETT PACKARD

DS/3000 HP 3000 to HP 3000

User/Programmer Reference Manual



HP AdvanceNet

DS/3000 HP 3000 to HP 3000 User/Programmer

Reference Manual

HEWLETT PACKARD
19420 HOMESTEAD ROAD, CUPERTINO, CA 95014

Part No. 32185-90001
U0787

Printed in U.S.A. DECEMBER 1985
Update 1, JULY 1987

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition	DEC 1985	32185B.52.00 (MPE V/E)
First Edition	DEC 1985	32189B.01.00(MPE IV)
Update #1	JUL 1987	32185B.52.00(MPE V/E)
Update #1	JUL 1987	32189B.01.00(MPE IV)

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars are removed but the dates remain. No information is incorporated into a reprinting unless it appears as a prior update. To verify that your manual contains the most current information, check that the date printed at the bottom of the page matches the date listed below for that page.

Effective Pages	Date
2-19 to 2-20	JUL 1987
8-1 to 8-16	JUL 1987

PREFACE

The Hewlett-Packard AdvanceNet is a set of hardware and software data communications products. One of these data communications products is DS/3000, an integrated software package that provides the capability of communication between HP computer systems.

This manual documents DS/3000 as it applies to an HP 3000 network. The manual explains how an HP 3000 user can communicate with another (or several other) HP 3000 computer systems by establishing a DS/3000 communications link. (Other manuals in the DS/3000 series document the other network combinations of computer types.)

This manual explains the basic use of DS/3000 to users and programmers. A companion manual, the *DS/3000 HP 3000 to HP 3000 Network Administrator Manual*, explains more advanced concepts such as configuring a system, and using TRACE for debugging.

Users of this manual should be familiar with the basic operating principles of the HP 3000 computer system using the MPE operating system, and should also be familiar with the subjects covered in the following manuals:

For MPE-IV (Versions not earlier than C.B1.A2):

- *HP 3000 Computer Systems, MPE Commands Reference Manual (30000-90009).*
- *HP 3000 Computer Systems, MPE Intrinsics Reference Manual (30000-90010).*
- *HP 3000 Computer Systems, System Manager/System Supervisor Reference Manual (30000-90014).*
- *HP 3000 Computer Systems, Console Operator's Guide (32002-90004).*

For MPE-V/E (Versions not earlier than G.00.00):

- *HP 3000 Computer Systems, MPE V Commands Reference Manual (32033-90006)*
- *HP 3000 Computer Systems, MPE V Intrinsics Reference Manual (32033-90007)*
- *HP 3000 Computer Systems, MPE V System Operation and Resource Management Reference Manual (32033-90005)*

For both:

- *Fundamental Data Communications Handbook (5957-4634)*
- *Data Communications Handbook, Section C, DS/3000 to 3000 (32185-90003)*
- *Data Communications Handbook, Section D, DS/3000 to 1000 (32185-90006)*
- *Data Communications Handbook, Section G, X.25 Link (32187-90006)*

For those users who also become involved in the selection and/or connection of the various network components, reference should be made to the appropriate component manuals, including the following:

- *HP 30010A Intelligent Network Processor (INP) Installation and Service Manual (30010-90001).*
- *HP 30020A Intelligent Network Processor (INP) Installation and Service Manual (30020-90001).*
- *HP 30020B Intelligent Network Processor (INP) Installation and Service Manual (30020-90005).*
- *HP 30010A/30020A/B Intelligent Network Processor (INP) Diagnostic Procedures Manual (30010-90002).*
- *HP 30055A Synchronous Single-Line Controller (SSLC) Installation and Service Manual (30055-90001).*
- *Hardwired Serial Interface (HSI) Installation and Service Manual (30360-90001).*

For those programmers who use other subsystems in conjunction with DS/3000, the following manuals should be referenced:

- *TurboIMAGE Reference Manual (32215-90050)*
- *BASIC/3000 Interpreter Manual (30000-90026)*
- *COBOL/II Reference Manual (32233-90001)*
- *KSAM/3000 Reference Manual (30000-90079)*

NOTE

Within the text of this manual, cross-references are made to these manuals by title. To obtain the part number of the referenced manual, refer to these lists of manuals in the Preface.

In this release, DS/3000 and X.25 Link are two separate products. If you are using DS/3000 between HP 3000 computers, either with or without X.25, you should use this manual for all DS information and refer to the *DS/3000 HP 3000 to HP 3000 Network Administrator Manual (32185-90002)* for information about protocols, hardware, configuration, and tracing. If you are communicating with a PAD terminal or an HP 2334A Cluster Controller, refer to the *X.25 Link for the HP 3000 Reference Manual (32187-90001)*.

CONVENTIONS USED IN THIS MANUAL

NOTATION	DESCRIPTION
nonitalics	Words in syntax statements which 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 which are in italics denote a parameter which 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: [A B] 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: {A B C} 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 which 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*);

underlining

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

NEW NAME? ALPHA

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

LET *var*[[*subscript*]] = *value*

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

CREATE (*parm1*,*parm2*,*flags*,*error*)

shading

Shading represents inverse video on the terminal's screen. In addition, it is used to emphasize key portions of an example.

The symbol ( ) may be used to indicate a key on the terminal's keyboard. For example, (RETURN) indicates the carriage return key.

(CONTROL)*char*

Control characters are indicated by (CONTROL) followed by the character. For example, (CONTROL)Y means the user presses the control key and the character Y simultaneously.

CONTENTS

CONTENTS (continued)

Section 5 USING A REMOTE DATA BASE

Section 6 PROGRAM-TO-PROGRAM COMMUNICATIONS

Section 7 NETWORK FILE TRANSFER

Section 8 DS APPLICATION DESIGN

CONTENTS (continued)

Appendix C DS/3000 BASIC INTERFACE

INTRODUCING DS/3000

SECTION 1

The Hewlett-Packard AdvanceNet is a combination of hardware and software products that make it possible for Hewlett-Packard computer systems to communicate with one another, and with IBM mainframes as well. The connections can be made over hardwired lines, and/or over the public telephone facility, and/or across Public Data Networks (PDNs), in any mixture. This capability, coupled with our proven remote entry capability to IBM computer systems, provides a total solution to large-company electronic data processing (EDP) needs.

But exactly what does this overall capability mean? It means that a large multidivisional corporation can have a truly coordinated world-wide network of computer systems. They are coordinated in the sense of tying together the various commercial and industrial functions within each division and factory, and they are also coordinated in the larger sense of tying together the various divisions and factories at the corporate level.

For example, imagine a large corporation which has factories in the United States, Canada, France, and West Germany. Within each factory there are HP 3000 computer systems performing such functions as inventory control, factory data collection, and operations management. With a Hewlett-Packard Distributed Systems Network these manufacturing information systems can be tied into an HP 3000 system which handles the factory's administrative functions (such as finance and accounting). The administrative systems of each factory can, in turn, be connected not only to one another but also (via remote job entry) to a large computer facility at corporate headquarters. This overall networking capability makes it possible to perform financial analysis and control at a group and corporate level as well as at the individual factories.

Within the realm of AdvanceNet is the software subsystem that accomplishes HP computer-to-HP computer communication over these connecting lines. This software subsystem is called DS Network Services (DS/3000). Among other features, DS/3000 includes such capabilities as:

- **Virtual Terminal/Remote Command Execution.** Gives the user remote interactive capabilities, even though the user's terminal is physically connected to the local HP system.

- **Remote File Access.** A user is allowed access to files in remote HP computer systems. An important aspect of this feature is the capability of using Interprocess Communications (IPC) between systems.

- **Remote Data Base Access.** A user can directly access data bases on any remote HP computer under the same security protection used by local data bases.

- **Program-to-Program Communication.** Permits programs residing in different HP computer systems to interactively exchange information with one another in a coordinated manner.

- **Network File Transfer.** A facility that efficiently transfers disc files between HP 3000 computer systems.

This manual describes how an HP 3000 user can communicate with several HP 3000 computers by establishing a DS/3000 communications link. DS/3000 is that part of the HP Distributed Systems Network in which several HP 3000 computer systems are connected to one another. DS/3000 can also be used for intercomputer communications with other families of computers (such as HP 3000/HP 1000, HP 3000/HP 250, and HP 3000/HP 98xx desktop computers), but these other combinations are described in separate reference manuals.

As a simplified example of a computer network, imagine that you are in the same room with an HP 3000 (labeled "System A" in Figure 1-1) and that another HP 3000 (labeled "System B") resides in another part of the building. These two computers are connected to one another by an interconnecting cable and a pair of communications interfaces. By virtue of DS/3000 you can use the processing capability of both of these HP 3000 machines and pass data back and forth between them by entering commands through a single terminal.

To see how DS/3000 works in this simple example, follow through the step-by-step procedure.

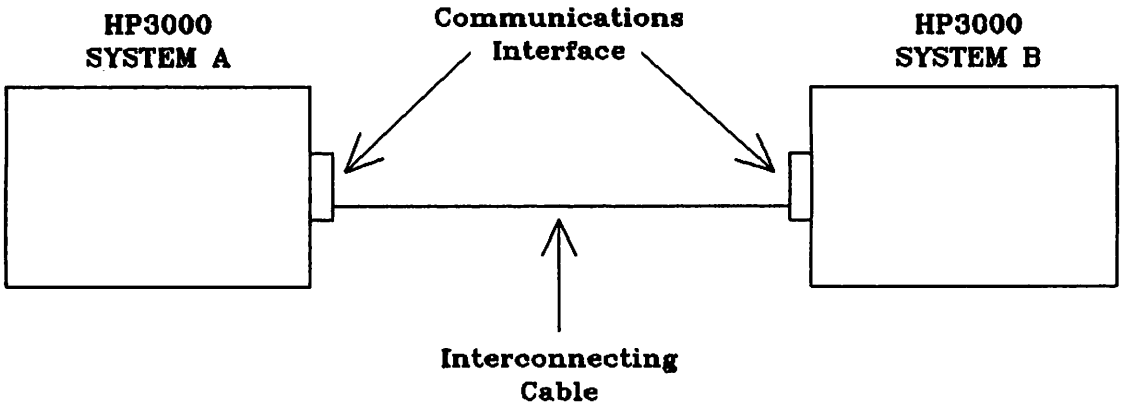


Figure 1-1. HP 3000 to HP 3000 Example

Step 1. Sit down at a terminal connected to System A and initiate a session.

```
RETURN
:HELLO USER.ACCOUNT

HP3000 / MPE V  G.02.00.   MON, SEP 30, 1985, 12:23 PM


WELCOME TO SYSTEM A.
:
```

Within the context of DS/3000, such a session is referred to as a "local" session because it is active within the HP 3000 to which your terminal is directly connected. This terminology becomes more meaningful later, since all you have actually done, so far, is initiate a standard MPE session. At this point, you have reached the situation illustrated in Figure 1-2.

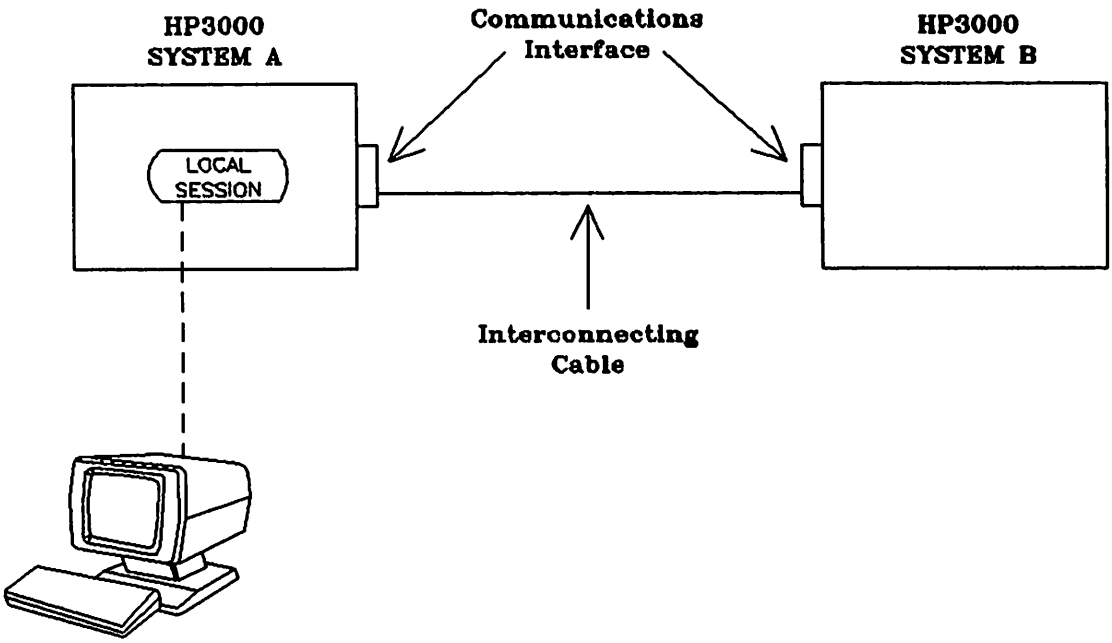


Figure 1-2. Initiating the Local Session

Step 2. Now, open a communications line between System A and System B. Do this by entering a :DSLINE command.

```
:DSLINE REMOTE1
DS LINE NUMBER = #L3
:
```

In this example, REMOTE1 is the DS device class name established during system configuration (in System A) for the particular line you wish to use. DS/3000 opens the line and then assigns you a line number (3 in this example). This line number is analagous to the file number returned to you by the MPE File System when you open a file programmatically using the FOPEN intrinsic. Within your local session, it uniquely identifies the particular line that you have opened. This becomes significant only if you must open more than one communications line during a session.

Step 3. Now that you have acquired access to a communications line between System A and System B, initiate a session in System B (from your local log-on terminal). Do this by entering a REMOTE command which includes an MPE HELLO command for the remote system.

```
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   MON, SEP 30, 1985, 12:23 PM

WELCOME TO SYSTEM B.

:
```

Within the context of DS/3000, this type of session is referred to as a "remote" session because it is active within the remotely located HP 3000 that is connected indirectly to your log-on terminal by way of a communications line and your local HP 3000. You now have two distinct sessions in progress concurrently: one in System A (under the user and account names USER.ACCOUNT) and the other in System B (under the user and account names RUSER.RACCOUNT). It is important to keep in mind that within System A your local session is operating under the capabilities and security restrictions defined (by the accounting structure of System A) for USER.ACCOUNT, while within System B your remote session is operating under the capabilities and security restrictions defined (by the accounting structure of System B) for RUSER.RACCOUNT. At this point, the situation is as illustrated in Figure 1-3. As will be seen in the next few steps, you can alternate freely between the two sessions.

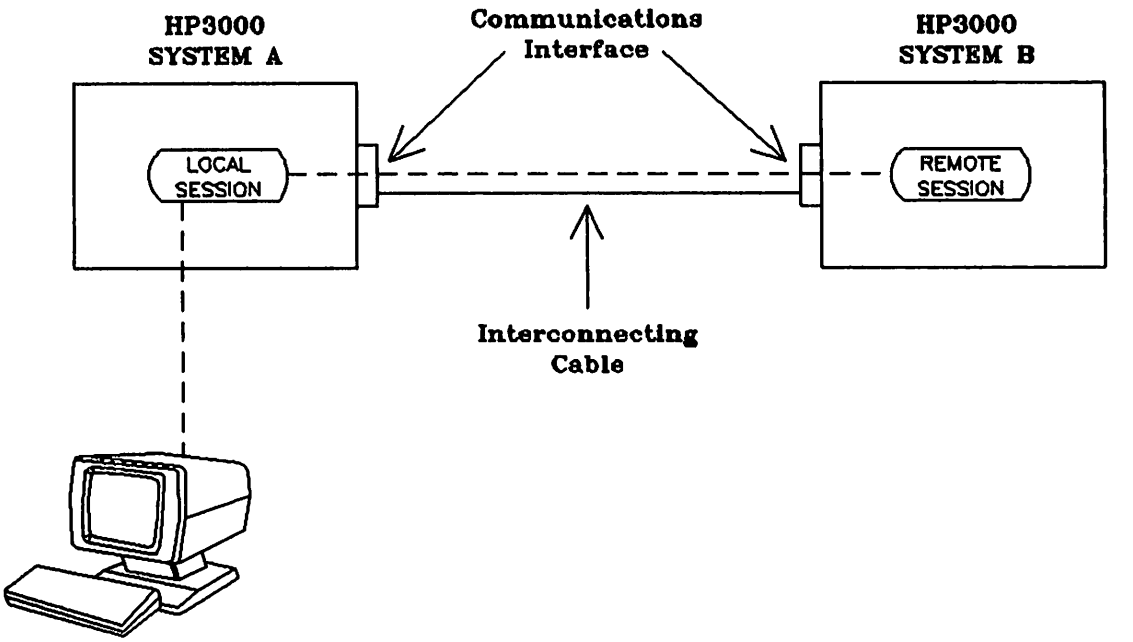


Figure 1-3. Initiating the Remote Session

Step 4. Now, see what files reside in the home group of the ACCOUNT account in System A.

```
:LISTF

FILENAME

DATA1     DATA3     FILE1     SOURCE2   SOURCE5
:
```

You can do the same for the home group of the RACCOUNT account in System B by entering the following command through the same terminal:

```
:REMOTE LISTF

FILENAME

DATA1     DATA5     DATA6     FILE3     SOURCE1
:
```

Notice that in both cases the same command was entered, but in the latter case the prefix REMOTE was used. The presence or absence of that prefix is what determines whether a command is to be executed in the local session or in the remote session.

Step 5. As a result of the LISTF and REMOTE LISTF displays, you can see that a source file, named SOURCE1, exists in System B but not in System A. Suppose you wish to modify one of the statements in that program. To do that, use the text editor in System B. This time, instead of prefixing your remote commands with REMOTE, try a different technique. Enter the following:

```
:REMOTE
#
```

This construct gets into the remote session in such a way that all commands can be entered in their normal form (without the prefix REMOTE). The # is the prompt character issued by DS/3000 (in place of the usual MPE colon prompt). In all other respects it will seem as though you are executing an MPE interactive session on your local computer.

Step 6. Now invoke the text editor, copy the content of SOURCE1 (which is a file in System B) into the editor's work file, display the content of the work file, modify the desired statement, and store the altered source code back in SOURCE1.

```
#EDITOR
HP32201A.7.17 EDIT/3000 WED, MAR 6, 1985, 3:47 PM
(C) HEWLETT-PACKARD CO. 1985
/SET FORMAT=COBOL
/T SOURCE1
/LIST ALL
    1 $CONTROL USLINIT,SOURCE
    1.1 IDENTIFICATION DIVISION.
    1.2 PROGRAM-ID. COBOL-TEST1.
    1.3 ENVIRONMENT DIVISION.
    1.4 DATA DIVISION.
    1.5 WORKING-STORAGE SECTION.
    1.6 77 EDIT-FIELD PIC $$$$$9.99.
    1.7 77 TOTAL-COST PIC 999V99.
    1.8 77 COST-OF-SALE PIC 99V99.
    1.9 77 TAX PIC 99V99.
    2 77 Y-N PIC X.
    2.1 PROCEDURE DIVISION.
    2.2 ENTER-ROUTINE.
    2.3 MOVE ZEROS TO TOTAL-COST.
    2.4 DISPLAY SPACE.
    2.5 DISPLAY "ENTER COST OF SALE".
    2.6 ACCEPT COST-OF-SALE.
    2.7 COMPUTE TAX = COST-OF-SALE * .06.
    2.8 ADD COST-OF-SALE, TAX TO TOTAL-COST.
    2.9 MOVE TOTAL-COST TO EDIT-FIELD.
    3 DISPLAY "TOTAL COST= " EDIT-FIELD.
    3.1 DISPLAY "ARE YOU FINISHED? (Y OR N)".
    3.2 ACCEPT Y-N.
    3.3 IF Y-N = "N" OR "n" GO TO ENTER-ROUTINE.
    3.4 STOP RUN.
```

```
/MODIFY 2.5
MODIFY 2.5
     DISPLAY "ENTER COST OF SALE".
                                   I (NO DECIMAL POINT)
     DISPLAY "ENTER COST OF SALE (NO DECIMAL POINT)".

/KEEP SOURCE1
SOURCE1 ALREADY EXISTS - RESPOND YES TO PURGE OLD AND KEEP NEW
PURGE OLD? YES
/EXIT

END OF SUBSYSTEM
#
```

Step 7. The work in System B is now completed; so terminate the remote session and return control to your local session.

```
#BYE

CPU=4. CONNECT=7. WED, MAR 6, 1985, 9:15 AM
#:
:
```

Note that you are now back in the local session in System A (signified by the colon prompt). The remote session no longer exists, but the communications line is still open. You could, if you wanted, initiate another remote session over the line by issuing another REMOTE HELLO command. To close the communications line, enter the following variation of the :DSLINE command:

```
:DSLINE REMOTE1 ;CLOSE
1 DS LINE WAS CLOSED.
:
```

Finally, terminate the local session.

```
:BYE

CPU=1. CONNECT=11. WED, MAR 6, 1985, 9:16 AM
```

THE COMMUNICATIONS LINK

SECTION 2

WHAT IS A COMMUNICATIONS LINK?

Within the context of DS/3000, a "communications link" consists of the following elements:

- An interactive session in progress on an HP 3000 computer.
- A physical communications line between that HP 3000 computer and another HP 3000 computer at a remote location.
- An interactive session in progress in the remote HP 3000 computer (initiated over the physical communications line from your local session).

Note that your local terminal is the log-on terminal for both the local session and the remote session. (See Figure 2-1.)

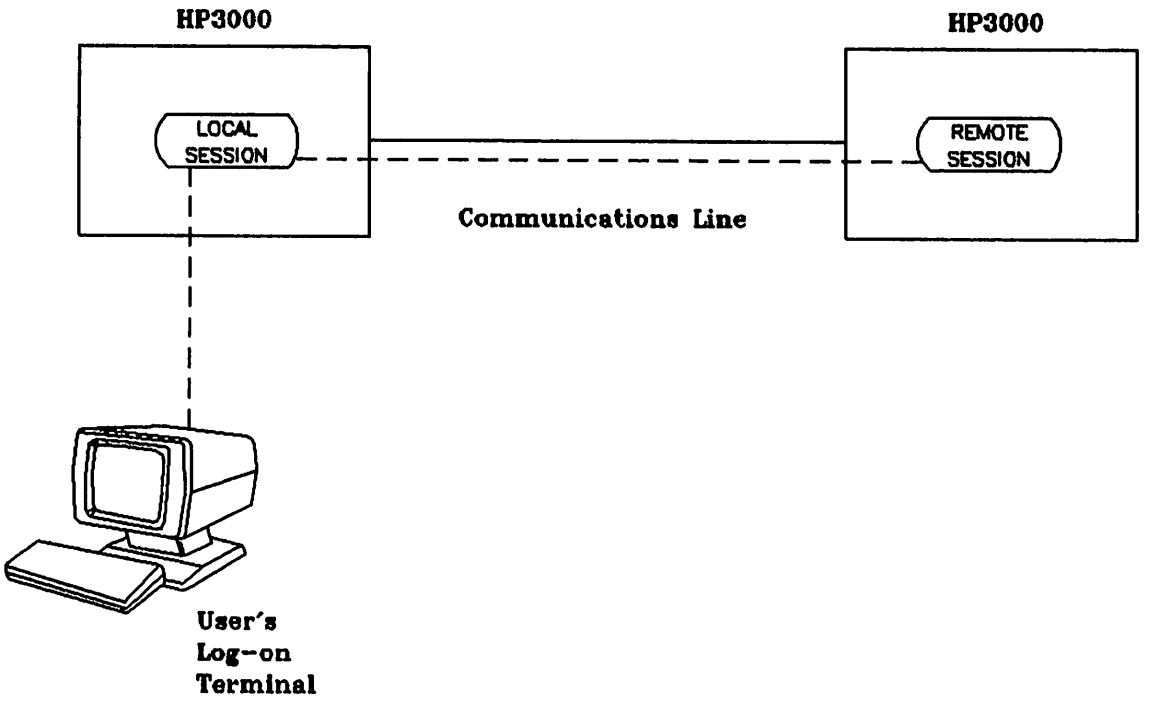


Figure 2-1. DS/3000 Communications Link (HP 3000 to HP 3000)

OPENING A LINE

A communications link can be established over a hardwired communications line, over the public telephone network, or over an X.21 or X.25 Public Data Network (PDN). The procedures for opening hardwired lines and for opening telephone lines differ only slightly. Therefore, the basic differences will be presented first, followed by the procedures that are essentially the same. Generally, once the connection to the remote computer is established, you will perceive no difference in the way DS/3000 performs.

Opening a Hardwired Line

What is a hardwired line? In the general field of data communications there are two types of lines commonly referred to as "hardwired." The first type is a dedicated path on the public telephone network that is leased from the telephone company for the private use of a computer-to-computer configuration. Such a line serves as a permanent connection between the two computers. The other type of hardwired line is a cable that is connected directly to the communications I/O interfaces of the two computers. Within the context of DS/3000, "hardwired" always refers to a cable connection. However, the technique for opening a line is the same for either a direct-connect line or a leased (nonswitched) telephone line.

The hardwired interconnecting cable connects to each HP 3000 by way of a communications interface. The communications interfaces that can be used for a hardwired connection are listed in Table 2-1. Although the INPs and the SSLC are the interfaces most commonly used for telephone line connections with modems, they can also be used in hardwired applications without modems. Refer to Table 2-1 to see which controller is used with which HP system.

Table 2-1. Associations between Controllers and Systems.

Controller	System
HP 30010A Intelligent Network Processor (INP)	HP 3000 Series II/III
HP 30020A Intelligent Network Processor	HP 3000 Series 30/33/39/40/42/44/48
HP 30020B Intelligent Network Processor	HP 3000 Series 30/33/39/40/42/44/48/64/68
HP 30055A Synchronous Single-Line Controller (SSLC)	HP 3000 Series II/III
HP 30360A Hardwired Serial Interface (HSI)	HP 3000 Series II/III

It is relatively straightforward to obtain access to a hardwired communications line. All you are required to do is identify the particular communications interface you wish to use.

NOTE

Throughout this manual, the term "dsdevice" will be used as a generic term for logical device number, class name or, in the case of X.25, a node name.

You identify the particular communications interface of the particular line you wish to use by specifying the dsdevice associated during system configuration with the desired interface. In the example in Section 1, the :DSLINE command was used for this purpose, as follows:

```
:DSLINE REMOTE1
```

If you are the first person to use a :DSLINE command after the operator has enabled the DS line, you may also wish to specify the size of the DS/3000 line buffer to be used in conjunction with the line. The size of this buffer determines the maximum amount of data that can be sent or received in a single physical transmission over the line. Note that a transmission as you normally think of it (sending or receiving all or part of a file) may actually consist of many physical transmissions. In essence, this buffer size defines a blocking factor for the line. (See Figure 2-2.) A default buffer size is established during system configuration, and in most cases (as in the example in Section 1), you will find it satisfactory to let this default value prevail. If you do wish to change the line buffer size, use the *linebuf* parameter as described on pages 2-17 and 2-18.

Assume that the DS/3000 line buffer size is 512 words and that the user has initiated the transmission of a block of data 1200 words in length from an HP 3000 to a remote HP 3000. The block of data would actually be sent in three separate transmissions, as follows. (DS/3000 appends an average of 20 words of overhead on each transmission.)

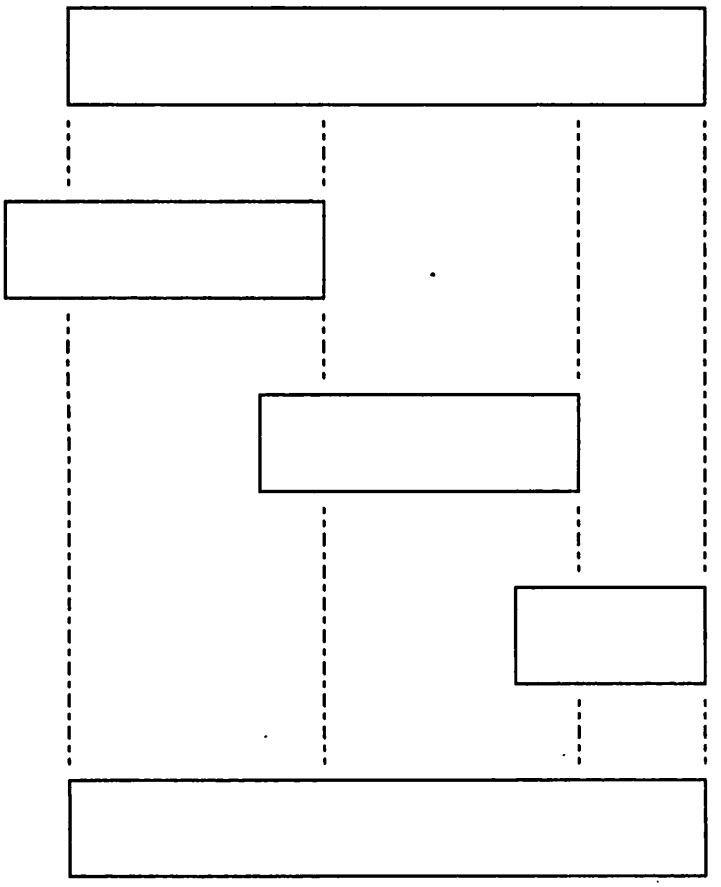


Figure 2-2. DS/3000 Line Buffer Example

When you enter a :DSLINE command, DS/3000 attempts to give you access to the specified communications line and, if successful, informs you of the assigned DS line number by displaying the following message at your terminal:

```
DS LINE NUMBER = #Lx
```

where x is the assigned DS line number. In the example in Section 1, the DS line number "3" was assigned. The DS line number is significant only if you open and use more than one communications line concurrently within a single local session (see "Opening Multiple Lines" later in this section).

At this point you have acquired a physical communications line but the communications link does not yet exist. The actual communications link between the two computers is established by initiating a remote session over the line. You do this by executing a REMOTE HELLO command. In the example in

Section 1, the REMOTE HELLO command contained the minimum parameters required (a username and an accountname), as follows:

```
:REMOTE HELLO RUSER.RACCOUNT
```

The communications link between the two HP 3000 computers now exists.

Opening a Telephone Line

A DS/3000 communications link can also be established over the public (dial-up) telephone network. In such a case, the information passed back and forth between the two computers travels over the same lines that are used for normal voice traffic. Each computer is interfaced to the telephone lines by way of a modem. (The term "modem" is a contraction of MOdulator-DEModulator.) A modem is a device that translates digital signals (electrical impulses) generated by a computer into analog signals (tones) that can be transmitted over telephone lines, and vice versa.

The modem is connected to the HP 3000 Computer System by way of a communications interface. The communications interfaces used with modems are listed in Table 2-2. Refer to Table 2-2 to see which controller is used with which HP system. Each INP or SSLC controls one modem (such as an HP 37210T, 37220T, or 37230A modem, or a Bell System Type 201, 208, or 209 modem), and is capable of both initiating and accepting a telephone connection with a remote computer over the public telephone network or a leased telephone line.

Table 2-2. Associations between Controllers and Systems.

Controller	System
HP 30010A Intelligent Network Processor (INP)	HP 3000 Series II/III
HP 30020A Intelligent Network Processor	HP 3000 Series 30/33/39/40/42/44/48
HP 30020B Intelligent Network Processor	HP 3000 Series 30/33/39/40/42/44/48/64/68
HP 30055A Synchronous Single-Line Controller (SSLC)	HP 3000 Series II/III

It is a little more complex to obtain access to a telephone line than to a hardwired line. First, you must identify the particular communications interface (INP or SSLC) you wish to use. You do this by specifying the device class name or logical device number of the communication line that was associated during system configuration with the desired interface. You use the :DSLINE command for this purpose, as follows:

```
:DSLINE REMOTE2
```

If you are the first person to use a :DSLINE command after the operator has enabled the DS line, you may also wish to specify the size of the DS/3000 line buffer to be used in conjunction with the line. The size of this buffer determines the maximum sized block that can be sent or received in a single physical transmission over the line. Note that a transmission as you normally think of it (sending or receiving all or part of a file) may actually consist of many physical transmissions. In essence, this buffer size defines a blocking factor for the line. (See Figure 2-2.) A default buffer size is established during system configuration, and in most cases (as in the example in Section 1), you will find it satisfactory to let this default value prevail. If you wish to change the line buffer size, use the *linebuf* parameter as described on pages 2-17 and 2-18.

Next, you may wish to supply a set of identification (ID) sequences to be used in verifying that the desired pair of computers are connected to one another. This is discussed under "ID Sequences" later in this section. Briefly, however, you may supply an ID sequence that identifies your HP 3000 and one or more ID sequences that identify the remote computer with which you wish to be connected. When a telephone connection is established between your HP 3000 and a remote HP 3000, the two computers exchange ID sequences and their validity determines whether or not the connection is to remain in effect. You use the :DSLINE command to supply ID sequences, as follows:

```
:DSLINE REMOTE2 ;LOCID="SYSTEM A" ;REMID="SYSTEM X"
```

where SYSTEM A is the ID sequence identifying your local HP 3000 and SYSTEM X is the ID sequence identifying the remote computer with which you want to establish a telephone connection.

Again, there are default values that can be established during system configuration. In most cases, however, you will at least want to explicitly identify the desired remote HP 3000 to be certain that the proper connection is being established.

Now you must establish the physical connection between the two computers by dialing (at the modem) the telephone number of the remote computer and responding (at the system console) to the dial request. If you wish to have the console operator of your HP 3000 dial the number for you, you may supply the desired number in the :DSLINE command and it will be displayed as part of a dial request message at the operator's console. In such a case, you would supply the telephone number as follows:

```
:DSLINE REMOTE2 ;LOCID="SYSTEM A" ;REMID="SYSTEM X" ;PHNUM=555-1234
```

If autodial equipment is installed on the REMOTE2 line, the telephone number supplied in the :DSLINE command is used instead of the number configured for the line.

The various possibilities involved in establishing a telephone connection with a remote computer are discussed under "Dialing the Remote Computer" later in this section.

When you execute the :DSLINE command, DS/3000 attempts to give you access to the specified communications interface (INP or SSLC) and, if the telephone connection is successfully established, informs you of the assigned DS line number by displaying the following message at your terminal:

```
DS LINE NUMBER = #Lx
```

where x is the assigned DS line number. In the example in Section 1, the DS line number "3" was assigned. The DS line number is significant only if you open and use more than one communications line concurrently within a single local session (see "Opening Multiple Lines" later in this section).

At this point, you have acquired a physical communications line, but the communications link does not yet exist. The actual communications link between the two computers is established by initiating a remote session over the line. You do this by executing a `:REMOTE HELLO` command. In the example in Section 1, a `:REMOTE HELLO` command was used that contained the minimum parameters required (a username and an accountname), as follows:

```
:REMOTE HELLO RUSER.RACCOUNT
```

The communications link between the two HP 3000 computers now exists.

Specifying a DS Line

As you have seen, in order to open either a hardwired communications line or a dial-up telephone line, you must specify a dsdevice identifying the particular communication line that is associated with the specific interface you wish to use. Deciding which name and number to use is complicated, but in real life, once the hardware and software configuration is installed and usable, most DS/3000 sites will post a notice defining all of the available communications lines and the proper device class names, logical device numbers, or nodenames for each. The following examples describe the procedure in case you wish to know how it works, or if your system does not post them.

For each communications interface, there is a pair of associated drivers. First, there is the actual INP, HSI, or SSLC driver that directly controls the operation of the interface board. In addition, there is a DS/3000 communications driver that controls the operation of the INP, HSI, or SSLC driver. The names of these drivers are as follows:

IOINP0 (INP driver)
CSHBSC0 (HSI driver)
CSSBSC0 (SSLC driver)
IODS0 (DS/3000 communications driver, while utilizing the bisync protocol)
IODSX (DS/3000 communications driver, while utilizing the X.25 Link software)
IODSTRM0 (DS/3000 virtual terminal driver, while utilizing the bisync protocol)
IODSTRMX (DS/3000 virtual terminal driver, while utilizing the X.25 Link software)

Now look at the appropriate sample I/O device table produced during system configuration. (See Figures 2-3 and 2-4 for a hardwired line or Figures 2-5 and 2-6 for a telephone line).

One or more virtual terminal drivers (IODSTRM0 or IODSTRMX) are also configured into a system. The IODSTRM0/IODSTRMX entries allow users on another system to be logged on to a system and regulate the number of remote Session Main Processes (SMP) that can be assigned to a given line. Each IODSTRM0/IODSTRMX entry is related to the proper communications interface entry by the number specified in the column labeled DRT. Figure 2-4 (the INP hardwired example) shows logical device 67 paired with logical device 27; Figure 2-6 (the INP telephone line example) shows logical device 68 paired with logical device 26.

In Figure 2-3, notice that the HSI board entries (logical devices 12 through 15) look the same except for the PORTMASK. The PORTMASK specifies which port on the board is to be used. There are also virtual terminals (logical devices 60 through 64) referencing back to logical device 12. Since only one port on the HSI board can be opened at a time, only one block of virtual terminal entries is needed for that board. As each port is opened individually by specifying the corresponding dsdevice in the :DSCONTROL command (see Section 9), the system automatically reallocates the virtual terminal entries to the proper HSI board entry. This reallocation will not, however, show up in the I/O configuration table. This use of virtual terminals is unique to the HSI. SSLC and INP configurations require a block of terminals for each interface configured.

In Figure 2-3, the shaded items in the column labeled DRIVER NAME show four HSI lines (CSHBSC0) configured into the system as logical devices 12 through 15. For each one of these lines, there is a DS/3000 communications driver, IODS0, also configured into the system. Each IODS0 entry is related to the proper HSI entry by the number specified in the column labeled DRT (the # prefix indicates a back reference to a previously defined logical device number). Logical devices 50 through 53 are paired with logical devices 12 through 15, respectively. In this example, it is the device class name or logical device number of the appropriate IODS0 entry that would be used to specify the desired line.

LOG DEV #	DRT #	UNIT	CHAN	TYPE	SUB TYPE	TERMINAL TYPE	TERMINAL SPEED	REC WIDTH	OUTPUT DEV	MODE	DRIVER NAME	DEVICE CLASSES
1	4	0	0	0	6			128	0		IOMDISC1	SPOOL SYSDISK
2	5	0	0	0	3			128	0		IOMDISC0	DISC
5	13	0	0	8	0			40	LP	JA S	IOCRD0	CARD
6	14	0	0	32	2			66	0	S	IOPRT0	LP
7	6	0	0	24	0			128	LP		IOTAPE0	TAPE
8	6	1	0	24	0			128	LP		IOTAPE0	TAPE
9	6	2	0	24	0			128	LP		IOTAPE0	TAPE
10	6	3	0	24	0			128	LP	JA S	IOTAPE0	BATAPE
11	20	0	0	34	0			128	0		IOPTPN0	PTPUNCH
12	16	0	0	19	3			0	0		CSHBSC0	HSI1
13	16	0	0	19	3			0	0		CSHBSC0	HSI2
14	16	0	0	19	3			0	0		CSHBSC0	HSI3
15	16	0	0	19	3			0	0		CSHBSC0	HSI4
20	7	0	0	16	0	10	??	40	20	JAID	IOTERM0	CONSOLE
21	7	1	0	16	0	11	??	40	21	JAID	IOTERM0	TERM
22	7	2	0	16	0	11	??	40	22	JAID	IOTERM0	TERM
23	7	3	0	16	0	11	??	40	23	JAID	IOTERM0	TERM
24	7	4	0	16	0	11	??	40	24	JAID	IOTERM0	TERM
25	7	5	0	16	1	11	??	40	25	JAID	IOTERM0	TERM
26	25	0	0	17	0			0	0		IOINP0	INP1
27	26	0	0	17	3			0	0		IOINP0	INP2
29	28	0	0	18	0			0	0		CSSBSC0	SSLC2
50	#12	0	0	41	0			128	0		IODS0	HDS1
51	#13	0	0	41	0			128	0		IODS0	HDS2
52	#14	0	0	41	0			128	0		IODS0	HDS3
53	#15	0	0	41	0			128	0		IODS0	HDS4
60	#12	0	0	16	0	??	??	40	60	J ID	IODSTRM0	DSTERM
61	#12	0	0	16	0	??	??	40	61	J ID	IODSTRM0	DSTERM
62	#12	0	0	16	0	??	??	40	62	J ID	IODSTRM0	DSTERM
63	#12	0	0	16	0	??	??	40	63	J ID	IODSTRM0	DSTERM
64	#12	0	0	16	0	??	??	40	64	J ID	IODSTRM0	DSTERM
66	#26	0	0	41	1			128	0		IODS0	DSLINE1
67	#27	0	0	41	1			128	0		IODS0	DSLINE2
68	#26	0	0	16	0	??	??	40	68	J ID	IODSTRM0	INP1
69	#27	0	0	16	0	??	??	40	69	J ID	IODSTRM0	INP2
70	#29	0	0	41	0		??	128	0		IODS0	SDS1
71	#29	0	0	16	0	??	??	36	71	J ID	IODSTRM0	DSTERM

Figure 2-3. Sample I/O Device Table (Hardwired Line with HSI)

In Figure 2-4, the shaded items in the column labeled DRIVER NAME show an INP (IOINP0) configured into the system as logical device 27. (Note that the subtype is 3, indicating hardwired.) For this line, there is a DS/3000 communications driver (IODS0) also configured into the system. The IODS0 entry is related to the proper INP entry by the number specified in the column labeled DRT (the # prefix indicates a back reference to a previously defined logical device number). Logical device 67 is paired with logical device 27. In this example, it is the device class name (DSLINE2) or logical device number (67) of the IODS0 entry that would be used to specify the desired line.

NOTE

Figure 2-4 does not show a line configured for X.25 activity to another HP 3000. The back referencing scheme is identical to this example. However, the following differences should be noted for an X.25 configuration:

- The communications driver is named IODSX.
- The device class name or logical device number of an IODSX entry should not be used in a :DSLINE command (or in other commands, such as DSCOPY). To access the desired remote node, we encourage X.25 Link users to enter a nodename whenever a dsdevice is required. See page 2-16 to determine a node name.

LOG DEV #	DRT #	UNIT	CHAN	TYPE	SUB TYPE	TERMINAL TYPE	TERMINAL SPEED	REC WIDTH	OUTPUT DEV	MODE	DRIVER NAME	DEVICE CLASSES
1	4	0	0	0	6			128	0		IOMDISC1	SPOOL SYSDISK
2	5	0	0	0	3			128	0		IOMDISC0	DISC
5	13	0	0	8	0			40	LP	JA S	IOCRD0	CARD
6	14	0	0	32	2			66	0	S	IOPRT0	LP
7	6	0	0	24	0			128	LP		IOTAPE0	TAPE
8	6	1	0	24	0			128	LP		IOTAPE0	TAPE
9	6	2	0	24	0			128	LP		IOTAPE0	TAPE
10	6	3	0	24	0			128	LP	JA S	IOTAPE0	BATAPE
11	20	0	0	34	0			128	0		IOPTPN0	PTPUNCH
12	16	0	0	19	3			0	0		CSHBSC0	HSI1
13	16	0	0	19	3			0	0		CSHBSC0	HSI2
14	16	0	0	19	3			0	0		CSHBSC0	HSI3
16	16	0	0	19	3			0	0		CSHBSC0	HSI4
20	7	0	0	16	0	10	??	40	20	JAID	IOTERM0	CONSOLE
21	7	1	0	16	0	11	??	40	21	JAID	IOTERM0	TERM
22	7	2	0	16	0	11	??	40	22	JAID	IOTERM0	TERM
23	7	3	0	16	0	11	??	40	23	JAID	IOTERM0	TERM
24	7	4	0	16	0	11	??	40	24	JAID	IOTERM0	TERM
25	7	5	0	16	1	11	??	40	25	JAID	IOTERM0	TERM
26	25	0	0	17	0			0	0		IOINP0	INP1
27	26	0	0	17	3			0	0		IOINP0	INP2
29	28	0	0	18	0			0	0		CSSBSC0	SSLC2
50	#12	0	0	41	0			128	0		IODS0	HDS1
51	#13	0	0	41	0			128	0		IODS0	HDS2
52	#14	0	0	41	0			128	0		IODS0	HDS3
53	#15	0	0	41	0			128	0		IODS0	HDS4
60	#12	0	0	16	0	??	??	40	60	J ID	IODSTRM0	DSTERM
61	#12	0	0	16	0	??	??	40	61	J ID	IODSTRM0	DSTERM
62	#12	0	0	16	0	??	??	40	62	J ID	IODSTRM0	DSTERM
63	#12	0	0	16	0	??	??	40	63	J ID	IODSTRM0	DSTERM
64	#12	0	0	16	0	??	??	40	64	J ID	IODSTRM0	DSTERM
65	#12	0	0	16	0	??	??	40	65	J ID	IODSTRM0	DSTERM
66	#26	0	0	41	1			128	0		IODS0	DSLINE1
67	#27	0	0	41	1			128	0		IODS0	DSLINE2
68	#26	0	0	16	0	??	??	40	68	J ID	IODSTRM0	INP1
69	#27	0	0	16	0	??	??	40	69	J ID	IODSTRM0	INP2
70	#29	0	0	41	0		??	128	0		IODS0	SDS1
71	#29	0	0	16	0	??	??	36	71	J ID	IODSTRM0	DSTERM

Figure 2-4. Sample I/O Device Table (Hardwired Line with INP)

In Figure 2-5, the shaded items in the column labeled DRIVER NAME show one SSLC (CSSBSC0) configured into the system as logical device 29. (Note that the subtype is 0, indicating a switched telephone line.) Notice the DS/3000 communications driver, IODS0, is related to the SSLC entry by the number specified in the column labeled DRT (the # prefix indicates a back reference to a previously defined logical device number). Logical device 70 is paired with logical device 29. It is the device class name (SDS1) or logical device number (70) of the IODS0 entry that would be used to specify the desired line.

LOG DEV #	DRT #	UNIT	CHAN	TYPE	SUB TYPE	TERMINAL TYPE	TERMINAL SPEED	REC WIDTH	OUTPUT DEV	MODE	DRIVER NAME	DEVICE CLASSES
1	4	0	0	0	6			128	0		IOMDISC1	SPOOL SYSDISK
2	5	0	0	0	3			128	0		IOMDISC0	DISC
5	13	0	0	8	0			40	LP	JA S	IOCRD0	CARD
6	14	0	0	32	2			66	0	S	IOPRT0	LP
7	6	0	0	24	0			128	LP		IOTAPE0	TAPE
8	6	1	0	24	0			128	LP		IOTAPE0	TAPE
9	6	2	0	24	0			128	LP		IOTAPE0	TAPE
10	6	3	0	24	0			128	LP	JA S	IOTAPE0	BATAPE
11	20	0	0	34	0			128	0		IOPTPN0	PTPUNCH
12	16	0	0	19	3			0	0		CSHBSC0	HSI1
13	16	0	0	19	3			0	0		CSHBSC0	HSI2
14	16	0	0	19	3			0	0		CSHBSC0	HSI3
16	16	0	0	19	3			0	0		CSHBSC0	HSI4
20	7	0	0	16	0	10	??	40	20	JAID	IOTERM0	CONSOLE
21	7	1	0	16	0	11	??	40	21	JAID	IOTERM0	TERM
22	7	2	0	16	0	11	??	40	22	JAID	IOTERM0	TERM
23	7	3	0	16	0	11	??	40	23	JAID	IOTERM0	TERM
24	7	4	0	16	0	11	??	40	24	JAID	IOTERM0	TERM
25	7	5	0	16	1	11	??	40	25	JAID	IOTERM0	TERM
26	25	0	0	17	0			0	0		IOINP0	INP1
27	26	0	0	17	3			0	0		IOINP0	INP2
29	28	0	0	18	0			0	0		CSSBSC0	SSLC2
50	#12	0	0	41	0			128	0		IODS0	HDS1
51	#13	0	0	41	0			128	0		IODS0	HDS2
52	#14	0	0	41	0			128	0		IODS0	HDS3
53	#15	0	0	41	0			128	0		IODS0	HDS4
60	#12	0	0	16	0	??	??	40	60	J ID	IODSTRM0	DSTERM
61	#12	0	0	16	0	??	??	40	61	J ID	IODSTRM0	DSTERM
62	#12	0	0	16	0	??	??	40	62	J ID	IODSTRM0	DSTERM
63	#12	0	0	16	0	??	??	40	63	J ID	IODSTRM0	DSTERM
64	#12	0	0	16	0	??	??	40	64	J ID	IODSTRM0	DSTERM
65	#12	0	0	16	0	??	??	40	65	J ID	IODSTRM0	DSTERM
66	#26	0	0	41	1			128	0		IODS0	DSLINE1
67	#27	0	0	41	1			128	0		IODS0	DSLINE2
68	#26	0	0	16	0	??	??	40	68	J ID	IODSTRM0	INP1
69	#27	0	0	16	0	??	??	40	69	J ID	IODSTRM0	INP2
70	#29	0	0	41	0		??	128	0		IODS0	SDS1
71	#29	0	0	16	0	??	??	36	71	J ID	IODSTRM0	DSTERM

Figure 2-5. Sample I/O Device Table (Telephone Line with SSLC)

In Figure 2-6, the shaded items in the column labeled DRIVER NAME show an INP (IOINP0) configured into the system as logical device 26. (Note that the subtype is 0. This indicates that the INP is used for a dial-up line.) For this line, there is a DS/3000 communications driver, IODS0, also configured into the system. The IODS0 entry is related to the proper INP entry by the number specified in the column labeled DRT (the # prefix indicates a back reference to a previously defined logical device number). Logical device 66 is paired with logical device 26. It is the device class name (DSLINE1) or logical device number (66) of the IODS0 entry that would be used to specify the desired line.

LOG DEV #	DRT #	UNIT	CHAN	TYPE	SUB TYPE	TERMINAL TYPE	TERMINAL SPEED	REC WIDTH	OUTPUT DEV	MODE	DRIVER NAME	DEVICE CLASSES
1	4	0	0	0	6			128	0		IOMDISC1	SPOOL SYSDISK
2	5	0	0	0	3			128	0		IOMDISC0	DISC
5	13	0	0	8	0			40	LP	JA S	IOCRD0	CARD
6	14	0	0	32	2			66	0	S	IOPRT0	LP
7	6	0	0	24	0			128	LP		IOTAPE0	TAPE
8	6	1	0	24	0			128	LP		IOTAPE0	TAPE
9	6	2	0	24	0			128	LP		IOTAPE0	TAPE
10	6	3	0	24	0			128	LP	JA S	IOTAPE0	BATAPE
11	20	0	0	34	0			128	0		IOPTPN0	PTPUNCH
12	16	0	0	19	3			0	0		CSHBSC0	HSI1
13	16	0	0	19	3			0	0		CSHBSC0	HSI2
14	16	0	0	19	3			0	0		CSHBSC0	HSI3
16	16	0	0	19	3			0	0		CSHBSC0	HSI4
20	7	0	0	16	0	10	??	40	20	JAID	IOTERM0	CONSOLE
21	7	1	0	16	0	11	??	40	21	JAID	IOTERM0	TERM
22	7	2	0	16	0	11	??	40	22	JAID	IOTERM0	TERM
23	7	3	0	16	0	11	??	40	23	JAID	IOTERM0	TERM
24	7	4	0	16	0	11	??	40	24	JAID	IOTERM0	TERM
25	7	5	0	16	1	11	??	40	25	JAID	IOTERM0	TERM
26	25	0	0	17	0			0	0		IOINP0	INP1
27	26	0	0	17	3			0	0		IOINP0	INP2
29	28	0	0	18	0			0	0		CSSBSC0	SSLC2
50	#12	0	0	41	0			128	0		IODS0	HDS1
51	#13	0	0	41	0			128	0		IODS0	HDS2
52	#14	0	0	41	0			128	0		IODS0	HDS3
53	#15	0	0	41	0			128	0		IODS0	HDS4
60	#12	0	0	16	0	??	??	40	60	J ID	IODSTRM0	DSTERM
61	#12	0	0	16	0	??	??	40	61	J ID	IODSTRM0	DSTERM
62	#12	0	0	16	0	??	??	40	62	J ID	IODSTRM0	DSTERM
63	#12	0	0	16	0	??	??	40	63	J ID	IODSTRM0	DSTERM
64	#12	0	0	16	0	??	??	40	64	J ID	IODSTRM0	DSTERM
65	#12	0	0	16	0	??	??	40	65	J ID	IODSTRM0	DSTERM
66	#26	0	0	41	1			128	0		IODS0	DSLINE1
67	#27	0	0	41	1			128	0		IODS0	DSLINE2
68	#26	0	0	16	0	??	??	40	68	J ID	IODSTRM0	INP1
69	#27	0	0	16	0	??	??	40	69	J ID	IODSTRM0	INP2
70	#29	0	0	41	0		??	128	0		IODS0	SDS1
71	#29	0	0	16	0	??	??	36	71	J ID	IODSTRM0	DSTERM

Figure 2-6. Sample I/O Device Table (Telephone Line with INP)

If you have only one communications interface configured into your system, there is no question about which name or number to specify in a :DSLINE command. If there is more than one communications interface, however, you must know (or ask someone who knows) which interface is connected to the physical line you want to use.

Specifying an X.25 Line

When specifying an X.25 line, you do not use the I/O device table. This is because X.25 uses node names, rather than LDEVs, to specify devices. To find out what node names are configured on your system, type the following:

```
:RUN NETCONF.PUB.SYS

LIST
```

See the *DS/3000 HP 3000 to HP 3000 Network Administrator Manual* for more information on NETCONF.

The DSLINE Command

The format of the :DSLINE command, as used to open a line, is presented here. In addition to opening a hardwired line or a telephone line, this command can also be used for closing one or more communications lines (see page 2-50).

SYNTAX

```
:DSLINE dsdevice  [;LINEBUF=buffersize]
                  [;EXCLUSIVE]
                  [;COMP]
                  [;NOCOMP]
                  [;QUIET]
                  [;PHNUM=telephonenumber]
                  [;LOCID=localidsequence]
                  [;REMID=remoteidsequence[,...remoteidsequence]]
                  [;SELECT=selectionsignal sequence]
                  [;OPEN]
                  [;QUEUE]
                  [;NOQUEUE]
```

PARAMETERS

The parameters that pertain to opening a line follow:

dsdevice

This is the device class name or logical device number assigned to the DS/3000 communications driver (IODS0) during system configuration, or a logical node name associated with the X.25 Link communications driver (IODSX) during network configuration. This parameter indirectly specifies which communications interface you wish to use.

For X.21, you can specify a node rather than a DS device. DS will first determine if a valid LDEV or device class name was given. If that is not the case, DS will search the configuration data base to determine if it is a node name.

NOTE

X.25 Link users should always specify a node name rather than a line identifier. The logical node name (mentioned in the *dsdevice* parameter description) appears in the configuration file, NETCON, for a Public Data Network (PDN). A Remote Node (RN) table entry relates the logical node name (specified in this command)

to the logical device number of the appropriate IODSX driver (the X.25 driver), and to the PDN address of the destination node.

buffersize A decimal integer specifying the size (in words) of the DS/3000 line buffer to be used in conjunction with the communications line. The integer must be within the range 304 < *buffer-size* < 4095 when used with the SSLC or HSI, or within the range 304 < *buffer-size* < 1024 when used with the INP. The default value is the buffer size entered in response to the PREFERRED BUFFER SIZE prompt during system configuration. This parameter overrides the MPE configured value when specified by the first user to open the given line. If you are using X.25, this parameter is ignored.

EXCLUSIVE This parameter, if present, specifies that you want exclusive use of the communications line. If the requested line or specified communications interface is already open and you have specified the EXCLUSIVE option, DS/3000 will deny you access to the line (you cannot open it). (See "Line Opening Failures" later in this section.) Opening an EXCLUSIVE line requires the user to have CS and ND capabilities.

COMP By using this parameter, you can override the current system default, which was set at configuration time (see Section 1 of the *DS/3000 Network Administrator Manual*) or set by the system operator (see Section 9), and activate data compression. In this way, the mode of operation is set for your subsequent DS activity. This parameter does not affect other users sharing the line.

NOCOMP This parameter deactivates the data compression mode.

QUIET When you issue the :DSLINE command with this parameter added, the message identifying the DS line number is suppressed. The messages associated with the subsequent REMOTE HELLO and REMOTE BYE commands will also be suppressed.

CAUTION

The messages suppressed when the QUIET parameter is used may include information you would normally want to see or expect to see. For instance, if the remote account requires a password, and you do not include the password as part of the REMOTE HELLO command, the remote system's prompt for that password will not be displayed. Since the REMOTE HELLO cannot finish until you supply that password, the terminal will appear to be "hung" when, in fact, it is only waiting for your response. Similarly, if the remote account has a logon UDC, any prompt produced by a command within that UDC will not appear. Again, until the unseen prompt is answered, the :REMOTE HELLO cannot complete, and the system will not respond with the appropriate remote or local system prompt (# or :).

The additional parameters that pertain only to opening a telephone line are as follows:

telephonenumber

A telephone number consisting of digits, dashes, and special characters. The permitted special characters are:

0 through 9

/ (separator used for automatic call units that have a second dial tone detect)

D (one-second delay. Used for European modems and automatic call units that require built-in delays)

(defined by the local telephone system)

* (defined by the local telephone system)

The maximum length permitted (including both digits and dashes) is 30 characters. Provided that YES was entered in response to the DIAL FACILITY prompt during system configuration, this telephone number will be displayed at the operator's console of your HP 3000 and the operator will then establish the telephone connection by dialing that number at the modem. (When the autodial feature is present in your system, the number provided here is dialed automatically.) The default telephone number is the one entered in response to the PHONE NUMBER prompt during system configuration.

localidsequence

A string of ASCII characters contained within quotation marks. If you wish to use a quotation mark within an ASCII string, use two successive quotation marks. The maximum number of ASCII characters allowed in the string is 16.

The supplied string of ASCII characters defines the ID sequence that will be sent from your HP 3000 to the remote HP 3000 when you attempt to establish the telephone connection. The default value is the ASCII string entered in response to the LOCAL ID SEQUENCE prompt during system configuration.

remoteidsequence

Same format as *localidsequence*.

The supplied strings of ASCII characters define those remote HP 3000 ID sequences that will be considered valid during an attempt to establish the telephone connection. If the remote HP 3000 does not send a valid ID sequence, the telephone connection is terminated. The default set of remote ID sequences consists of the ASCII strings entered in response to the REMOTE ID SEQUENCE prompt during system configuration.

SELECT

The SELECT parameter is the equivalent of PHNUM for X.21 switched lines. The *selectionsignalsequence*, SELECT's equivalent of a phone number, must be a string of 1 to 30 alphanumeric and/or special characters.

It must be delimited between two identical special characters. If the specified *selectionsignalsequence* is less than 30 characters, it will be padded at the end with blanks. The *selectionsignalsequence* can be given as one or more blanks to indicate you wish to use the direct dial facility. (You must subscribe to it through your X.21 network.) The SELECT parameter value takes precedence over that specified in the configuration data base.

QUEUE

This parameter allows your request to be queued when the X.21 network connection you wish to open is already in use. Your call will be connected as soon as the circuit is free. If another user is already queued, and you issue a :DSLINE command that allows queueing, a message will be printed telling you that you are queued behind a previous request. The default condition is NOQUEUE.

NOQUEUE

This parameter deactivates queueing when you are opening an X.21 line. If the line is already in use when your :DSLINE is issued, your call request will not be completed when the ciruit becomes available. This is the default condition.

Dialing the Remote Computer

When you are opening a telephone line, you may supply a telephone number as an optional parameter in the :DSLINE command to be dialed at the modem connected to the specified interface. If you supply a telephone number, DS/3000 displays a message on the system console telling the operator to dial that number. The operator, after dialing the specified number, enters YES or NO through the system console =REPLY command to let DS/3000 know whether or not the telephone connection was successfully made. If the operator enters YES, DS/3000 proceeds with the exchanging of ID sequences. If the operator enters NO, your :DSLINE request is denied (you cannot open the line). In either case, your terminal's keyboard is disabled until the console operator responds.

If you do not supply a telephone number, the sequence of events is as described in the above paragraph, except that DS/3000 uses (by default) the first telephone number in the PHONELIST established during system configuration.

If you do not supply a telephone number and no PHONELIST was established during system configuration, an I/O request message is displayed at the system console, but it does not include the number to be dialed. This method might be used when you will dial the remote HP 3000 yourself. Remember, however, that the console operator must still know whether you dialed successfully, since he must respond to the console message before you are granted access to the line. Because your terminal's keyboard is disabled until the console operator responds with YES or NO, it is recommended that you always supply a telephone number in the :DSLINE command.

ID Sequences

Once a telephone connection to a remote HP 3000 exists, the two computers exchange ID sequences with one another. Within the context of DS/3000, an ID sequence is a string of up to 16 ASCII or EBCDIC characters, or octal or hexadecimal numbers, that identifies a particular computer.

During system configuration, each HP 3000 can be assigned a local ID sequence and a list of remote ID sequences. The local ID sequence identifies the particular HP 3000 in which it is established; the remote ID sequences identify those remote computers with which a communications link can be established over the public telephone network.

In the :DSLINE command, you can supply (as optional parameters) a local ID sequence and one or more remote ID sequences to be used instead of those established during system configuration. (See Figure 2-9.)

When a telephone connection is established between your HP 3000 and a remote HP 3000, the local ID sequence supplied in your :DSLINE command is transmitted to the remote system. Then the remote system transmits its local ID sequence over the telephone line to your HP 3000. The received ID sequence is then compared against the remote ID sequence(s) supplied in your :DSLINE command. If that ID sequence is found to be valid, the telephone connection is considered successful and DS/3000 grants you access to the line. If the ID sequence received at either end of the line is not considered valid, your :DSLINE request is denied (you cannot open the line).

If you do not supply any ID sequences, DS/3000 uses those established during system configuration. If no ID sequences were established during system configuration and you do not supply any, no local ID sequence is transmitted from your HP 3000 to the remote system and any remote ID sequence received is considered valid.

Multiple Users

Within a DS/3000 environment, it is possible for several users at either end of the line to share access to the same physical communications line or for a single user at one end of the line to obtain exclusive access to the line.

As previously mentioned in the presentation of the :DSLINE command, the EXCLUSIVE parameter can be used to obtain exclusive access to the specified physical communications line. If you specify this parameter (and if access to the line is granted), no other user in either computer will be permitted to open that line until you close it. If you ask for exclusive access to a particular line and that line is already in use, DS/3000 denies your request (you cannot open the line). (See "Line Opening Failures" later in this section.)

For hardwired lines and for dial-up lines, multiple users at either end of the line can specify the same physical line in :DSLINE commands and obtain access to that line as long as none of them requests exclusive access. In such a case, the users' data is multiplexed, so that each user's access to the line appears to be completely independent of all others. The exception for a telephone line is that all users, other than the one who originally opened the line, specify (explicitly or by default) the currently active remote ID sequence. Figures 2-7 through 2-11 present annotated examples, illustrating successful and unsuccessful attempts by different users to obtain access to the same line.

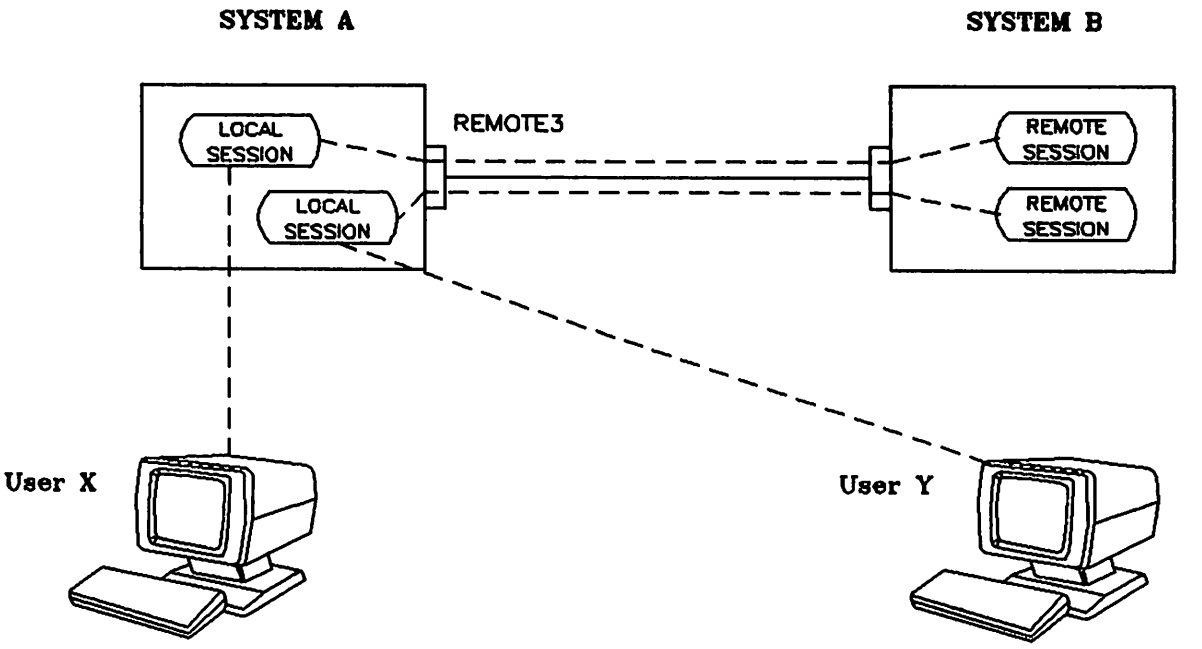


Figure 2-7. Multiple User Example 1

```
:HELLO USER.X                    :HELLO USER.Y

:DSLINE REMOTE3                  :DSLINE REMOTE3

:REMOTE HELLO USER.X             :REMOTE HELLO USER.Y
```

In this example, User X initiates a local session in System A, obtains access to the hardwired communications line that connects System A to System B, and initiates a remote session in System B. User Y subsequently initiates a local session in System A, obtains access to the same communications line, and initiates a remote session in System B. The request by User Y for the particular communications line is granted by DS/3000 because neither user asked for exclusive access to the line.

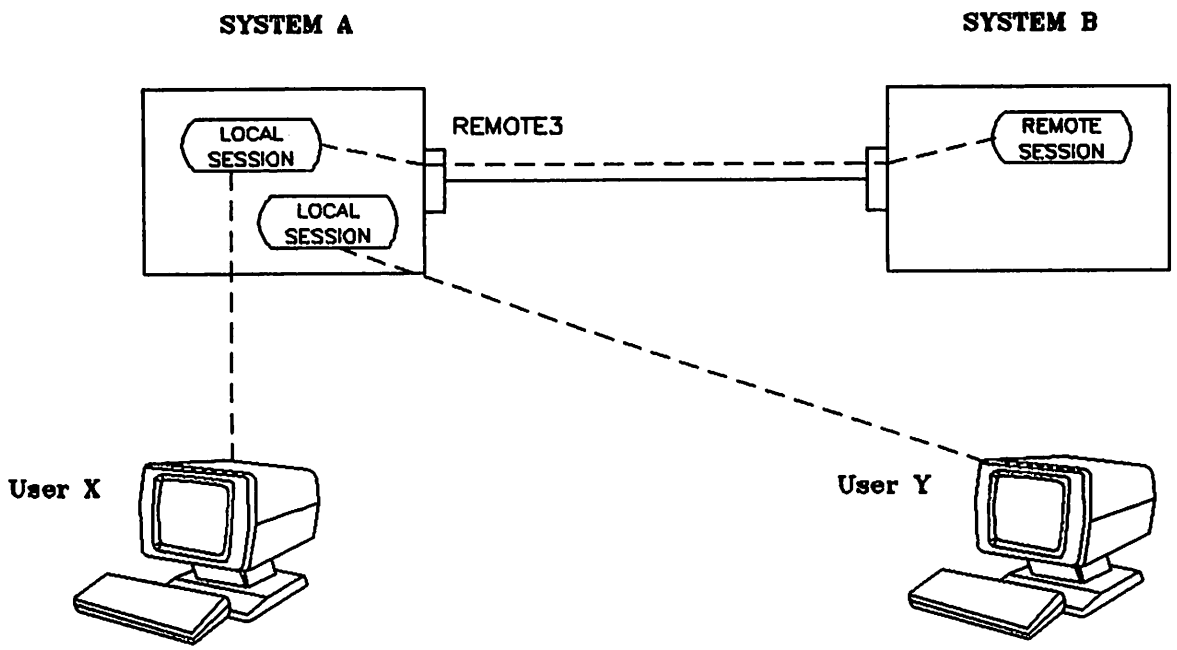


Figure 2-8. Exclusive Option Example

```
:HELLO USER.X                    :HELLO USER.Y

:DSLINE REMOTE3 ;EXCLUSIVE       :DSLINE REMOTE3

:REMOTE HELLO USER.X
```

In this example, User X initiates a local session in System A, obtains exclusive access to the hardwired communications line that connects System A to System B, and initiates a remote session in System B. User Y subsequently initiates a local session in System A and requests access to the same communications line. The request is denied by DS/3000 because User X already has exclusive access to the specified line. DS/3000 responds with:

```
241 DS LINE IN USE EXCLUSIVELY OR BY ANOTHER SUBSYSTEM.  (DSERR 241)
```

Configured Local ID: A

Configured Remote IDs: B,C

Configured Local ID: B

Configured Remote IDs: A,C

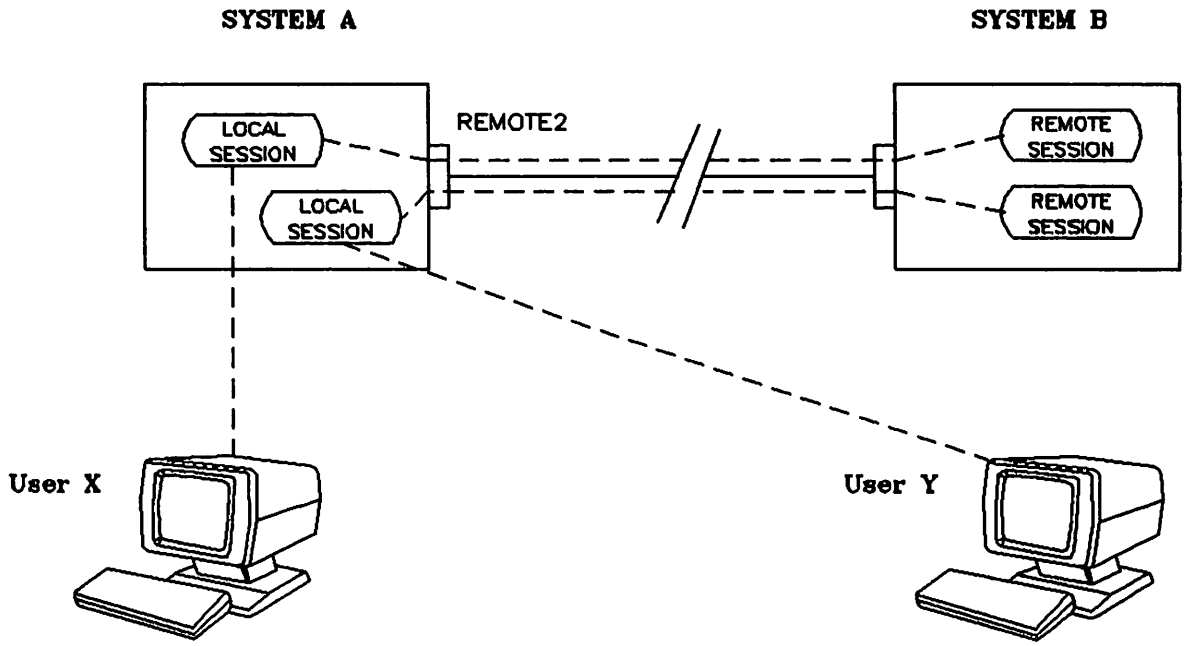


Figure 2-9. Dial-up Line Multiple User Example 1

```
:HELLO USER.X

:DSLINE REMOTE2 &
        ;PHNUM=555-1234 &
        ;REMID="B"

:REMOTE HELLO USER.X
```

```
:HELLO USER.Y

:DSLINE REMOTE2 &
        ;PHNUM=555-1234 &
        ;REMID="B"

:REMOTE HELLO USER.Y
```

In this example User X initiates a local session in System A and obtains access to the line identified by the device class name REMOTE2. The supplied telephone number is displayed at the system console of System A. The console operator establishes the telephone connection by dialing the number at the modem connected to the particular line and then enters YES through the system console to let DS/3000 know that the telephone connection was successfully made. The two computers exchange their configured local ID sequences. System A compares the received ID sequence (B) against the remote ID sequence specified by User X (REMID="B"). Since the received ID sequence is found to be valid, the telephone connection is allowed to remain in effect. User X then initiates a remote session in System B over the telephone line from his local log-on terminal.

User Y subsequently initiates a local session in System A and requests access to the same line (REMOTE2). Since that line is already open, DS/3000 ignores the supplied telephone number (no message is displayed at the system console). Access to the currently opened line is granted to User Y because neither user requested exclusive access and User Y specified the currently active remote ID sequence (REMID="B") in the :DSLINE command.

Configured Local ID: A

Configured Remote IDs: B,C

Configured Local ID: B

Configured Remote IDs: A,C

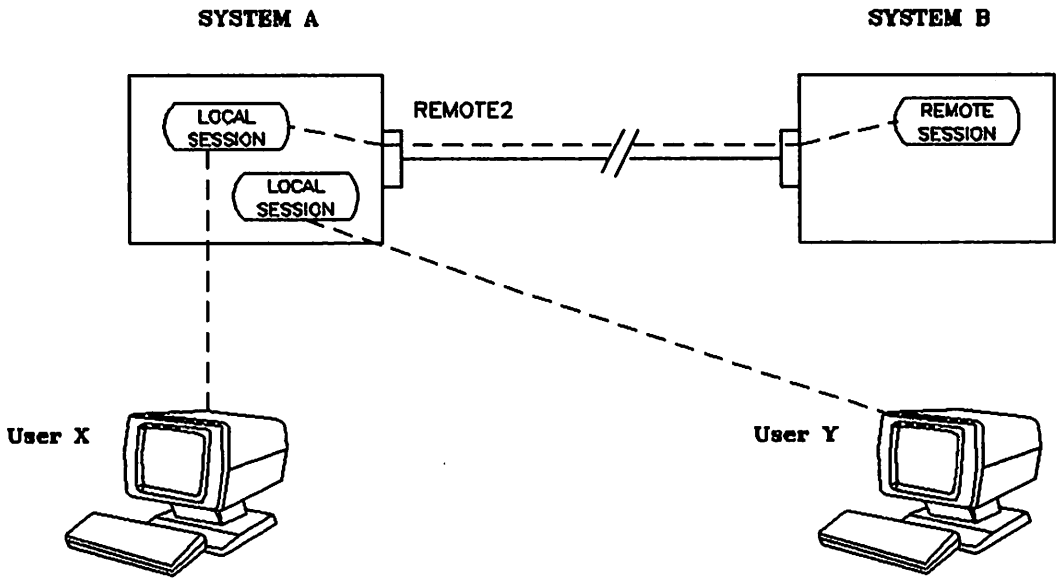


Figure 2-10. Dial-up Line Multiple User Example 2

```
:HELLO USER.X

:DSLINE REMOTE2 &
        ;PHNUM=555-1234 &
        ;REMID="B"

:REMOTE HELLO USER.X
```

```
:HELLO USER.Y

:DSLINE REMOTE2 &
        ;PHNUM=555-2001 &
        ;REMID="C"
```

In this example User X initiates a local session in System A and obtains access to the line identified by the device class name REMOTE2. The supplied telephone number is displayed at the system console of System A. The console operator establishes the telephone connection by dialing the number at the modem connected to the particular line and then enters YES through the system console to let DS/3000 know that the telephone connection was successfully made. The two computers exchange their configured local ID sequences. System A compares the received ID sequence (B) against the remote ID sequence specified by User X (REMID="B"). Since the received ID sequence is found to be valid, the telephone connection is allowed to remain in effect. User X then initiates a remote session in System B over the telephone line from his local log-on terminal.

User Y subsequently initiates a local session in System A and requests access to the same line (REMOTE2). Since that line is already open, DS/3000 ignores the supplied telephone number, and no message is displayed at the system console. The request is denied by DS/3000 because the specified line is already open and User Y did not specify the currently active remote ID sequence (B) in his :DSLINE command. DS/3000 responds with:

```
255 COMMUNICATIONS INTERFACE ERROR.  UNANTICIPATED CONDITION.  (DSERR 255)
```

Configured Local ID: (none)

Configured Remote IDs: (none)

Configured Local ID: (none)

Configured Remote IDs: (none)

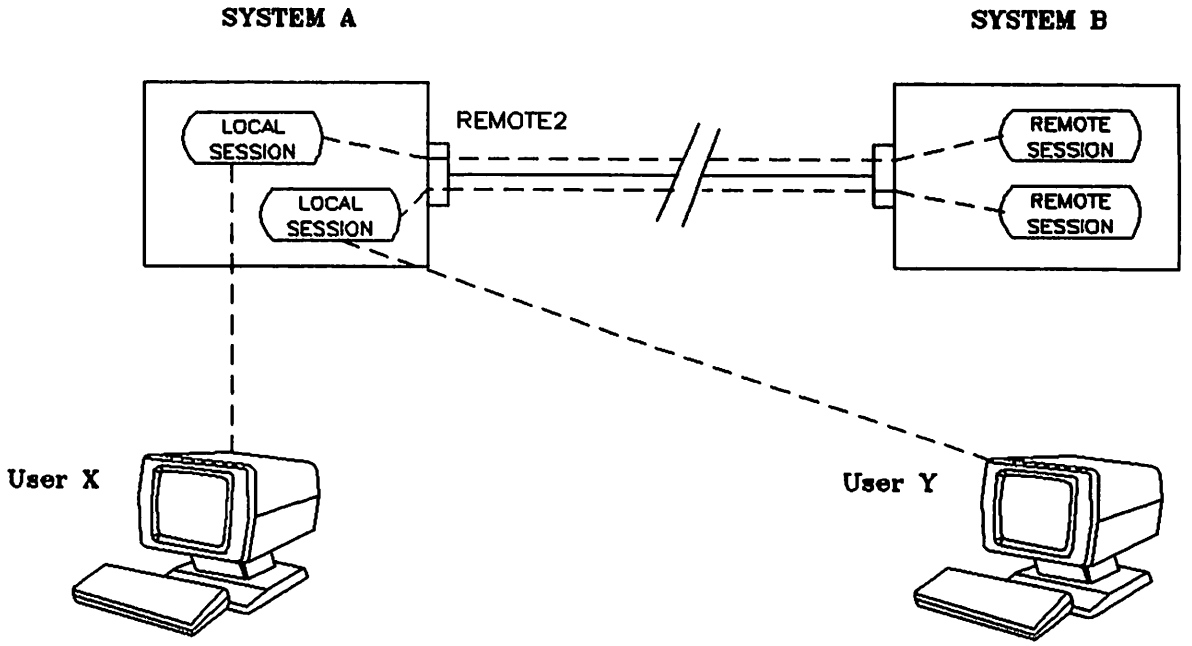


Figure 2-11. Dial-up Line Multiple User Example 3

```
:HELLO USER.X
:DSLINE REMOTE2 ;PHNUM=555-1234
:REMOTE HELLO USER.X
```

```
:HELLO USER.Y
:DSLINE REMOTE2 ;PHNUM=555-1234
:REMOTE HELLO USER.Y
```

In this example User X initiates a local session in System A and obtains access to the line identified by the device class name REMOTE2. The supplied telephone number is displayed at the system console of System A. The console operator establishes the telephone connection by dialing the number at the modem connected to the particular line and then enters YES through the system console to let DS/3000 know that the telephone connection was successfully made. No ID sequences are exchanged because none were established (in either HP 3000) during system configuration and User X didn't specify any in the :DSLINE command. User X then initiates a remote session in System B over the telephone line from his local log-on terminal.

User Y subsequently initiates a local session in System A and requests access to the same line (REMOTE2). Since that line is already open, DS/3000 ignores the supplied telephone number (no message is displayed at the system console). Access to the currently opened line is granted to User Y because neither user requested exclusive access and User Y specified the currently active remote ID sequence (in this case none) in his :DSLINE command.

NOTE

When no ID sequences are configured and the users don't supply any in their :DSLINE commands, both are taking it on faith that they are connected to the proper remote computer. In this example, if User Y had specified PHNUM=555-2001, DS/3000 would have ignored this telephone number because the line is already open. User Y would have been connected to the currently active remote computer rather than the requested remote system. The total absence of configured or supplied ID sequences is safe only under very controlled circumstances. It is strongly recommended that all computers in a DS/3000 network that are capable of communicating over telephone lines have default local and remote ID sequences established during system configuration and that all line users specify the ID sequence of the desired remote computer (REMID=x) in their :DSLINE commands.

The REMOTE HELLO Command

Once you have obtained access to a physical communications line using the :DSLINE command, you use the :REMOTE HELLO command to actually establish the communications link. The :REMOTE HELLO command initiates a remote session on your behalf in the HP 3000 connected to the other end of the communications line.

The format of the :REMOTE HELLO command is presented here. Notice that, except for the two shaded items, it has exactly the same format as the standard MPE :HELLO command.

Because the :REMOTE HELLO command is initiating a session for you in a remote HP 3000, the parameters in that command specify information which pertains to the operating environment of the remote HP 3000 instead of the local one. For instance, *username*, *accountname*, and *groupname* must all be valid as defined by the accounting structure of the **remote** HP3000, and the *cpuseconds* specified by the optional TIME parameter impose a time limit on the remote, rather than the local, session.

SYNTAX

```
:REMOTE HELLO [sessionname,]username[/userpass].acctname[/acctpass]
                [,groupname[/grouppass]]

              [;TERM=termtype]

              [;TIME=cpusecs]

              [;PRI={BS|CS|DS|ES}]

              [;INPRI=inputpriority]
              [;HIPRI               ]

              [;DSLINE=dsdevice]
```

PARAMETERS

sessionname

Arbitrary name used in conjunction with *username* and *acctname* parameters to form a session identity. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Default is that no session name is assigned.

username

A user name, established by the account manager, that allows you to log on under this account. Contains from 1 to 8 alphanumeric characters, beginning with a letter.

(Required parameter.)

userpass

User password, optionally assigned by the account manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter.

acctname

Name of account, as established by the account manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter. The *acctname* parameter must be preceded by a period.

(Required parameter.)

acctpass

Account password, optionally assigned by the system manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter.

groupname

Name of the group to be used for local file domain and CPU time charges. Established by the account manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter. Default is your home group, if you are assigned one by the account manager.

(Optional if you have a home group; required if a home group has not been assigned.)

grouppass

Group password, optionally assigned by the account manager. Contains from 1 to 8 alphanumeric characters, beginning with a letter.

(Required if assigned and you are logging on under other than your home group; optional if you are logging on under your home group.)

termtype

Ignored. The TERM=*termtype* parameter of the HELLO command that initiated the local session also implicitly defines the log-on terminal type for any remote sessions initiated from the local session.

cpusecs

Maximum CPU time that the session can use, entered in seconds. When the limit is reached, the session is aborted. Must be a value from 1 to 32767. To specify no limit, enter a question mark (?) or UNLIM, or omit the parameter. Default is no limit.

BS, CS, DS, ES

Execution priority class. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest permitted priority for your account or user name by the system, MPE assigns the highest priority possible below BS. Default is CS.

NOTE

DS and ES are used primarily for batch jobs. Their use for sessions is discouraged.

inputpriority

Relative input priority used in checking against access restrictions imposed by the jobfence, if one exists. Takes effect at log-on time. Must be a value from 1 (lowest priority) to 13 (highest priority). If a value is specified that is less than or equal to current jobfence set by the console operator, the session is denied access. Default is 8 or 13, depending upon the System Logging options in effect.

HIPRI

Request for maximum session-selection input priority, causing the session to be scheduled regardless of current jobfence or execution limit for sessions. This parameter can be specified only by users with System Manager or System Supervisor capability. (If not, the system tries to log you on with INPRI=13.) Default is the current jobfence and execution limit.

dsdevice

The device class name or logical device number assigned to the DS/3000 communications driver (IODS0) during system configuration, or a logical node name associated with the X.25 Link communications driver (IODSX) during network configuration using NETCONF. This parameter, if present, specifies which line you wish to use.

(Optional parameter if a line is already open; otherwise it is required.)

NOTE

X.25 Link users should always use a node name rather than a line identifier.

So far, we have been talking entirely about the :DSLINE and :REMOTE HELLO commands being used in conjunction with one another: the :DSLINE command obtaining access to a physical line and the :REMOTE HELLO command actually establishing the communications link by initiating a remote session over the acquired line. As you may have guessed from the above parameter definitions, the :DSLINE parameter of the :REMOTE HELLO command gives you a new, and simpler, way to obtain a line and establish a communications link. If you are satisfied to use the default DS/3000 line buffer size and you do not need exclusive use of the line, you can acquire a line and initiate a remote session over that line by using a single command: a :REMOTE HELLO command with the :DSLINE parameter. If you open a line in this way, however, it remains open only for the duration of the particular remote session (when the remote session is terminated the line is automatically closed). If, on the other hand, you use the :DSLINE command to open a line, the line remains open for the duration of the local session (or until you explicitly close the line).

To illustrate this, look again at the example in Section 1. In that example, the :DSLINE command was used to obtain access to the hardwired line REMOTE1 and the :REMOTE HELLO command was used to initiate a remote session over the line:

```
:DSLINE REMOTE1
DS LINE NUMBER = #L3
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 9:08 AM

WELCOME TO SYSTEM B.
:
```

NOTE

In this case, the acquired line remains open when the remote session is terminated.

By including the :DSLINE parameter in the :REMOTE HELLO command, essentially the same operations could be performed while using a single command, as follows:

```
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=REMOTE1
DS LINE NUMBER = #L3

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 9:08 AM

WELCOME TO SYSTEM B.
:
```

NOTE

In this case, the acquired line is closed when the remote session is terminated.

These same examples can be followed when you use a dial-up line. However, the telephone line will

work properly for you only under the following very limited circumstances:

- You must be satisfied to use the default DS/3000 line buffer size established during system configuration.
- The default ID sequences established in both computers during system configuration must properly identify both your local HP 3000 and the desired remote HP 3000 (or no ID sequences were established during system configuration in either computer).
- You must dial the remote computer yourself at the local system modem, or the line must be connected to, and configured for, autodialing. Note that if you cannot successfully make the telephone connection you cannot abort the :REMOTE HELLO command; the command will be rejected by DS/3000 if no connection is established within 15 minutes, unless another value was specified at configuration time.

The likelihood of all of the above conditions existing for a particular use of DS/3000 is rather slim. In most DS/3000 environments you will want to explicitly define the ID sequence of the desired remote computer to guarantee that the proper connection is established, and you will want to provide a telephone number so that you can let DS/3000 know immediately if a telephone connection cannot be made. (It is not acceptable to tie up a communications interface and your log-on terminal waiting for an unsuccessful :DSLINE or :REMOTE HELLO request to be rejected.)

CAUTION

Special consideration must be paid to the contents of any remote logon UDCs activated by a :REMOTE HELLO command. These UDCs are commonly constructed to execute a series of batch processes or initiate a turnkey application. In such cases, the command :BYE is often the final command in the sequence. The execution of this :BYE in a remote logon UDC will generate the following error messages:

```
REMOTE HELLO Must Be Done To Initiate Remote
   Session.  (DSERR 227)
UNABLE TO COMPLETE THE REMOTE COMMAND.  (CIERR 1316)
```

These messages are generated because the :REMOTE HELLO does not actually complete until it finishes executing the commands within the UDC file. The final :BYE, however, terminates the remote session. So when the remote system replies to the :REMOTE HELLO, DS/3000 finds that no remote session has been established and returns an error.

It is not safe to assume, however, that the error message has been generated solely because of the :BYE command in the remote logon UDC, since the same message will be generated if the :REMOTE HELLO fails for some other reason, or if the execution of a UDC fails on an internal error. When this error message appears, then, it may be difficult to interpret the actual cause of the error. Accordingly, it is recommended that the user avoid inserting the :BYE command in the remote account's logon UDC, and find an alternative way of accomplishing the particular batch or application processing. One method would be to execute a job stream from within the local session.

Opening Multiple Lines

Within your local session, you can open more than one physical communications line and you can have remote sessions active concurrently over all of the opened lines. However, when operating without X.25 Link, you are limited to one remote session per physical line at any given time.

If access to the specified line is obtained, DS/3000 responds to each :DSLINE command by displaying a DS line number at your log-on terminal. This line number is roughly analogous to the file number returned by the MPE FOPEN intrinsic, in that it is an arbitrary number that uniquely identifies (within your local session) your current access to a particular communications line. It has no relationship to the logical device number or any other configuration parameter associated with the line. DS line numbers are meaningful only if you have more than one line open concurrently within a single local session. In that case, you are assigned a separate DS line number for each line you have opened, and you subsequently use these numbers to specify which line you wish to use for a given remote command (or sequence of remote commands) or to close a particular line without closing the others.

Figure 2-12 illustrates a situation where a user has established two hardwired communications links concurrently from within a single local session. Take a closer look at that situation and examine the sequence of commands that was used to create it.

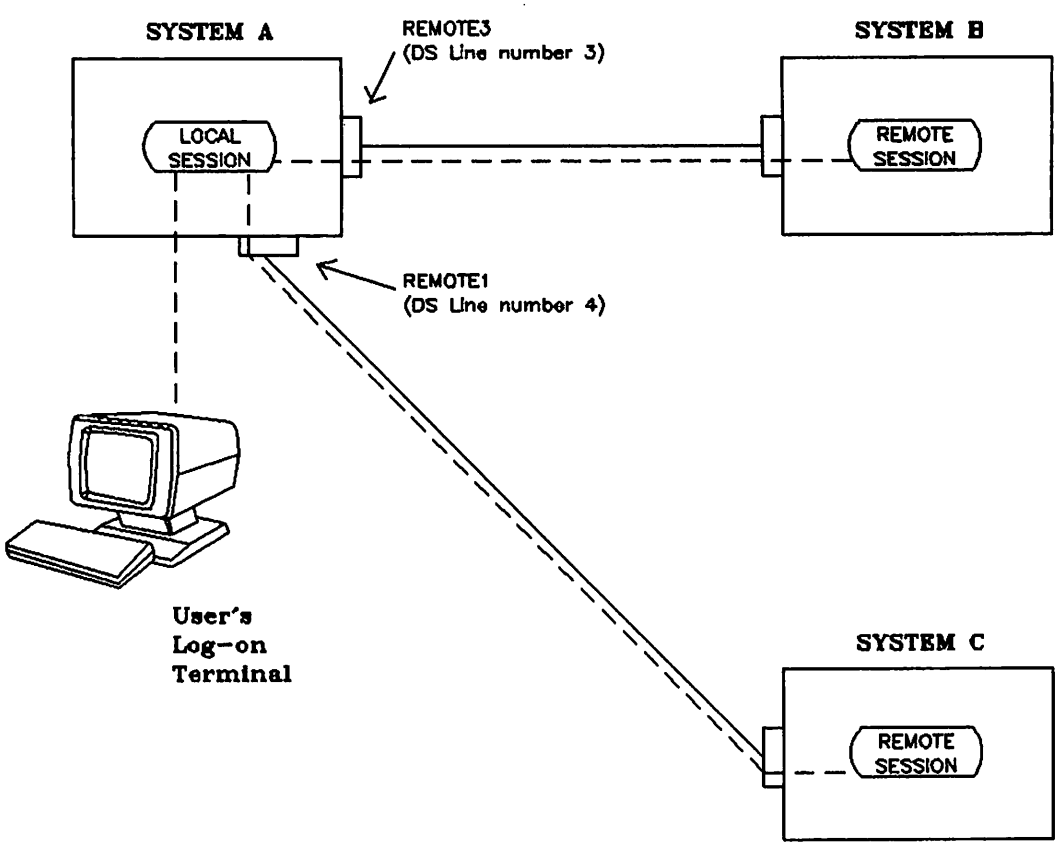


Figure 2-12. Multiple Line Example (Hardwired Lines)

First the user sat down at a terminal connected to System A and initiated a local session:

```
:HELLO USER.ACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:37 PM

WELCOME TO SYSTEM A.
:
```

USER and ACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System A.

Now, we have the situation illustrated in Figure 2-13. Notice that, at this point, no communications link exists between any of the three systems.

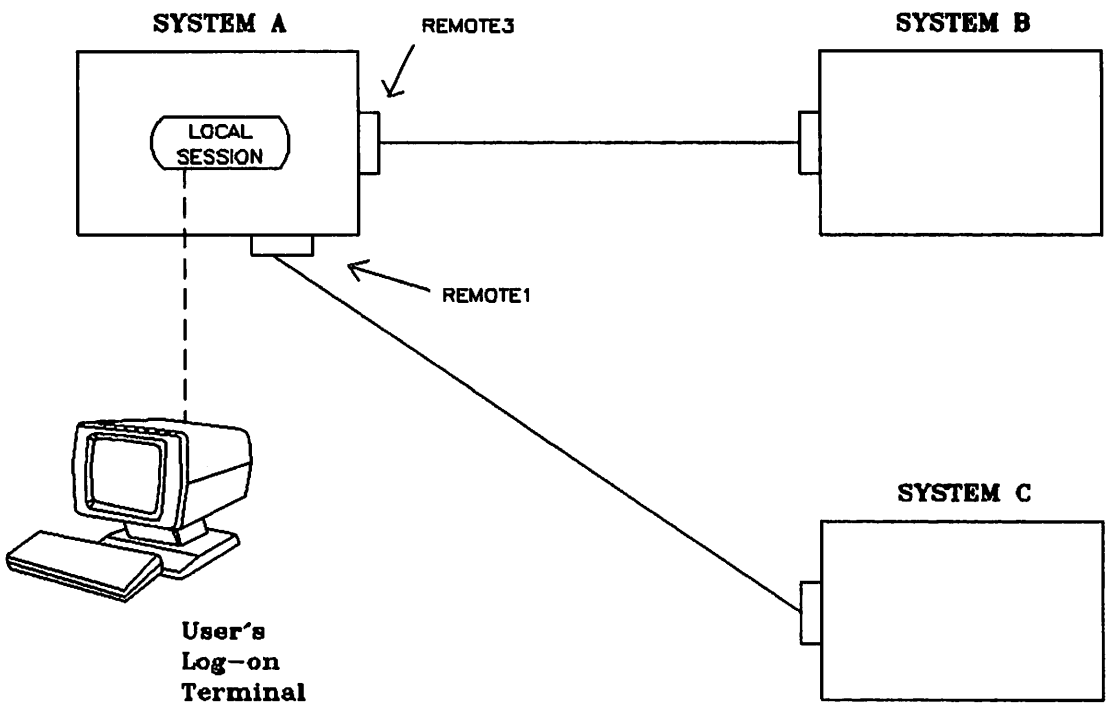


Figure 2-13. Initiating the Local Session (Hardwired Example)

Next, the user acquired access to a line between Systems A and B and initiated a remote session in System B:

```
:DSLINE REMOTE3
DSLINE NUMBER = #L3
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:38 PM

WELCOME TO SYSTEM B.
:
```

REMOTE3 is the device class name (as defined within System A) associated with the particular line. RUSER and RACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System B.

Now we have the situation illustrated in Figure 2-14.

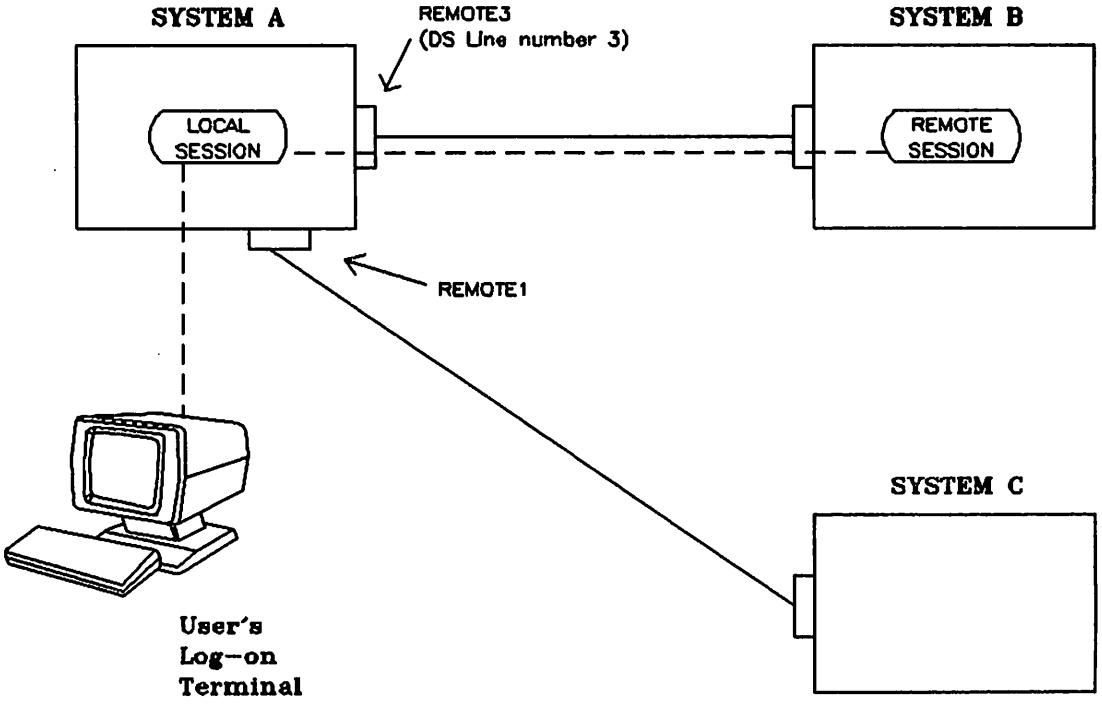


Figure 2-14. Establishing the Link With System B (Hardwired Example)

Finally, the user acquired access to a line between Systems A and C and initiated a remote session in System C:

```
:DSLINE REMOTE1
DS LINE NUMBER = #L4
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:39 PM

WELCOME TO SYSTEM C
:
```

REMOTE1 is the device class name (as defined within System A) associated with the particular line. RUSER and RACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System C.

We end up with the situation illustrated in Figure 2-15, which is identical to Figure 2-12 that started this example.

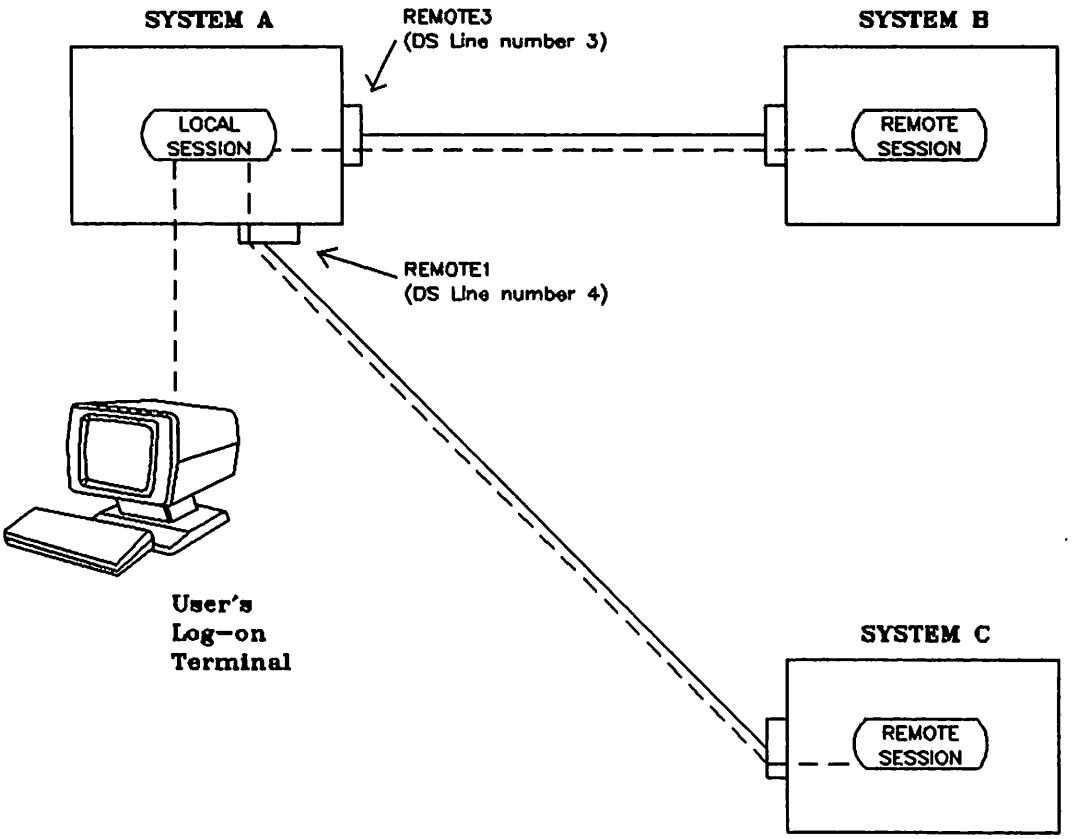


Figure 2-15. Establishing the Link With System C (Hardwired Example)

Figure 2-16 illustrates a situation where a user has established two telephone communications links concurrently from within a single local session. Take a closer look at that situation and examine the sequence of commands that was used to create it.

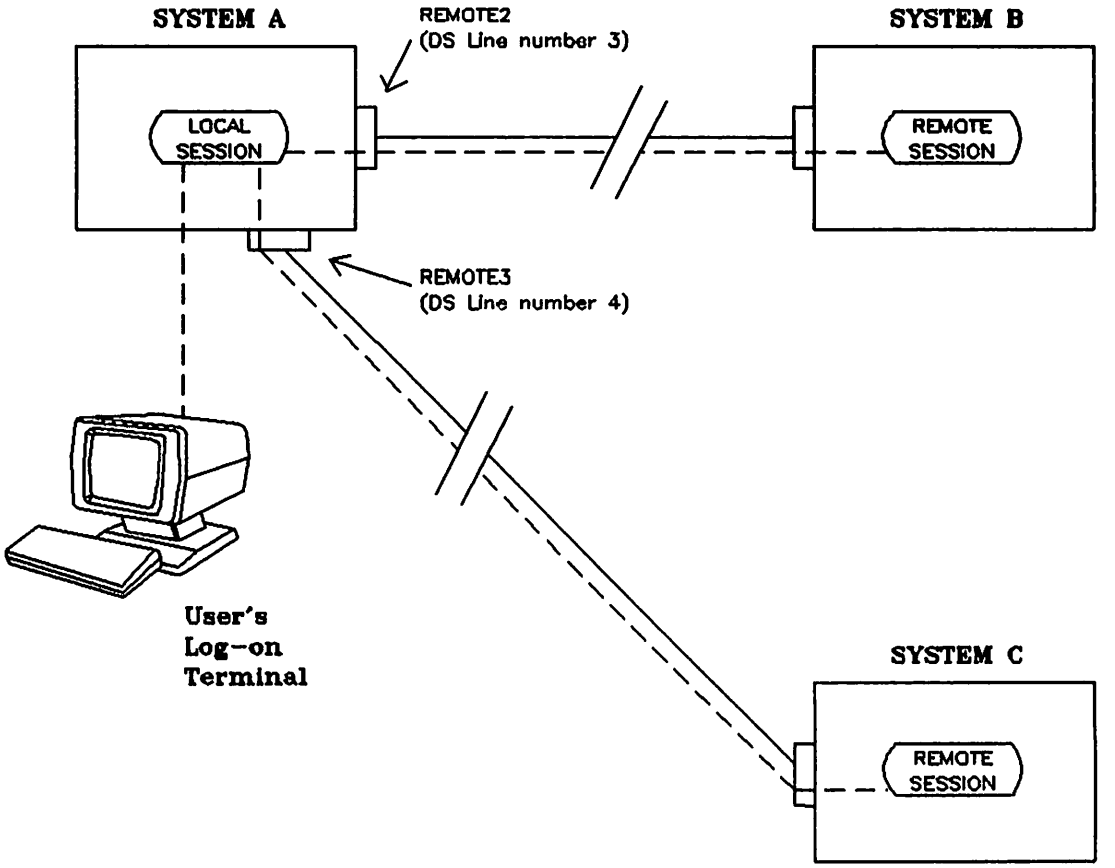


Figure 2-16. Multiple Line Example (Telephone Lines)

First the user sat down at a terminal connected to System A and initiated a local session:

```
:HELLO USER.ACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:37 PM

WELCOME TO SYSTEM A.
:
```

USER and ACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System A.

At this point, we have the situation illustrated in Figure 2-17. Notice that, so far, no communications link exists between any of the three systems.

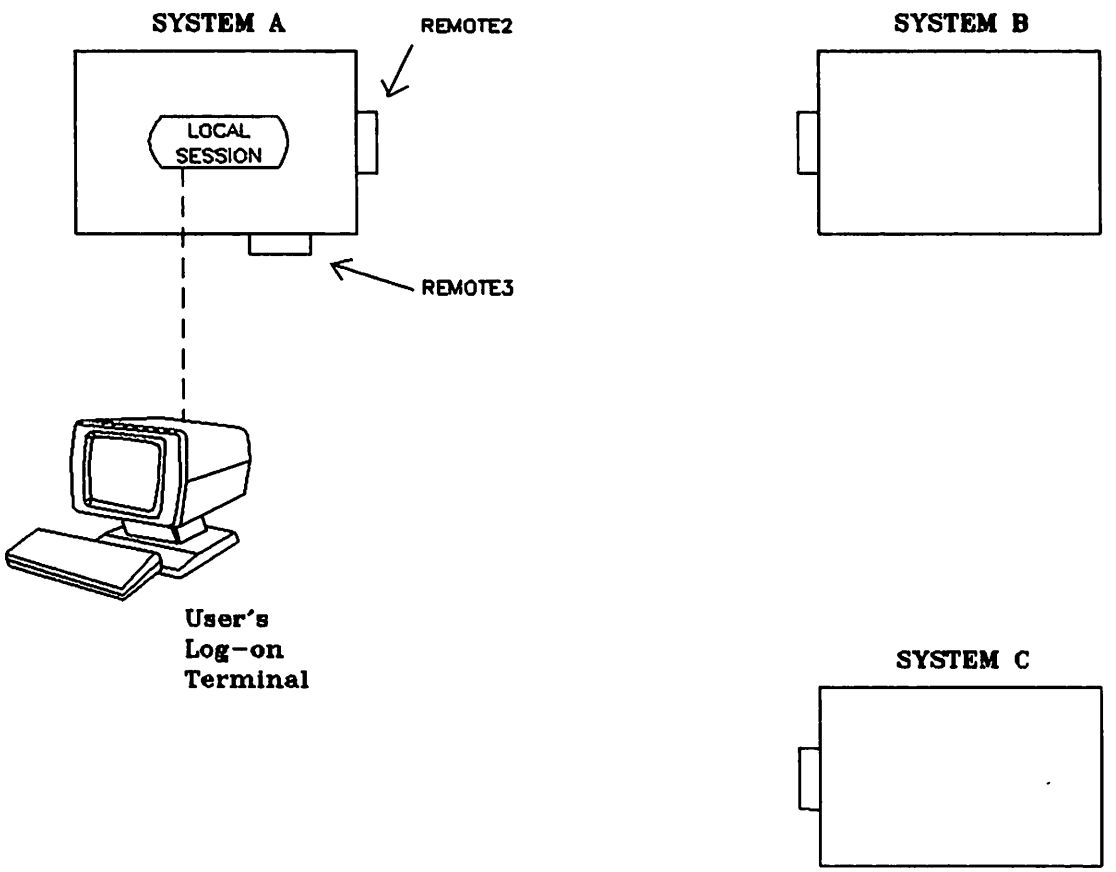


Figure 2-17. Initiating the Local Session (Dial-up Example)

Next, the user acquired access to a telephone connection between Systems A and B and initiated a remote session in System B:

```
:DSLINE REMOTE2 ;LOCID="A" ;REMID="B" ;PHNUM=555-8001
DS LINE NUMBER = #L3
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:38 PM

WELCOME TO SYSTEM B.
:
```

REMOTE2 is the device class name (as defined within System A) associated with the particular line, A and B are the ID sequences identifying Systems A and B, respectively, and 555-8001 is the telephone number of the modem connected to the communications interface at System B. RUSER and RACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System B.

Now we have the situation illustrated in Figure 2-18.

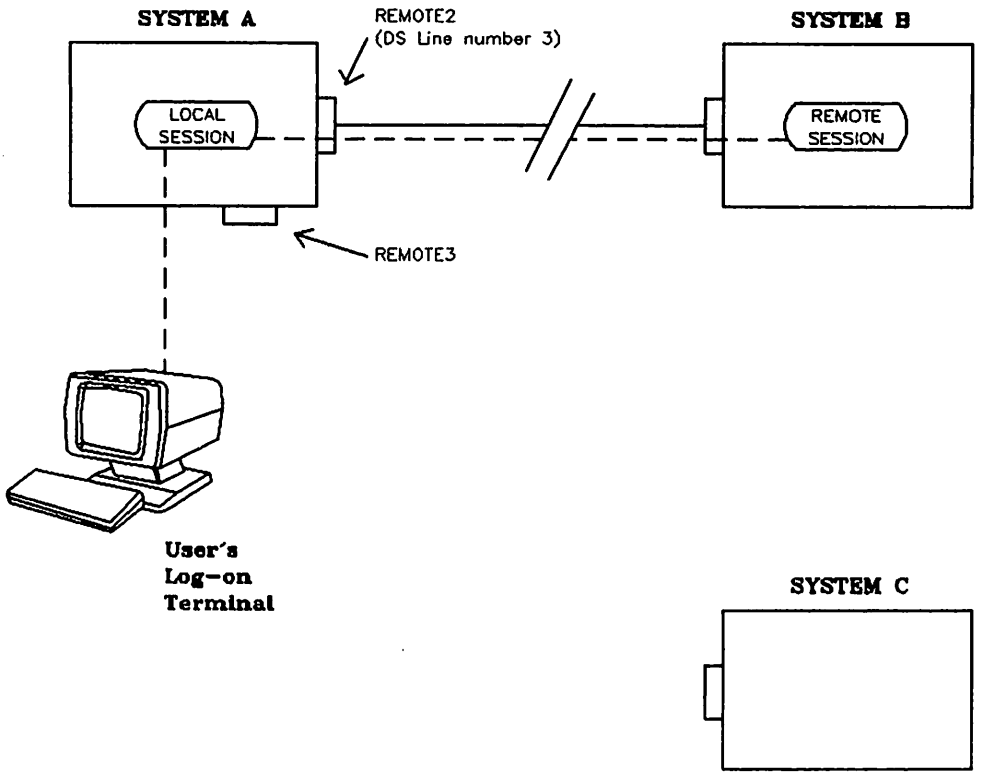


Figure 2-18. Establishing the Link With System B (Dial-up Example)

Finally, the user acquired access to a line between Systems A and C and initiated a remote session in System C:

```
:DSLINE REMOTE3 ;LOCID="A" ;REMID="C" ;PHNUM=377-2000
DS LINE NUMBER = #L4
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:39 PM

WELCOME TO SYSTEM C.

:
```

REMOTE3 is the device class name (as defined within System A) associated with the particular line, A and C are the ID sequences identifying Systems A and C, respectively, and 377-2000 is the telephone number of the modem connected to the communications interface at System C. RUSER and RACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System C.

We end up with the situation illustrated in Figure 2-19, which is identical to Figure 2-16 that started this example.

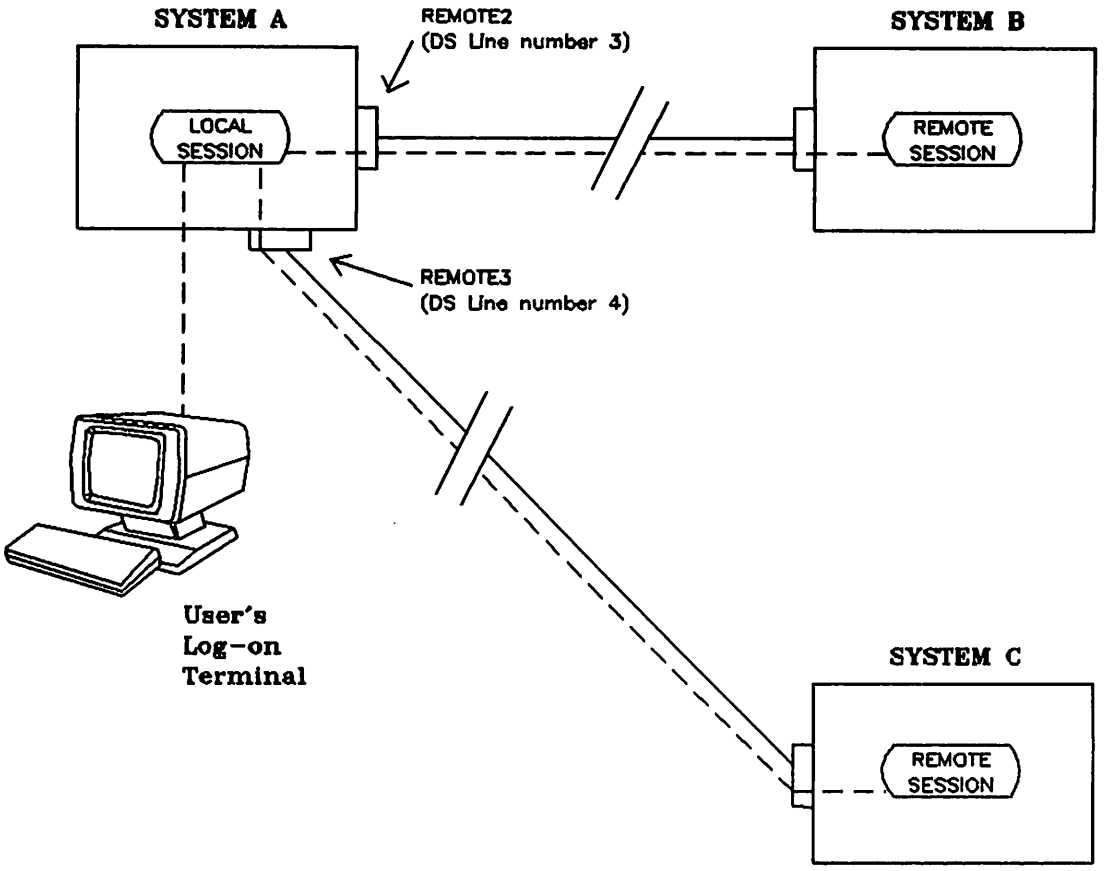


Figure 2-19. Establishing the Link With System C (Dial-up Example)

Figure 2-20 illustrates a situation where a user has established two X.25 communications links concurrently from within a single local session. Take a closer look at that situation and examine the sequence of commands that was used to create it.

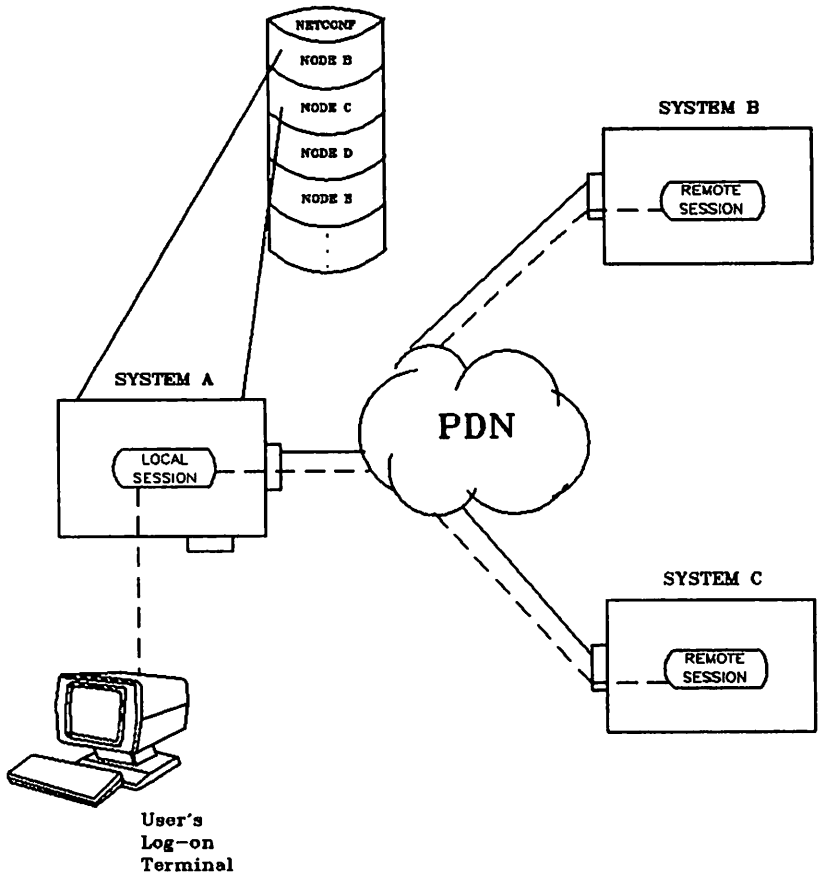


Figure 2-20. Multiple Line Example (X.25 Lines)

First the user sat down at a terminal connected to System A and initiated a local session:

```
:HELLO USER.ACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:37 PM

WELCOME TO SYSTEM A.
:
```

USER and ACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System A.

At this point, we have the situation illustrated in Figure 2-21. Notice that, so far, no communications link exists between any of the three systems.

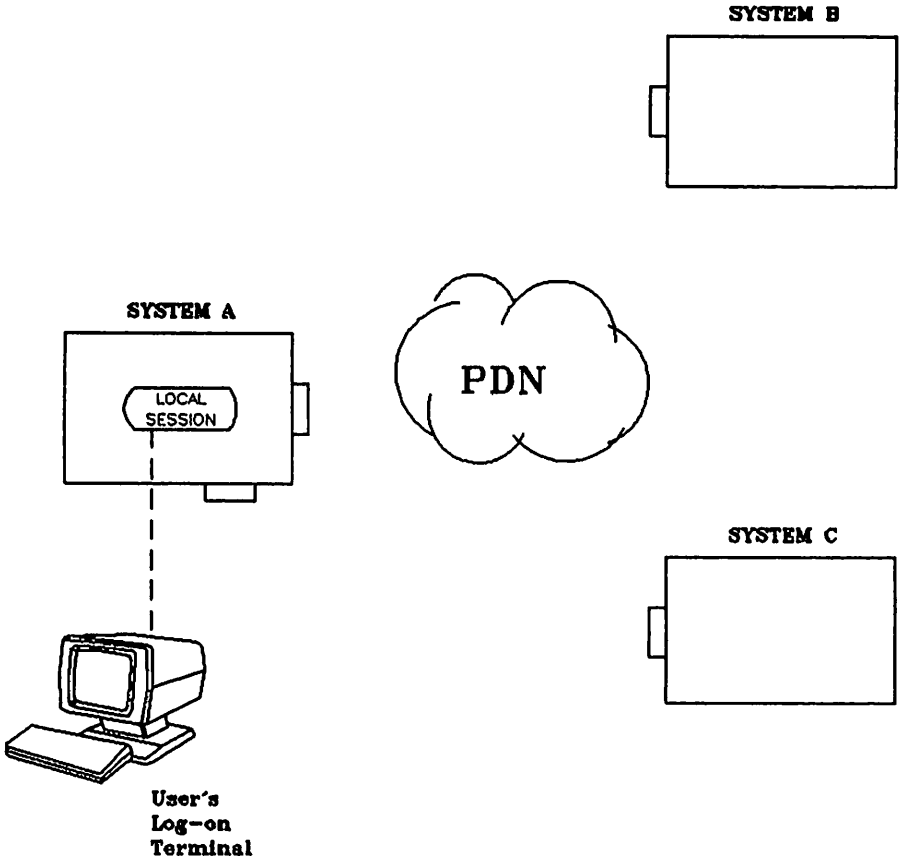


Figure 2-21. Initiating the Local Session (X.25 Example)

Next, the user acquired access to an X.25 connection between Systems A and B and initiated a remote session in System B:

```
:DSLINE NODEB
DS LINE NUMBER = #L3
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:38 PM

WELCOME TO SYSTEM B.
:
```

NODEB is the nodename (as defined within System A) associated with the particular line. RUSER and RACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System B.

Now we have the situation illustrated in Figure 2-22.

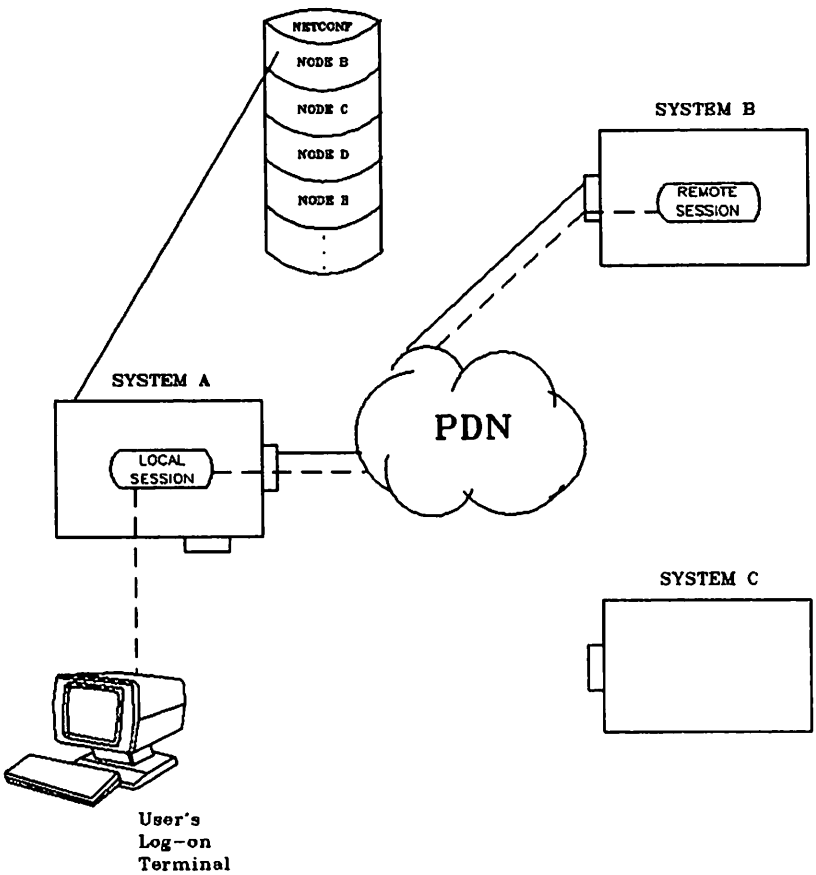


Figure 2-22. Establishing the Link With System B (X.25 Example)

Finally, the user acquired access to a line between Systems A and C and initiated a remote session in System C:

```
:DSLINE NODEC
ENVIRONMENT 2: REMOTE3
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.   TUE, OCT 22, 1985, 1:39 PM

WELCOME TO SYSTEM C.

:
```

NODEC is the nodename (as defined within System A) associated with the particular line. RUSER and RACCOUNT are valid user and account names, respectively, as defined by the accounting structure of System C.

We end up with the situation illustrated in Figure 2-23, which is identical to Figure 2-20 that started this example.

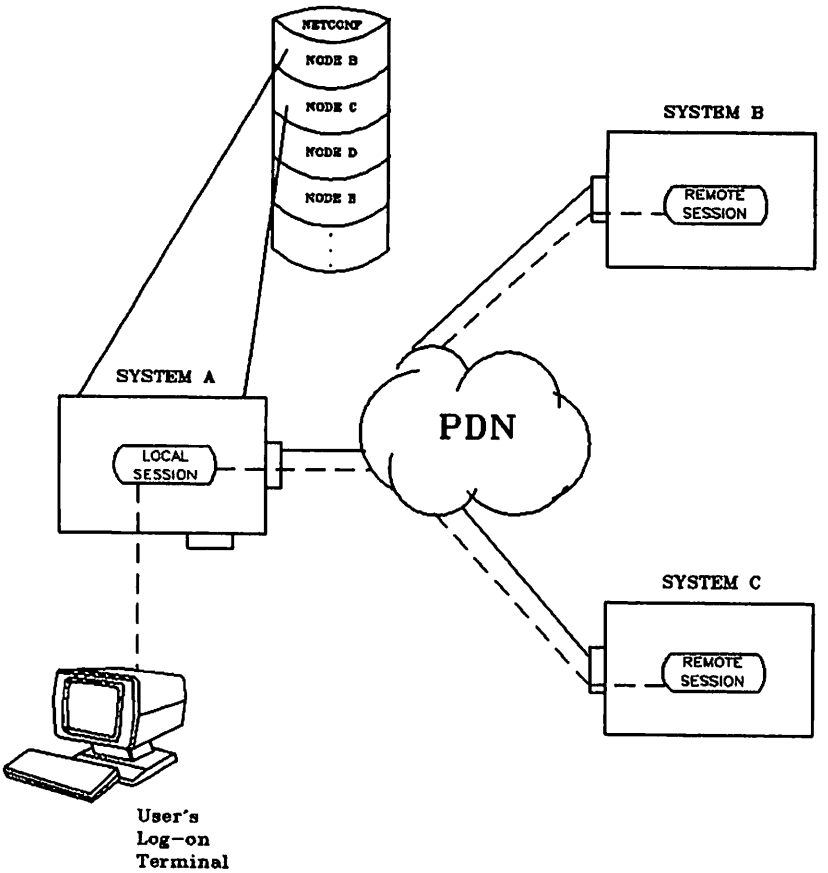


Figure 2-23. Establishing the Link With System C (X.25 Example)

Line Opening Failures

There are several reasons why a :DSLINE command for opening a communications line might be rejected by DS/3000, some of which have been illustrated earlier in this section.

The following list summarizes the likely causes of a line opening failure that are common to lines in general:

- You made a syntax error in the :DSLINE command.
- You gave an erroneous line or node name specification (*dsdevice*) in the :DSLINE command. (There is no IODSO entry in the system configuration with the specified device class name or logical device number, or the remote node name you specified was not defined in the network configuration database).
- The line was not opened by the local console operator.
- The line was not opened by the remote console operator.
- The remote operator may have lowered the session LIMIT.
- The local console operator may have used the SLAVE option to limit DS activity to incoming users only.
- The remote console operator may have used the MASTER option to prohibit access to the system.
- All virtual terminals on the remote system are already in use, which means there are no remote resources available for you to establish a remote session.
- All virtual circuits are in use (X.25 only).
- Someone has exclusive access to the specified line.
- You asked for EXCLUSIVE access to a line that is in use.
- DS/3000 detected a hardware problem (the communications interface board is not responding correctly).

The following list summarizes the additional causes of a line opening failure on a dial-up telephone line:

- The operator was not able to make the requested telephone connection and entered NO through the system console in response to the dial request message.
- The remote computer rejected your local ID sequence.
- The remote computer did not send a valid ID sequence (the received ID sequence did not match any of the remote ID sequences that you specified or, if you didn't specify any, did not match any of the configured remote ID sequences).
- The specified line is already in use and the remote ID sequence you supplied did not match the one used by the currently connected remote HP 3000.

The various error numbers and messages that might appear as a result of line opening failures are included in the summary of error codes and messages in Appendix A.

Closing a Line

Once you have opened one or more communications lines, you can close any or all of them by using a variation of the :DSLINE command. The line closing format of the :DSLINE command is presented here.

SYNTAX

```
:DSLINE {dsdevice     } ;CLOSE
        {dslineNumber }
        {@            }
```

PARAMETERS

dsdevice — The device class name, logical device number, or logical node name specified in the :DSLINE command that opened a particular line.

dslineNumber — The DS line number assigned to you by DS/3000 when the particular line was opened. When this parameter is used, it must appear in the format #Ln, where n is the line number (see "Examples" on the following page).

@ — This parameter specifies that you wish to close all of the lines that you currently have open.

;CLOSE — This parameter specifies that you wish to close the specified line(s).

If no line identifier (*dsdevice*, *dslineNumber*, or @) is specified, DS/3000 closes the line that you most recently opened.

NOTE

When the last node issues a :DSLINE CLOSE command, the message CS I/O ERROR will appear on the remote console. This is normal, and should be ignored.

Examples

The following examples illustrate the variations of the :DSLINE command that can be used for closing one or more communications lines.

```
:DSLINE REMOTE3 ;CLOSE
```
This form closes the line that is identified by the device class name REMOTE3.

```
:DSLINE 55 ;CLOSE
```
This form closes the line that is identified by the logical device number 55.

```
:DSLINE @ ;CLOSE
```
This form closes all the lines that you currently have open.

```
:DSLINE #L3 ;CLOSE
```
This form closes the line that is identified by #L3.

```
:DSLINE ;CLOSE
```
This form closes the line that you most recently opened.

If you are sharing one or more physical communications lines with other users, the above forms of the :DSLINE command close the line(s) for your application only (the other user's applications are not affected).

REMOTE SESSIONS

SECTION 3

A communications link exists after you have initiated a session in the remote HP 3000 under the username, accountname, and groupname specified in the :REMOTE HELLO command. You now have two distinct sessions in existence simultaneously from the same log-on terminal: a local session (in the HP 3000 to which you first logged on) and a remote session (in the HP 3000 at the other end of the communications line).

Within the local session, you have access to all I/O devices and disc files in your local HP 3000 (subject to the usual MPE file security, of course). This is a normal MPE interactive session in every respect. You enter MPE commands and use the various language and utility subsystems exactly as you would if DS/3000 were not present. This local session is running under the username, accountname, and groupname specified in the :HELLO command that you used to first log on. All user capabilities and file access available to you within the local session are determined by those log-on parameters.

Within the remote session, you have access to all I/O devices and disc files in the remote HP 3000 (again, subject to the usual MPE file security). With the few minor exceptions described in the following pages, this is also a normal MPE interactive session. All MPE commands and subsystems are, however, executed in the remote HP 3000. The terminal output resulting from the executed commands and subsystems appears at your local log-on terminal. The remote session is running under the username, accountname, and groupname specified in the :REMOTE HELLO command that you used in establishing the communications link. All user capabilities and file access available to you within the remote session are determined by those log-on parameters.

For the sake of clarity and as a learning aid, the remainder of this section will treat local and remote sessions as separate (and essentially unrelated) entities that use only those resources available in the particular HP 3000 in which they are running. Actually, it is possible to access the I/O devices and disc files of the remote HP 3000 computer from your local session, and it is also possible to access the I/O devices and disc files of the local HP 3000 from your remote session. This more advanced activity will be covered in Section 4, "Remote File Access."

ISSUING REMOTE COMMANDS

Remember that, in the previous sections, the following sequence of commands was used to establish the communications link:

```
(RETURN)
:HELLO USER.ACCOUNT

HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 9:05 AM
```

HELLO command and log-on display for local session.

```
WELCOME TO SYSTEM A.
:DSLINE REMOTE2
DS LINE NUMBER = #L3
```

```
:REMOTE HELLO RUSER.RACCOUNT

HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 9:06 AM

WELCOME TO SYSTEM B.
:
```

HELLO command and log-on display for remote session.

At this point, the remote session has been initiated, but you are currently in the local session (as signified by the colon prompt character). To execute a command in the remote session, use the following construct:

```
:REMOTE [xxx] command
```

where xxx is the DS line number returned by DS/3000 when the communications line was opened, and command is the desired MPE command in its normal format. (The DS line number is necessary only if you have more than one communications line open simultaneously; if it is omitted, then the line which you most recently opened is referenced by default). In the example in Section 1, this construct was used to execute a LISTF command, as follows:

```
:REMOTE LISTF

FILENAME

DATA1     DATA5     DATA6     FILE3     SOURCE1
:
```

Because the prefix REMOTE was included, the LISTF command is executed in the remote session (the implied account and group names are those established by the :REMOTE HELLO command that initiated the remote session). Although the LISTF command is executed in the remote HP 3000, the output generated by the command is displayed at your local log-on terminal.

Notice, in the above example, that the DS line number associated with the particular communications line was not specified (3 in this example). This is because, if no line number is specified, DS/3000 uses (by default) the line most recently opened. Only one communications line is open from your local session; so DS/3000 uses that line by default. If you had opened a second line, you would need to tell DS/3000 in which remote computer the remote command is to be executed. To tell DS/3000, include the appropriate DS line number in the remote command, as follows:

```
:REMOTE 3 LISTF

FILENAME

DATA1     DATA5     DATA6     FILE3     SOURCE1
:
```

The above construct only allows you to execute a single remote command. After the remote command has been executed, control returns to your local session (as signified by the colon prompt character).

But suppose that you want to execute a whole series of remote commands. It would obviously be a nuisance to have to prefix each command with the word REMOTE. DS/3000 provides a convenient solution to this situation. To execute a series of commands in the remote session, use the following construct:

```
:REMOTE [xxx]
```

where xxx is again the DS line number of the desired communications line (specifying in which remote HP 3000 you want to execute commands). DS/3000 then prompts you for each command by displaying a # in column 1 of your terminal (in place of the standard MPE colon prompt). In the example in Section 1, this construct was used for entering two remote MPE commands, EDITOR and BYE.

After reviewing the example in Section 1, try another example that uses more than those two remote commands:

```
:REMOTE
#LISTF

FILENAME

DATA1     DATA5     DATA6     FILE3     SOURCE1
#PURGE DATA5
#PURGE DATA6
#LISTF

FILENAME

DATA1     FILE3     SOURCE1
#RUN FCOPY.PUB.SYS

HP32212A.0.03 FILE COPIER

>FROM=DATA1 ;TO=DATA2 ;NEW
EOF FOUND IN FROMFILE AFTER RECORD 679

680 RECORDS PROCESSED *** 0 ERRORS

>EXIT

END OF PROGRAM
#LISTF

FILENAME

DATA1     DATA2     FILE3     SOURCE1
#BYE

CPU=4. CONNECT=7. MON, OCT 28, 1985, 9:13AM
#:
:
```

Notice that except for the # prompt (in place of the standard colon prompt) this looks exactly like a normal MPE interactive session. All of the commands shown in the previous example are entered through the local log-on terminal, but the MPE and FCOPY commands are executed in the remote session within the remote HP 3000. After each remote MPE command was executed, however, control remained in the remote session (as signified by the # prompt character). When the remote session was terminated and the user typed a colon (:) in response to the # prompt following the log-off message, control was then returned to the local session (as signified by the colon prompt).

Using the Remote Subsystem from a Batch Job

While in a batch job, you can establish a remote session by using the :DSLINE or :REMOTE HELLO command.

The job to be streamed may be similar to the following:

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

NOTE

The remote # prompt is optional.

An important point to remember is that, once established, the remote session is interacting with the job in the same way as it would interact with a terminal. If the remote session detects an error, the error is printed to $STDLIST. If the error generates a user prompt, the next record in the job file is read as the response (in the same manner as waiting for a character or RETURN on a terminal). The record is then lost to the job.

The BREAK Key

Within a remote session, you can use BREAK to temporarily interrupt remote processing. When doing so, either you may return control to the MPE Command Interpreter of your local HP 3000, or you may temporarily suspend the remote subsystem that you are executing without returning control to the local HP 3000. This is determined by how you execute commands in the remote session. There are two ways to execute commands in a remote session:

- By prefixing each command with the word REMOTE.
- By entering the word REMOTE, which prompts you for each command.

Prefixing Each Command with REMOTE.

When you are conducting a remote session by prefixing each command with the word REMOTE, pressing BREAK returns control to the local Command Interpreter and you receive the colon (:) prompt. To continue remote processing at the point where it was interrupted, you merely enter REMOTE RESUME in response to the local MPE colon prompt.

As an example, assume that you are in the midst of using the text editor in a remote session when you decide to start a job stream executing concurrently in your local HP 3000. The sequence of commands would be similar to the following:

```
:REMOTE EDITOR

HP32201A.7.17 EDIT/3000 MON, OCT 28, 1985, 9:11 AM
(C) HEWLETT-PACKARD CO. 1985
/ADD
    1 DOE, JOHN              29 M CHI
    2 BLACK, PATRICIA        23 F SF
    3 SIMON, NEIL            43 M NY
    4 MACK, SHIRLEY          38 F DET
    5 <------------BREAK pressed here.
Local
session
prompt.
  |
  |
  |-> :STREAM COBTEST1          }   Control is now
      #J19                    }   in the local
    :REMOTE RESUME            }   session.

    READ pending
    MICHAELS, WILLIAM      32 M CHI                  }   Control is now
    6 O'LEARY, TIMOTHY       49 M DET                }   back in the
    7 MARTIN, MARY           34 F LA                 }   remote session.
    8 MURIN, JOICE           42 F CHI

      .
      .
      .
```

Notice that when (BREAK) was pressed, the text editor in the remote HP 3000 was waiting for you to enter the text for line 5. (BREAK) interrupted the remote session and passed control to the MPE Command Interpreter of the local HP 3000 (as signified by the colon prompt). The STREAM command was issued within the local session, which caused the file COBTEST1 to be executed in the local HP 3000. Then, when the RESUME command was issued, control was passed back to the remote session at the point where it was interrupted (that is, the text editor in the remote HP 3000 is now waiting for you to enter the text for line 5). When the text for line 5 is entered, the remote session proceeds as though nothing had happened.

Note that by the end of the example, the local job stream, the local session, and the remote session are all operational simultaneously.

Entering REMOTE.

When you are interacting with the remote Command Interpreter by having entered the word REMOTE and you are receiving the remote prompt (#), pressing (BREAK) will temporarily suspend the subsystem you are executing and will return control to the remote Command Interpreter. To continue remote processing at the point where it was interrupted, you merely enter RESUME in response to the remote prompt.

As an example, assume that you are in the midst of using the text editor in a remote session when you decide to start a job stream executing in your remote HP 3000. The sequence of commands would be similar to the example shown previously, but with a few minor differences, as follows:

```
:REMOTE
#EDITOR

HP 32201A.7.17 EDIT/3000 MON, OCT 28, 1985, 9:20 AM
(C) HEWLETT-PACKARD CO. 1985
/ADD
    1 LEWIS, LEO            51 M SV
    2 LAGERGREN, FRED       25 M SJ
    3 DICKINSON, MARY       21 F SC
    4 LAGREGREN, LINDA      24 F SJ
    5 <┐
      └── BREAK pressed here.

#STREAM APLTEST1          }   Control is
  #J20                    }   still in the
#RESUME                   }   remote session.

READ pending
MELLO, HENRY            44 M SJ
    6 SOARES, JOE           59 M LA
    7 LAWRENCE, ALICE       44 F SJ
    8 LEWIS, BOB            29 M WASH
```

Notice that when (BREAK) was pressed, the text editor in the remote HP 3000 was waiting for you to enter the text for line 5. (BREAK) interrupted the remote session, but control remained in the remote HP 3000 (as signified by the remote # prompt). The STREAM command executed the file APLTEST1 within the remote HP 3000. Then, when the RESUME command was issued, control was passed back to the point where the text editor was interrupted (that is, the text editor is waiting for you to enter the text for line 5). When the text for line 5 is entered, the remote session proceeds as though nothing had happened.

The Control Keys

Within a remote session (CONTROL)H, (CONTROL)X, and (CONTROL)Y perform exactly the same functions as they do in a normal MPE interactive session.

For example, if you are using FCOPY or the text editor in a remote session, you can use (CONTROL)Y to prematurely terminate an FCOPY or text editor operation. When the operation terminates, control is still in the particular subsystem within the remote session.

Similarly, you can use (CONTROL)H to delete the last character entered or (CONTROL)X to delete the line of text currently being entered. In both of these cases, after the deletion occurs, control remains in the remote session.

ISSUING LOCAL COMMANDS

Whenever the standard MPE colon prompt is displayed at your terminal, you are in the local session. Within the local session, you enter MPE commands in their normal format in response to the colon prompt. If you are in the midst of a remote session (that is, you used the command :REMOTE, and DS/3000 is issuing the # prompt character), you can return control to your local session by entering a colon, as follows:

```
#:
```

In response to the remote colon, control returns to the MPE Command Interpreter of your local HP 3000 which then prompts you for local commands with the colon prompt character. Note that the remote colon does not terminate the remote session; you can resume processing in the remote session by again using either of the constructs described under "Issuing Remote Commands."

TERMINATING A REMOTE SESSION

You can terminate a remote session either from within the local session or from within the remote session itself.

From the Local Session

Whenever the standard MPE colon prompt is displayed at your terminal, you are in the local session. To terminate a remote session from within your local session, use the following command:

```
:REMOTE [xxx] BYE
```

where xxx is the DS line number associated with the communications line connecting the particular remote session to your local session. (The DS line number is necessary only if you have more than one communications line open simultaneously; if it is omitted then the line that you most recently opened is referenced by default.)

For instance, in the example in Section 1, either of the following sequences could have been used to terminate the remote session:

```
#:
:REMOTE BYE

CPU=4. CONNECT=7. MON, OCT 28, 1985, 9:13 AM
:
```

*** OR ***

```
#:
:REMOTE 3 BYE

CPU=4. CONNECT=7. MON, OCT 28, 1985, 9:13 AM
:
```

In both cases, the remote colon was used to return control from the remote session to the local session. In either case, the remote session is terminated. Remote sessions are also automatically terminated when the local session is terminated.

If the communications line was opened using the DSLINE= parameter of the :REMOTE HELLO command, the line is automatically closed when the remote session terminates. To initiate another remote session over the same communications line, you must once again open the line (using either the :DSLINE command or the DSLINE= parameter of the :REMOTE HELLO command) and then issue another :REMOTE HELLO command.

If the communications line was opened using the :DSLINE command, it is still open. To initiate a new remote session over the same communications line, merely issue another :REMOTE HELLO command (you do not need to issue another :DSLINE command because the communications line is still open). To close the communications line, use the constructs presented in Section 2.

From the Remote Session

Whenever the # prompt is displayed at your terminal, you are in the remote session. To terminate a remote session from within the remote session itself, use the following command:

```
#BYE
```

Note that you do not need to supply a DS line number in this case, because DS/3000 knows implicitly which remote session you wish to terminate (that is, the one in which the #BYE command is executed).

Remember that this command was used to terminate the remote session in the example at the end of Section 1, as follows:

```
#BYE

CPU=4. CONNECT=7. MON, OCT 28, 1985, 9:15 AM
#
```

Notice that although the remote session is terminated, DS/3000 is still issuing the # prompt character. To return control to the local session, issue a colon (described earlier under "Issuing Local Commands").

If the communications line was opened using the :DSLINE command, it is still open. To initiate another remote session over the same communications line, merely issue an appropriate remote MPE

HELLO command. (You do not need to use the prefix REMOTE because DS/3000 is still waiting for you to enter a remote command; nor do you need to issue another DSLINE command because the communications line is still open.) To close the communications line, use the constructs presented in Section 2.

If the communications line was opened using the DSLINE= parameter of the :REMOTE HELLO command, the line is automatically closed when the remote session terminates. To initiate another remote session over the same line, you must once again open the line (using the :DSLINE command or the DSLINE= parameter of a :REMOTE HELLO command) and then issue another :REMOTE HELLO command.

NOTE

Including a :BYE command in a remote logon UDC is not recommended. For a fuller explanation regarding the use of logon UDCs on a remote system, refer to the discussion of the :REMOTE HELLO command in Section 2, pages 2-29 to 2-33.

REMOTE FILE ACCESS

SECTION 4

In the preceding sections, you have seen how you can establish a communications link between two HP 3000s and thereby use the computing power of the remote HP 3000. But through the use of the DS/3000 Remote File Access (RFA) capability, programs running in your local session can:

- Use any of the devices connected to the remote HP 3000 as though they were connected directly to your local HP 3000
- Access any of the disc files of the remote HP 3000 (subject to the normal MPE file security, of course) as though they resided at your local HP 3000 site.

Section 4 is divided into two main parts. The first part, "Interactive Access", describes how you can issue local MPE FILE commands that define devices and/or files residing at the remote HP 3000 site. The second part, "Programmatic Access", describes how you can use the standard set of MPE File System intrinsics within your local programs to access devices and/or files residing at the remote HP 3000 site.

INTERACTIVE ACCESS

After a DS/3000 communications link has been established, you can issue local MPE FILE commands that define devices and/or files residing at the remote HP 3000 site. To make this possible, the DEV= parameter of the MPE :FILE command was expanded to include a DS line specification in addition to the usual device specification. To specify a file that resides across a DS line, the format of the DEV= parameter is as follows:

;DEV=[*dsdevice*]#[*device*]

where *dsdevice* is the device class name, logical device number, or node name that you used when establishing the particular communications link (this specifies the physical line connecting the two computers); and *device* is the device class name or logical device number of the desired remote device as established within the remote HP 3000. It defaults to DISC. (Refer to the *MPE Commands Reference Manual* for the complete syntax and all parameters.)

NOTE

When the :FILE command is entered on a remote system to point back to a file on the local system, *dsdevice* is omitted.

The *dsdevice#* parameter (within the DEV= parameter) is the only parameter of the MPE :FILE command that is unique to DS/3000. This one small item of syntax is enormously powerful. It means that from within your local session you can access any of the devices and/or disc files of a remote HP 3000 as though they resided at your local HP 3000 site. Access to remote disc files is, of course, subject to the usual MPE file security. The user, account, and group names that you specified in the

:REMOTE HELLO command when establishing the communications link are the ones used by MPE in the remote HP 3000 for determining your file access capabilities.

If *device* is omitted, the local system is used instead of the remote system. This is useful in cases where, for example, a program is located on a remote system, but the data it uses is located on the local system. See Example #5.

Following are five annotated examples illustrating remote device and file access from a local session.

Example #1: Remote Off-Line Listing

Assume that you, using the Text Editor on your HP 3000, are to maintain an ASCII file containing both uppercase and lowercase characters, but that you don't have an upper/lowercase line printer. Assume further that elsewhere in the same building there is another HP 3000 with an upper/lowercase line printer, that both HP 3000s have DS capability, and that they are connected to one another by an interconnecting cable and communications interfaces. You can access the remote line printer as follows.

First, the console operators of both computer systems OPEN the line. (See Section 9.) Then, you log on to your HP 3000 and establish a communications link with the remote HP 3000.

```
:HELLO USER.ACCOUNT
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
```

where USER and ACCOUNT are valid user and account names (respectively) within the accounting structure of your local HP 3000, RUSER and RACCOUNT are valid user and account names (respectively) within the accounting structure of the remote HP 3000, and LINE2 is the device class name of the local IODS0 entry (or the node name associated with the IODSX entry) for the line that you want to use.

Next, issue a local MPE :FILE command that defines the desired line printer as a remote device.

```
:FILE LIST;DEV=LINE2#SLOWLP
```

where LIST is the formal designator by which you will subsequently reference the line printer, LINE2 is the device class name (or node name) you used when establishing the particular communications link, the # symbol tells the local file system that the next parameter references a device on the remote system, and SLOWLP is the device class name (as established within the remote HP 3000) of the upper/lowercase line printer.

Then, invoke the Text Editor of your local HP 3000, specifying the remote line printer as the off-line listing device:

```
:EDITOR *LIST
```

Thereafter, direct the Text Editor offline output to the remote upper/lowercase line printer as though it were connected directly to your local HP 3000. For example, you could print the content of the file TEXTFILE on the upper/lowercase line printer as follows:

```
/TEXT TEXTFILE
/LIST ALL,OFFLINE
```

The entire command sequence is as follows (see Figure 4-1):

```
:HELLO USER.ACCOUNT
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:51 PM

WELCOME TO SYSTEM A.

:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
DS LINE NUMBER = #L3
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:52 PM
```

```
WELCOME TO SYSTEM B.

:FILE LIST;DEV=LINE2#SLOWLP
:EDITOR *LIST
HP32201A.7.17 EDIT/3000 MON, OCT 28, 1985, 12:53 PM
(C) HEWLETT-PACKARD CO. 1985
/TEXT TEXTFILE
/LIST ALL,OFFLINE
*** OFF LINE LISTING BEGUN. ***

.
.
.
```

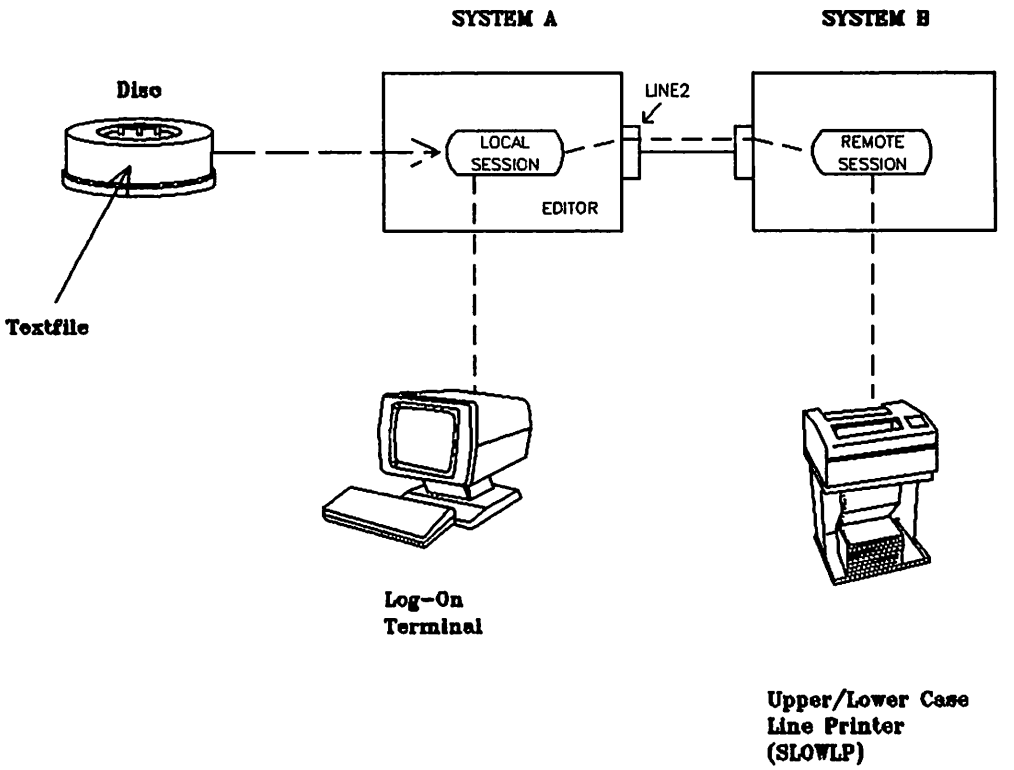


Figure 4-1. Remote Off-Line Listing Example

Example #2: Locally Sorting a Remote File

Assume that there is a file named SOURCE residing on a disc connected to a remote HP 3000 and that SOURCE contains a list of clients sorted alphabetically by the clients' names. Assume further that the remote HP 3000 and your local HP 3000 both have DS/3000 configured and that they are interconnected by a hardwired connection. You wish to access the remote file SOURCE from your local HP 3000, sort its content alphabetically by the names of the states in which the clients reside, and store the sorted version in a newly created disc file named SORTED on your local HP 3000. You can do that (without disturbing the original content of SOURCE) as follows.

First, the console operators of both computer systems OPEN the line. (See Section 9.) Then, log on to your local HP 3000 and establish a communications link with the remote HP 3000.

```
:HELLO USER.ACCOUNT
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
```

where USER and ACCOUNT are valid user and account names (respectively) within the accounting structure of your local HP 3000, RUSER and RACCOUNT are valid user and account names (respectively) within the accounting structure of the remote HP 3000, and LINE2 is the device class name of the local IODS0 entry (or the node name associated with the IODSX entry) for the line that you want to use.

Next, issue a local MPE :BUILD command to create the local disc file SORTED that will receive the sorted output.

```
:BUILD SORTED;DISC=250,1,1;REC=-80,16,F,ASCII
```

Then, issue two local MPE :FILE commands: one that defines the remote disc file SOURCE as the sort input file and one that defines the local disc file SORTED as the sort output file.

```
:FILE INPUT=SOURCE;DEV=LINE2#DISC
:FILE OUTPUT=SORTED
```

Then, invoke the Sort program, specify the sort key, and initiate the actual sort.

```
:RUN SORT.PUB.SYS
>KEY 50,9
>END
```

Note that the sort is performed in your local HP 3000, using the remote disc file SOURCE as the sort input file; the output of the sort is stored in the local disc file SORTED; and the original content of SOURCE is not altered.

The entire command sequence is as follows (see Figure 4-2):

```
:HELLO USER.ACCOUNT
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:51 PM

WELCOME TO SYSTEM A.
```

```
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
DS LINE NUMBER = #L3
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:52 PM

WELCOME TO SYSTEM B.

:BUILD SORTED;DISC=250,1,1;REC=-80,16,F,ASCII
:FILE INPUT=SOURCE;DEV=LINE2#DISC
:FILE OUTPUT=SORTED
:RUN SORT.PUB.SYS
>KEY 50,9
>END

                    STATISTICS

NUMBER OF RECORDS = 221
RECORD SIZE (IN BYTES) = 80
NUMBER OF INTERMEDIATE PASSES = 0
SPACE AVAILABLE (IN WORDS) = 13,346
NUMBER OF COMPARES = 45
NUMBER OF SCRATCHFILE IO'S = 10
CPU TIME (MINUTES) = .01
ELAPSED TIME (MINUTES) = .14

 END OF PROGRAM
:
.
.
.
```

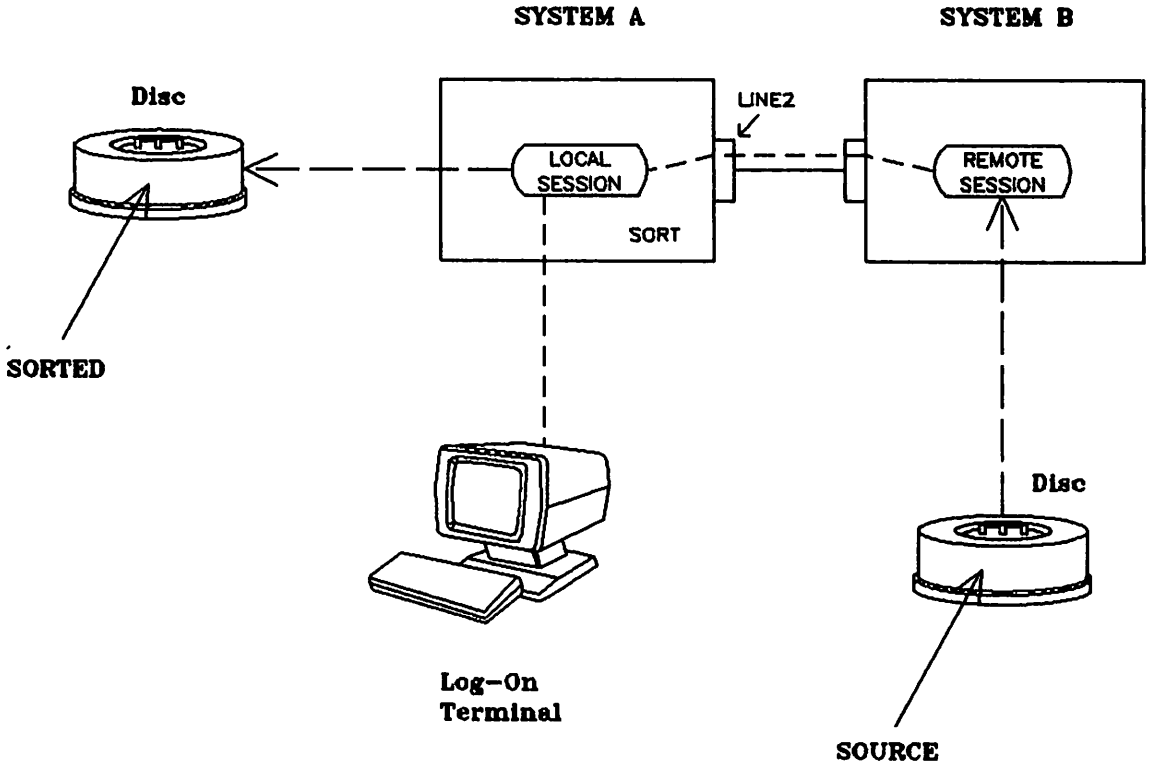


Figure 4-2. SORT Remote File Access Example

Example #3: FCOPYing to a remote system

Suppose that you want to copy a disc file from your local HP 3000 to a remote HP 3000. Assume a hardwired connection and DS/3000 is configured. You can perform the file copy operation as follows.

First, the console operators of both computer systems OPEN the line. (See Section 9.) Then, you log on to your local HP 3000 and establish a communications link with the remote HP 3000.

```
:HELLO USER.ACCOUNT
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
```

where USER and ACCOUNT are valid user and account names (respectively) within the accounting structure of your local HP 3000, RUSER and RACCOUNT are valid user and account names (respectively) within the accounting structure of the remote HP 3000, and LINE2 is the device class name of the local IODS0 entry (or the node name associated with the IODSX entry) for the line that you want to use.

Next, issue a local MPE :FILE command defining the destination file (REMFILE) as being a remote disc file.

```
:FILE REMFILE;DEV=LINE2#DISC
```

Then, invoke the File Copier and specify the file copy parameters.

```
:RUN FCOPY.PUB.SYS
>FROM=LOCFILE;TO=*REMFILE;NEW
```

A new disc file named REMFILE is created in the home group of the RACCOUNT account in the remote HP 3000 and the content of the local disc file LOCFILE is then copied over the communications line into REMFILE.

The entire command sequence is as follows (see Figure 4-3):

```
:HELLO USER.ACCOUNT
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:51 PM

WELCOME TO SYSTEM A.

:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
DS LINE NUMBER = #L3
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:52 PM

WELCOME TO SYSTEM B.

:FILE REMFILE;DEV=LINE2#DISC
:RUN FCOPY.PUB.SYS
HP32212A.3.19 FILE COPIER (C) HEWLETT-PACKARD CO. 1984
```

```
>FROM=LOCFILE;TO=*REMFILE;NEW
EOF FOUND IN FROMFILE AFTER RECORD 2017

2018 RECORDS PROCESSED *** 0 ERRORS

>EXIT
END OF PROGRAM

    .
    .
    .
```

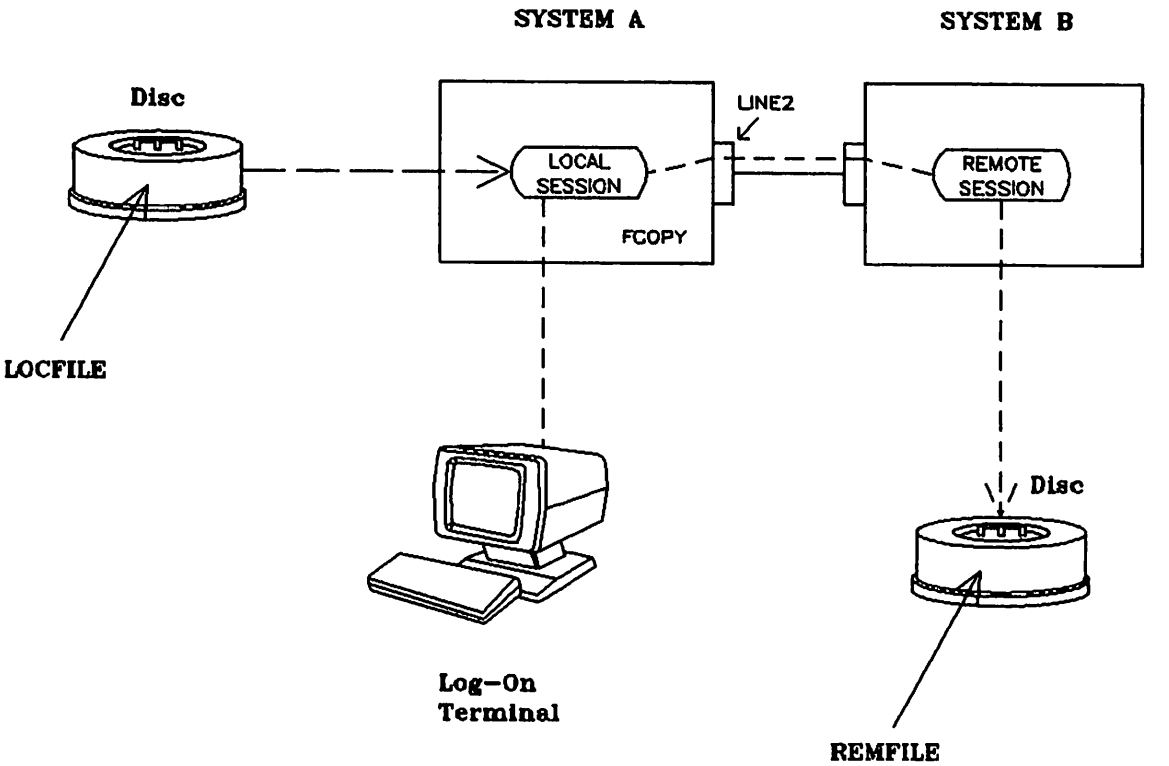


Figure 4-3. FCOPY Remote File Access Example

Example #4: Locally Running Remote Programs

Assume that there is a COBOL source file named SOURCE1 residing on a disc connected to a remote HP 3000 and that you want to compile, prepare, and execute that program on your local HP 3000. Assume further that the remote HP 3000 and your local HP 3000 both have DS/3000 configured and a hardwired interconnection. You can locally compile, prepare, and execute the remote source file as follows.

First, the console operators of both computer systems OPEN the line. (See Section 9.) Then, log on to your HP 3000 and establish a communications link with the remote HP 3000.

```
:HELLO USER.ACCOUNT
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
```

where USER and ACCOUNT are valid user and account names (respectively) within the accounting structure of your local HP 3000, RUSER and RACCOUNT are valid user and account names (respectively) within the accounting structure of the remote HP 3000, and LINE2 is the device class name of the local IODS0 entry (or the node name associated with the IODSX entry) for the line that you want to use.

Next, issue a local MPE :FILE command defining the file SOURCE1 as being a remote disc file.

```
:FILE SOURCE1;DEV=LINE2#DISC
```

where LINE2 is the node name or the device class name that you used when establishing the communications link and DISC is the device class name (as established within the remote HP 3000) of the disc on which SOURCE1 resides.

Then, invoke the COBOL compiler and the Segmenter of your local HP 3000, specifying the remote disc file SOURCE1 as the inputfile.

```
:COBOLGO *SOURCE1
```

The content of the remote disc file SOURCE1 is compiled, prepared, and executed in your local HP 3000.

The entire command sequence is as follows (see Figure 4-4):

```
:HELLO USER.ACCOUNT
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:51 PM

:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
DS LINE NUMBER = #L3

HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:52 PM

:FILE SOURCE1;DEV=LINE2#DISC
:COBOLGO *SOURCE1

PAGE 0001 HP32213C.02.12 (C) HEWLETT-PACKARD CO. 1983
```

(SOURCE1 is now being compiled.)

```
    DATA AREA IS %000341 WORDS.
    CPU TIME = 0:00:01.  WALL TIME = 0:00:07.
END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.

 END OF COMPILE
```

(The compiled version of SOURCE1 is now being prepared by the MPE Segmenter.)

```
 END OF PREPARE
```

(The compiled and prepared version of SOURCE1 is now being executed.)

```
 END OF PROGRAM
:
.
.
.
```

NOTE

Due to the amount of time and system resources required for COBOL activity, this example does not make efficient use of a DS line. The general rule is to do the COBOL compile, preparation, and run on the same system where the data resides. Sometimes this means copying the data files to another system before (or after) COBOL activity, rather than copying across the line during the COBOL activity.

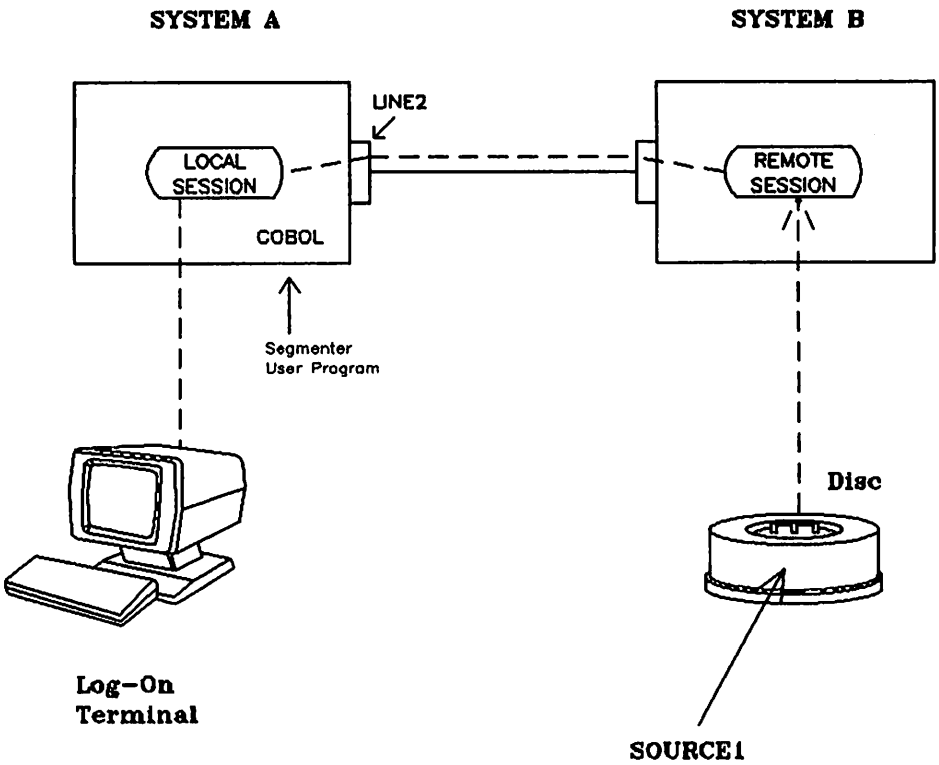


Figure 4-4. COBOLGO Remote File Access Example

Example #5: Remote Programs and Local Data

Assume that there is a COBOL source program named SOURCE1 residing on a disc connected to a remote HP 3000 and that you want to run that program using a data file named DATA1 residing on your local system. Assume further that the remote HP 3000 and your local HP 3000 both have DS capability and a hardwired interconnection. You can do that as follows:

First, log on to your HP 3000 and establish a communications link with the remote HP 3000.

```
:HELLO USER.ACCOUNT
:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
```

where USER and ACCOUNT are valid user and account names (respectively) within the accounting structure of your local HP 3000, RUSER and RACCOUNT are valid user and account names (respectively) within the accounting structure of the remote HP 3000, and LINE2 is the device class name of the local IODS0 entry (or the node name associated with the IODSX entry) for the line that you want to use.

Next, issue a remote MPE :FILE command that defines the data file DATA1 as being a local disc file.

```
:REMOTE
#FILE SOURCE1;DEV=#
```

If you prefer, the :FILE command can be issued from your local session as follows:

```
:REMOTE FILE DATA1;DEV=#
```

where # indicates that the remote session should "go back" to the local session to find the data file.

Then, invoke the COBOL compiler and the Segmenter of your remote HP 3000.

```
#COBOLGO SOURCE1
```

Or, if you prefer, you can run the job from your local session, as follows:

```
:REMOTE COBOLGO SOURCE1
```

The entire command sequence is as follows (see Figure 4-5):

```
:HELLO USER.ACCOUNT
HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:51 PM

:REMOTE HELLO RUSER.RACCOUNT;DSLINE=LINE2
DS LINE NUMBER = #L3

HP3000 / MPE V  G.02.00.  MON, OCT 28, 1985, 12:52 PM

:REMOTE
#FILE DATA1;DEV=#
#COBOLGO SOURCE1

PAGE 0001 HP32213C.02.12 (C) HEWLETT-PACKARD CO. 1983
```

(SOURCE1 is now being compiled.)

```
    DATA AREA IS %000341 WORDS.
    CPU TIME = 0:00:01.  WALL TIME = 0:00:17.
END COBOL/3000 COMPILATION.  NO ERRORS.  NO WARNINGS.

 END OF COMPILE
:
```

(SOURCE1 is now being prepared by the MPE Segmenter.)

```
 END OF PREPARE
```

(The compiled and prepared version of SOURCE1 is now being executed, and is accessing DATA1 for data.)

```
 END OF PROGRAM
:
.
.
.
```

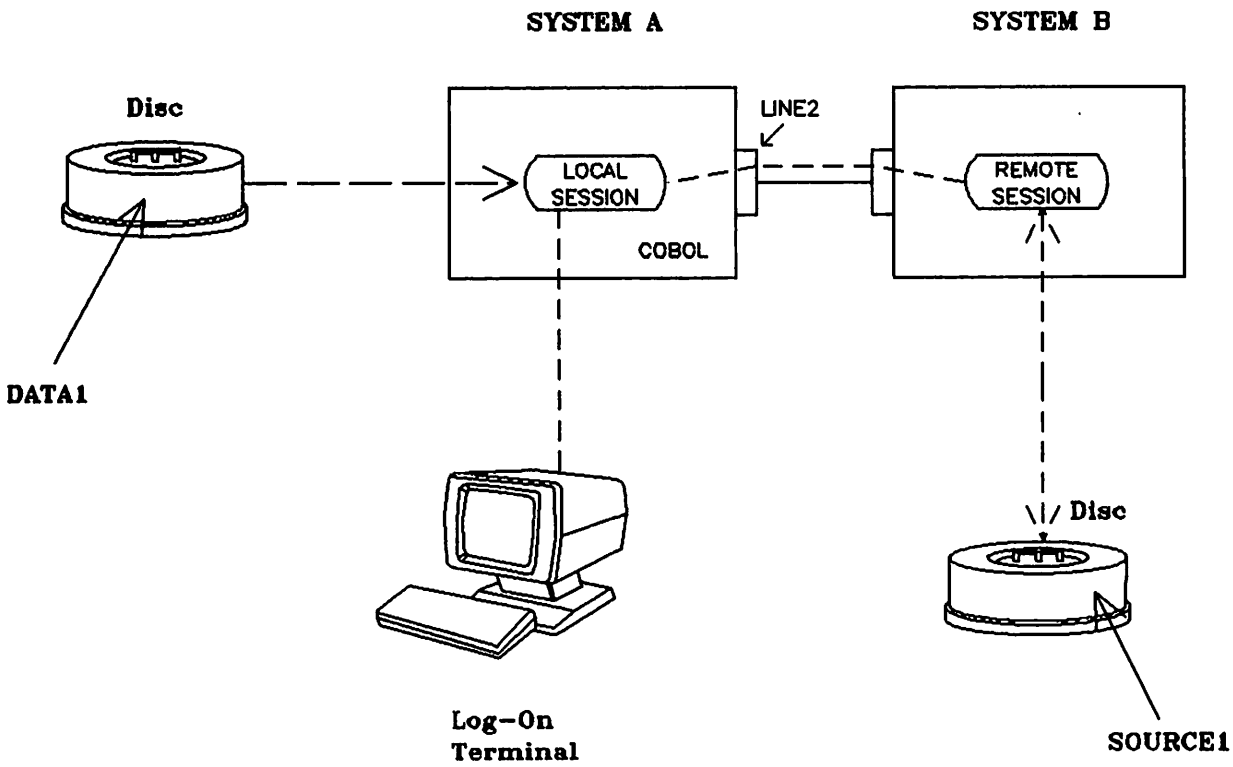


Figure 4-5. COBOL Remote File Access Example

NOTE

Many aspects of system resources (for example, memory size, CPU load, type of CPU load, time quantums, etc.) affect how COBOL and remote data base access activities are conducted in a DS/3000 environment. In general, it is more efficient to transfer data, USL, and SL files before or after (but not during) COBOL activity.

PROGRAMMATIC ACCESS

Once a DS/3000 communications link has been established between your HP 3000 and a remote HP 3000, you can use the standard set of MPE File System intrinsics within your local programs to access devices and/or files residing at the remote HP 3000 site. To make this possible, the format of the byte array referenced by the device parameter of the MPE FOPEN intrinsic was expanded to include a DS line specification in addition to the usual device specification. The format of the device byte array is as follows:

[*dsdevice*]#[*device*]

where *dsdevice* is the device class name, logical device number, or node name that you used when establishing the particular communications link (this specifies the physical line connecting the two computers) and *device* is the device class name or logical device number of the desired remote device as established within the remote HP 3000. (For a complete presentation of all FOPEN intrinsic parameters, refer to the *MPE Intrinsics Reference Manual.*)

NOTE

When the :FILE command is entered on a remote system to point back to a file on the local system, *dsdevice* is omitted.

The addition of *dsdevice#* to the format of the byte array referenced by the device parameter has enormously powerful implications. It means that programs executing in your local HP 3000 can easily access any of the devices and/or disc files of a remote HP 3000 as though they resided at your local HP 3000 site. Access to remote files is, of course, subject to the usual MPE file security. The user, account, and group names that you specified in the :REMOTE HELLO command when establishing the communications link are the ones used by MPE in the remote HP 3000 for determining your file access capabilities.

On the following pages, an annotated example illustrates remote device and file access from a local program running within a local session.

The Condition Codes for the various MPE File System intrinsics retain their normal meanings. Any communications line errors will return a CCL. In the event of an error, you can call the MPE FCHECK intrinsic to determine what happened. When using the MPE File System intrinsics for remote file

access, the Message Block B (File System) error codes apply to the remote file. You may also use the MPE `PRINTFILEINFO` intrinsic to display the status of a remote file.

The following program example tests RFA capabilities by writing to a remote file, reading the records back, then listing the contents of the remote file on the remote line printer.

Example

The following program illustrates how remote files can be accessed by using file system intrinsics.

```
$CONTROL USLINIT,ADR,MAP,CODE
 BEGIN
 INTEGER
    A,
    I:=-1,
    RDISCNUM,
    RLPNUM;

 BYTE ARRAY RMTLP'FILNAM(0:3):="RLP ";
 BYTE ARRAY RLPDEV(0:11);
 BYTE ARRAY RMTDISC'FILNAM(0:5):="RDISC ";
 BYTE ARRAY MSG(0:71);
 BYTE ARRAY RDISCDEV(0:11);

 LOGICAL ARRAY LMSG(*)=MSG;

 INTRINSIC PRINT,READ,FOPEN,FWRITEDIR,FREADDIR,FWRITE,FCLOSE;

<< User enters remote disc device class name into RDISCDEV. >>

 MOVE MSG:="INPUT REMOTE DISC DEVICE CLASS NAME ";
 PRINT(MSG,-35,0);
 MOVE MSG:="IN THE FORM .. DSDEVICE#DISCDEV ";
 PRINT(MSG,-31,0);
 A:=READ(LMSG,-12);
 MOVE RDISCDEV:=MSG,(A);

<< User enters remote LP device class name into RLPDEV.  >>

 MOVE MSG:="INPUT REMOTE LP DEVICE CLASS NAME ";
 PRINT(MSG,-33,0);
 MOVE MSG:="IN THE FORM .. DSDEVICE#LPDEV ";
 PRINT(MSG,-29,0);
 A:=READ(LMSG,-12);
 MOVE RLPDEV:=MSG,(A);

<< Open remote disc file. >>

 MOVE MSG:="OPENING REMOTE DISC FILE ";
 PRINT(MSG,-24,0);
 RDISCNUM:=FOPEN(RMTDISC'FILNAM,4,%104,-80,RDISCDEV);
 IF <> THEN
    BEGIN
       MOVE MSG:="COULD NOT OPEN REMOTE DISC FILE ";
       PRINT(MSG,-31,0);
```

```
      GO TO OUT;
   END;

<< Initialize remote disc file. >>

 MOVE MSG:="WRITING TO REMOTE DISC FILE ";
 PRINT(MSG,-27,0);
 MOVE MSG:=" ";
 MOVE MSG(1):=MSG(0),(71);

<< Write ten copies of MSG to remote disc file as a line check. >>

 WHILE (I:=I+1) <10 DO
    BEGIN
       MOVE MSG:="REMOTE FILE ACCESS TEST ";
       FWRITEDIR(RDISCNUM,LMSG,36,DOUBLE(I)); <<RECORD TO BE
                                               WRITTEN>>
       IF <> THEN
          BEGIN
             MOVE MSG:="ERROR WHEN WRITING TO REMOTE DISC ";
             PRINT (MSG,-33,0);
             GO TO OUT;
          END;
       END;

<< Initialize LP file. >>

 MOVE MSG:="OPENING REMOTE LP FILE ";
 PRINT(MSG,-22,0);
 RLPNUM:=FOPEN(RMTLP'FILNAM,4,1,,RLPDEV);
 IF <> THEN
    BEGIN
       MOVE MSG:="COULD NOT OPEN REMOTE LP FILE ";
       PRINT (MSG,-29,0);
    END;

<< Read 10 records from remote disc and write them to remote >>
<< LP, one at a time. >>

 I:=-1;
 WHILE (I:=I+ 1) < 10 DO
    BEGIN
       FREADDIR(RDISCNUM,LMSG,36,DOUBLE(I));
       IF <> THEN
          BEGIN
             MOVE MSG:="COULD NOT READ REMOTE DISC FILE ";
             PRINT(MSG,-31,0);
          END;
       FWRITE(RLPNUM,LMSG,36,0);
       IF <> THEN
          BEGIN
             MOVE MSG:="COULD NOT PRINT TO REMOTE LP FILE ";
             PRINT(MSG,-34,0);
```

```
            END;
        END;

    OUT;
    END.
```

USING A REMOTE DATA BASE

SECTION 5

TurboIMAGE is HP's data base management system. With DS/3000, you can use it to access data bases on other computer systems.

NOTE

All references to TurboIMAGE in this section apply equally to IMAGE/3000 if you are using either MPE IV or a version of MPE V prior to G.02.00. Also note that IMAGE/3000 and TurboIMAGE databases can access one another, but with some constraints. Please refer to the *TurboIMAGE Reference Manual* cited in the Preface of this manual.

If you want to access a data base that resides on one HP 3000 computer system while operating a session on another HP 3000 computer system, you may do so provided both systems have DS/3000 capability. You may use a data base on a remote HP 3000 either from a program that is running on the remote system or from a program running on your local HP 3000. The various ways to open a communications line and initiate a remote session are described in Sections 1 through 4. This section describes ways to access data bases programmatically with DS/3000. For example, you can establish a communications link and remote session and then run a remote program accessing a data base on the remote machine as illustrated in Figure 5-1.

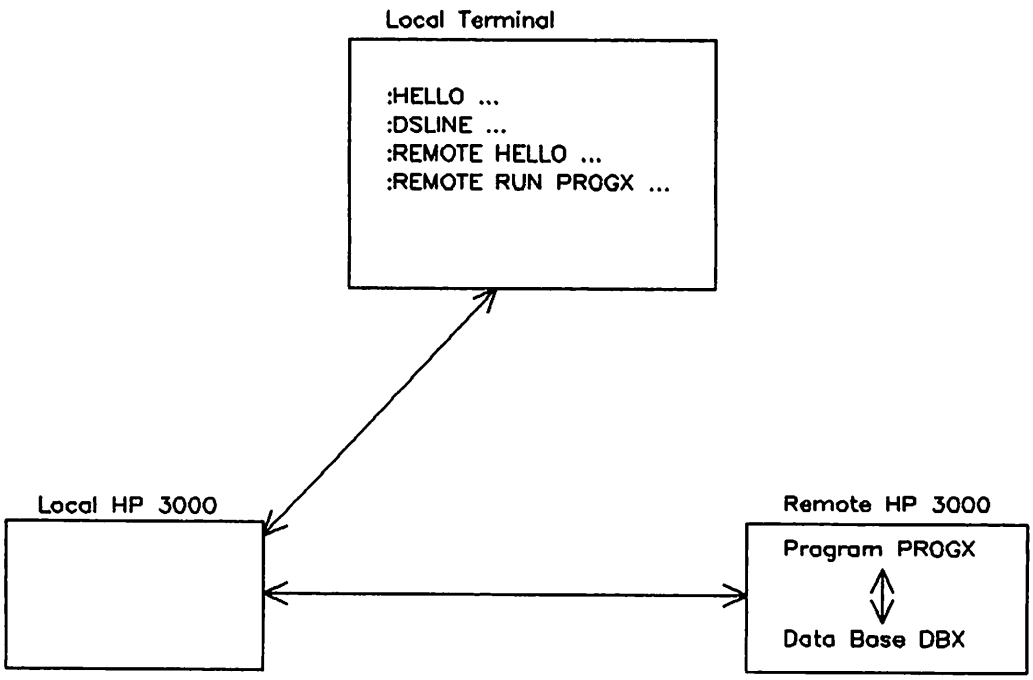


Figure 5-1. Using a Remote Program.

ACCESS THROUGH A LOCAL APPLICATION PROGRAM

If you want to access a remote data base using a local application program, there are three methods you may choose from. In all cases, a local program accesses a remote data base and the data is passed across the communications line.

Method 1: Establishing the Session Interactively

To use the first method, you interactively establish a communications link and a remote session and enter a `:FILE` equation for each remote data base. The `:FILE` equation specifies which data base is to be access on which remote system and device. A local application program can now access a remote data base, as shown in Figure 5-2.

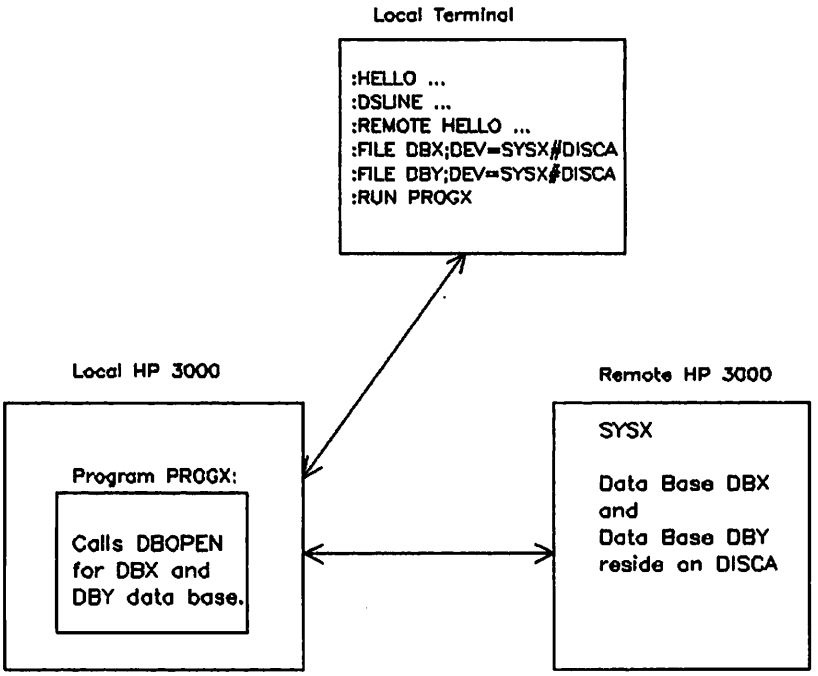


Figure 5-2. Using Method 1.

Method 2: Using the Command Intrinsic

The second method is very similar to the first, but you use the MPE command intrinsic within your application program to establish the communications link, remote session and remote data base access. In order to use this method in a COBOL, RPG, or BASIC application program, you must write a procedure in SPL or FORTRAN and call the procedure.

To use this method, you must issue a :REMOTE HELLO command (either as part of the :DSLINE parameter, or by issuing the :DSLINE as a seperate command) and a :FILE equation by calling the command intrinsic for each of these commands. The command intrinsic is explained in the *MPE Intrinsics Reference Manual*, and information about accessing remote files is given in Section 4. Figure 5-3 contains a diagram of Method 2.

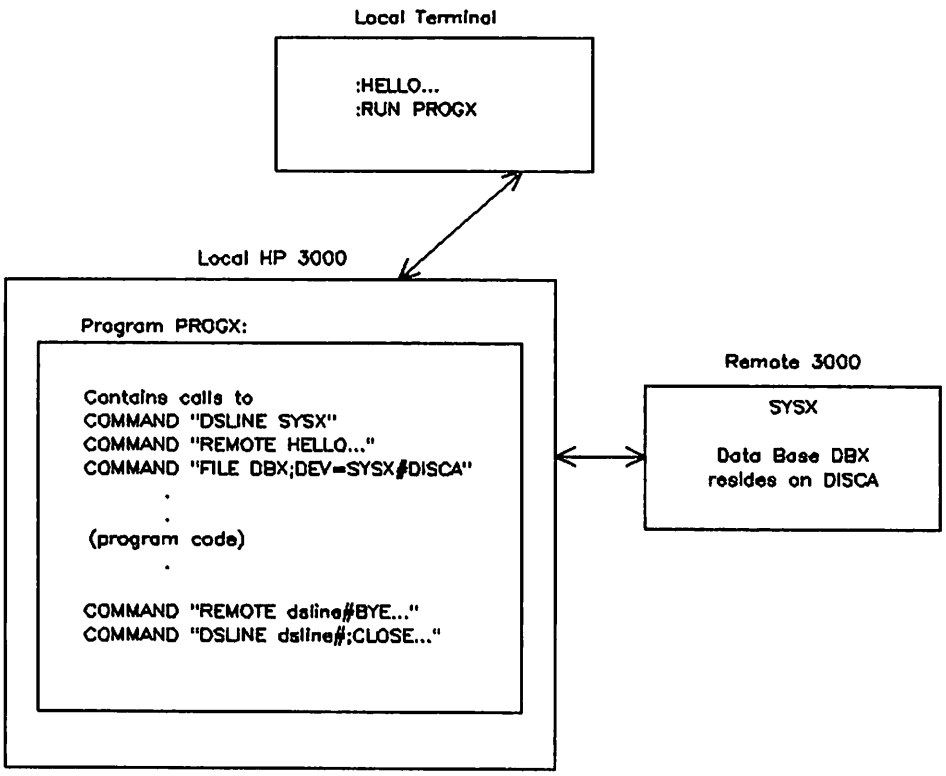


Figure 5-3. Using Method 2

If you want to access more than one remotely located data base with an application program, you must enter one :FILE equation for each remote data base. It is important to remember that a :REMOTE HELLO to the same remote computer should not be repeated within a process since the second request for a remote session would log off the first one.

When the application program calls the DBCLOSE procedure, or is ready to terminate execution, it must programmatically issue :REMOTE BYE and :DSLINE commands for the communications line specified with the foregoing command intrinsic. (Note that the application must also close all other files and devices it opened on the remote system, not just the database files, before the :REMOTE BYE and :DSLINE ;CLOSE commands.

If you use this method, any change in the data base name, account or password information requires modification of the application program. Since the application program maintains logical control over the commands that are issued, it is responsible for checking all status words returned by the remote system.

Method 3: Using a Data Base-Access File

The third method involves creating a special file which we shall call the data base-access file (DBA file). This file provides TurboIMAGE with the necessary information to establish a communications link and a remote session. It also specifies the remote data base or data base-access file name so that the necessary TurboIMAGE intrinsics can be executed on the remote computer.

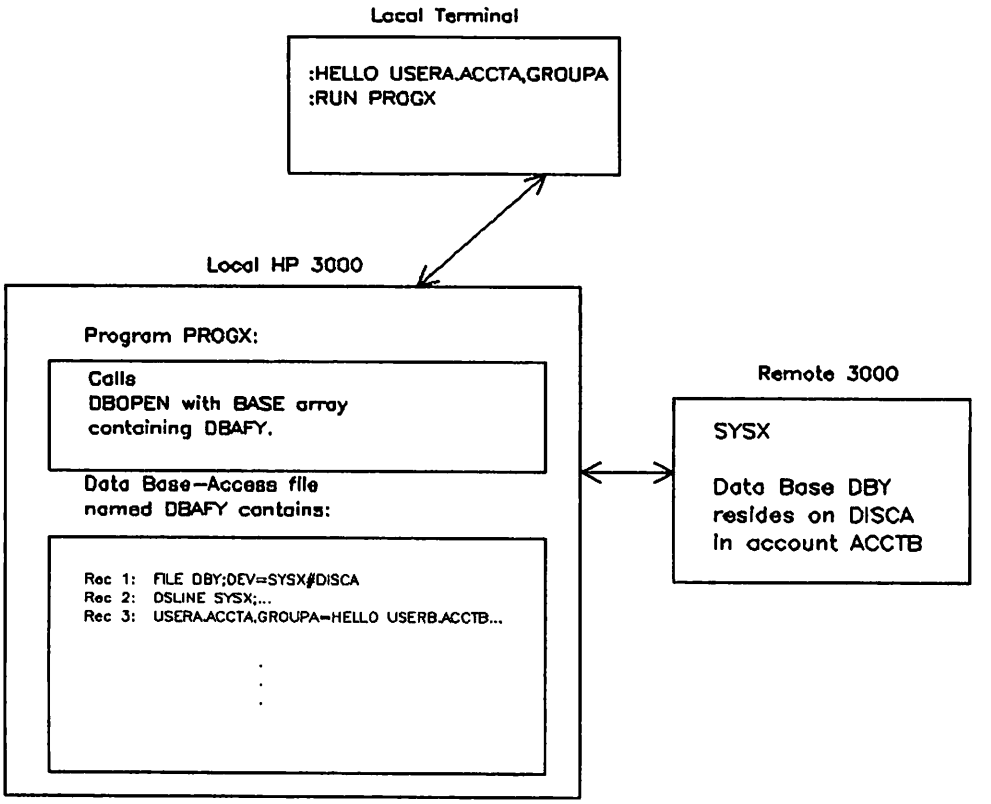


Figure 5-4. Using Method 3

NOTE

With Method 3, which uses the data base-access file, only one data base can be accessed using each data base-access file per :DSLINE. For example, if two computers are linked through two :DSLINEs, you can open one data base on each line. However, a second :REMOTE HELLO on either :DSLINE terminates the previous :REMOTE HELLO for that particular line. For multiple remote data base access, Method 1 or Method 2 is recomended. If the data base-access file is used, automatic :REMOTE BYE and :DSLINE ;CLOSE commands are issued on the communications line specified in the data base-access file when the application program terminates execution. Note that even though the :REMOTE BYE and DSLINE ;CLOSE commands happen automatically, the application (PROGX in this example) program should provide for closing all remote file and remote devices that might be open before the application ends.

By using this approach, the data base administrator can set up a user-table that provides more control over the data base users, and thus, enhances data base security. To create the data base-access file, use the Editor (Edit/3000). First use the :SET LENGTH command to accommodate the largest record to be included in the data base-access file, up to a maximum of 128 characters. The content of this file should be created in the format shown below.

SYNTAX

Record 1 :FILE *dbname1*[=*dbname2*]; DEV=*dsdevice#*[DISC]

Record 2 :DSLINE *dsdevice* [;LINEBUF=*buffer-size*] [;LOCID=*local-id-sequence*] [;REMID=*remote-id-sequence*] [;PHNUM=*telephone-number*] [;EXCLUSIVE] [;QUIET]

Record 3 *lusername*.*lacctname*[,*lgroupname*]=HELLO *rusername* [/*rupasw*] *racctname*[/*rapasw*] [,*rgroupname*[/*rgpasw*]][;TIME=*cpusecs*] [;PRI=*priority*] [;HIPRI / ;INPRI=*inputpriority*]

Records 4 through n Same format as Record 3. Specifies other user.account,group identification.

PARAMETERS

dbname1 is the name of the data base or the data base-access file on the remote system you want to access, or is the formal file designator used in the program if *dbname2* is specified.

dbname2 is the name of the data base or the data base-access file on the remote system you want to access.

dsdevice is the device class name or logical device number assigned to the DS/3000 communications driver (IODS0) during system configuration, or a logical node name associated with the X.25 Link communications device (IODSX) during network configuration using NETCONF.

buffer-size is a decimal integer specifying the size (in words) of the DS/3000 line buffer to be used in conjunction with the communication line. The integer must be within the range 304 <*buffer-size*<4096. The default value is the buffer size entered in response to the PREFERRED BUFFER SIZE prompt during system configuration.

local-id-sequence is a string of ASCII characters contained within quotations marks. If you wish to use a quotation mark within an ASCII string, use two successive quotation marks. The maximum number of ASCII characters allowed in the string is 16.

The supplied string of ASCII characters defines the ID sequence that will be sent from your HP 3000 to the remote HP 3000 when you attempt to establish the telephone connection. If the remote HP 3000 does not recognize the supplied ID sequence as a valid one, the telephone connection is terminated. The default value is the ASCII string entered in response to the LOCAL ID SEQUENCE prompt during system configuration.

remote-id-sequence Same format as *local-id-sequence*.

The supplied string of ASCII characters defines those remote HP 3000 ID sequences that will be considered valid when you attempt to establish the telephone connection. If the remote HP 3000 does not send a valid ID sequence, the telephone connection is terminated. The default set of remote ID sequences consists of the ASCII strings entered in response to the REMOTE ID SEQUENCE prompt during system configuration.

telephone-number is a telephone number consisting of digits and dashes. The maximum length permitted (including both digits and dashes) is 30 characters. If YES was entered in response to the DIAL FACILITY prompt during system configuration, this telephone number will be displayed at the operator's console of your HP 3000 and the operator will then establish the telephone connection by dialing that number at the modem. The default telephone number is the first one entered in response to the PHONE NUMBER prompt during system configuration.

EXCLUSIVE specifies that you want exclusive use of the particular communications line. If the specified HSI, SSLC or INP is already open and you have specified the exclusive option, DS/3000 will deny you access to the line (you cannot open it). Opening an EXCLUSIVE line requires the user to have CS capability. This capability may be granted by a system manager or account manager.

QUIET specifies that the message identifying the DS line number will be suppressed. The messages associated with subsequent :REMOTE HELLO and :REMOTE BYE commands will also be suppressed. In this case, the terminal operator is totally unaware that remote processing is taking place.

lusername is a user name on the local HP 3000, as established by an account manager, that allows you to log on under this account. This name is unique within the account. It contains from 1 to 8 alphanumeric characters, beginning with a letter. An at sign (@) may be used to indicate the log on user name.

lacctname is the name of your account on the local HP 3000 as established by a system manager. It contains 1 to 8 alphanumeric characters, beginning with a letter. An at sign (@) may be used to indicate the log on account.

lgroupname is the name of a file group to be used for the local file domain and central processor time charges, as established by an account manager. It contains

from 1 to 8 alphanumeric charcters, beginning with a letter. An at sign (@) may be used to indicate the log on group.

rusername is a user name on the remote HP 3000 that allows you to log on under the remote account. It follows the same rules as *username*. An at sign (@) may be used to indicate *rusername* as with *lusername*.

racctname is the name of the log on account on the remote HP 3000. It follows the same rules as *lacctname*. An at sign (@) may be used to indicate *racctname* is the same as *lacctname*.

rgroupname is the name of the log on group on the remote HP 3000. It follows the same rules as *lgroupname*. An at sign (@) may be used to indicate *rgroupname* is same as *lgroupname*.

rupasw is the password assigned to *rusername*.

rapasw is the password assigned to *racctname*.

rgpasw is the password asigned to *rgroupname*.

TIME=*cpusecs* is the maximum central processor time that your remote session can use, entered in seconds. When this limit is reached, the remote session is aborted. It must be a value from 1 to 32767. To specify no limit, enter a question mark or omit this parameter. Default: No limit.

PRI={BS, CS, DS, ES} is the execution priority class that the Command Intepreter uses for your remote session, and also the default priority for all programs executed within the remote session. BS is highest priority; ES is lowest. If you specify a priority that exceeds the highest that the system permits for *racctname* or *rusername*, MPE assigns the highest priority possible below BS. Default: CS.

NOTE

DS and ES are intended primarily for batch jobs; their use for sessions is generally discouraged.

INPRI=*inputpriority* is the relative input priority used in checking against access restrictions imposed by the job fence, if one exists. It takes effect at log on time. It must be a value from 1 (lowest) to 13 (highest priority). If you supply a value less than or equal to the current job fence set by the console operator, the session is denied access. Default: 8 if logging of session/job initiation is

enabled, 13 otherwise.

HIPRI is a request for maximum session-selection input priority, causing the remote session to be scheduled regardless of the current job fence or execution limit for sessions.

NOTE

You can specify this only if you have system manager or supervisor capability. (Optional parameter)

Syntax Considerations

The following syntax should be noted:

- No spaces are allowed around the periods in the optional file reference, or separating *dsdevice* and the # sign, in Record 1.
- Passwords are not allowed with the local user, account, and group names. They are not neccessary since the local user passes the security password checks when logging on the local session.

NOTE

Remote logon parameters must define a valid logon known to the remote machine. For example, if a particular user name requires a password on the remote machine, the password parameter is not optional in the data base-access file and must be supplied in the :HELLO command.

USER IDENTIFICATION.

Records 3 through n in a DBA file tell TurboIMAGE which user, account, and group names on the local computer may access which user, account, and group names on the remote computer. You may specify remote user identification for more than one local user by creating a record for each local `user.account,group` in the format of Record 3 shown above. An at sign (@) may be substituted for any user, account, or group name in the record. If an at sign is substituted for *lusername*, *lacctname*, or *lgroupname*, the name is replaced with the corresponding name specified at log on time.

When a local user runs a program that DBOPEN the DBA file, TurboIMAGE searches for a match between the local user, account and group names in the DBA file and the names the user entered when logging on to the local session. When a match is found, TurboIMAGE performs the `:FILE` and `:DSLINE` commands, and a `:REMOTE HELLO` using the corresponding *rusername*, *racctname*, *rgroupname*, and passwords if present. If an at sign is found, it is replaced with the corresponding name to the left of =HELLO. For example, if the record contains `USERA.ACCTA,GROUPA=HELLO @.ACCTB,@`, TurboIMAGE replaces the first at sign with USERA and the second with GROUPA. If an at sign is not found, no substitutions are made. In either case, the information to the right of =HELLO is used as the remote log on identification.

EXAMPLE. The following syntax should be noted:

```
Record 1  FILE STORE;DEV=DSL1#DISC

Record 2  DSLINE DSL1

Record 3  USERA.ACCTA,GROUPA=HELLO USERB.ACCTA,GROUPB

Record 4  @.ACCTA,GROUPA=HELLO USERA.ACCTA,GROUPA

Record 5  USERB.ACCTB,@=HELLO USERB.ACCTX,@

End of file
```

If a user logs on with the log on identification indicated in the first column below, TurboIMAGE will use the corresponding `user.acct,group` identification in the second column to establish communication with the remote system.

Log-on Identification	Remote Identification Used
User1 USERA.ACCTA,GROUPA	USERB.ACCTA,GROUPB
User2 USERB.ACCTA,GROUPA	USERA.ACCTA,GROUPA
User3 USERB.ACCTB,GROUPB	USERB.ACCTX,GROUPB
User4 USERA.ACCTB,GROUPB	None, no match found.

The first user's log-on identification matches the local user, account, and group names specified in Record 3, so the remote names specified in that record are used. The second user's account matches Record 3 but the user name does not, so TurboIMAGE looks for another table entry with account ACCTA. Since the entry in Record 4 specifies any user (@) of ACCTA if their group is GROUPA, the second user's remote identification will be that specified in Record 4.

The third user logs on to ACCTB and a match is found in Record 5, since it specifies the same user name and accepts any group in the account.

The fourth user's account matches Record 5 but the user name does not match. Therefore, the fourth user cannot access the remote data base with this application program.

FILENAME.

After you have created the file with the Editor, you should KEEP it UNNumbered. The file name must follow the same rules as a data base name. It must be an alphanumeric string from 1 to 6 characters, the first character must be alphabetic.

ACTIVATING A DATA BASE-ACCESS FILE.

After you have constructed a data base-access file, you must use the DBUTIL utility program to activate the file. The complete syntax for running the utility program is given in the *TurboIMAGE Reference Manual.* Here is a summary of the operating instructions:

```
:RUN DBUTIL.PUB.SYS
        .
        .
>>ACTIVATE data base-access file name

Verification follows:

    FILE command:     <result>
    DSLINE command:   <result>
    HELLO command:    <result>

ACTIVATED

>>EXIT
```

DBUTIL verifies that the file to be activated:

- has a file code of zero
- is an UNNUMBERED, ASCII file
- has a record length <=128 characters
- has at least three records.

If any of these conditions is not satisfied, activation fails. If all of the above are satisfied, DBUTIL prints the following message:

```
Verification Follows:
```

Then the utility program verifies the syntax of:

- Record 1
- Record 2 through *dsdevice*, which must be identical to the *dsdevice* specified in Record 1
- Records 3 through *n*, through the parameter *rgpasw*.

This means that for Records 2 through *n* only the positional parameters (those whose function is determined by their relative position within the command) are verified by DBUTIL. The remaining key word parameters are checked by the command interpreter at DBOPEN time.

If all of the above conditions are met, DBUTIL successfully activates the data base-access file, by changing the file code to the TurboIMAGE reserved code -402. The example in Figure 5-5 illustrates how to create and activate a data base-access file. In this case, the file named DSASTR is to be used to gain access to the STORE data base residing on a remote system in the PAYACCT account. The remote system is referenced by dsdevice name MY.

After the data base-access file is created using the Editor, it is enabled by using the DBUTIL utility program.

```
HELLO MEMBER1.PAYACCT                                   Log on to the local
          .                                             system.
          .
:EDITOR                                                 Run the Editor.

HP32201A.07.17 EDIT/3000  THU, OCT 3, 1985  2:37 PM
(C)  HEWLETT-PACKARD CO.  1985
/A                                                      Create records in data
      1     FILE STORE;DEV=MY#                          base-access file.
      2     DSLINE MY
      3     MEMBER1.PAYACCT=HELLO MEMBER1.PAYACCT
      4     MEMBER2.PAYACCT=HELLO @.PAYACCT
      5     //
...
/K DBASTR,UNN                                           Keep the file.
/E

END OF SUBSYSTEM
:RUN DBUTIL.PUB.SYS                                     Run DBUTIL to enable
>>ACTIVATE DBASTR                                       file.
  Verification follows:
    FILE command:     Looks good
    DSLINE command:   Looks good
    HELLO command:    Looks good
    HELLO command:    Looks good
  ACTIVATED
>>EXIT

END OF PROGRAM
```

Figure 5-5. Preparing a Data Base-Access File.

ACCESSING DATA BASES.

To reference the data base from your local application program, use the data base-access file name instead of the root file name when calling the TurboIMAGE procedure. The word array specified as the *base* parameter must contain a pair of blanks followed by the left-justified data base-access file name and terminated by a semicolon or blank. TurboIMAGE recognizes the -402 file code and establishes a communications link to the remote HP 3000. If the data base is succesfully opened, TurboIMAGE replaces the pair of blanks with the extra data segment number of the assigned Remote Data Base Control Block. The *base* parameter must remain unchanged for the remainder of the process. When the application program calls the DBCLOSE procedure or terminates execution, automatic :REMOTE BYE and :DSLINE commands are issued to terminate the session and close the communications line.

Figure 5-6 illustrates use of the data base-access file through a program named APPLICAN. After logging on to the local system, the user runs the program named APPLICAN from the local session. The *base* array in this program contains " DBASTR". When a call to DBOPEN is executed, TurboIMAGE establishes a communication line and remote session. When the program closes the data base, TurboIMAGE closes the line and terminates the remote session.

```
:HELLO MEMBER2.PAYACCT                                  Log on to local
          .                                             system.
          .
:RUN APPLICAN                                           Execute application
                                                        program.
DS LINE NUMBER = #L4
HP3000 IIB.   MON, OCT 28, 1985,  1:56 PM               TurboIMAGE establishes
                                                        a communications line
WELCOME TO SYSTEM B.                                    and remote session.

CPU=2.  CONNECT=1. MON, OCT 28, 1985, 1:59 PM           When the data base is
1 DS LINE WAS CLOSED                                    closed, TurboIMAGE
                                                        closes the line and
:BYE                                                    terminates remote
                                                        session.

                                                        Log off local system.
```

Figure 5-6. Using a Data Base-Access File.

QUERY/3000

When you run QUERY/3000, which is accessed with RUN QUERY.PUB.SYS, you can specify the data base to be used in two ways. You can use the DEFINE command, which then prompts you for:

DATA-BASE,
PASSWORD,
MODE (see Table 5-1 for available modes),
DATA-SETS to be accessed,
PROC-FILE, which stores FIND, REPORT, and UPDATE commands as procedures,
OUTPUT, which specifies the output device.

You can also specify these options individually, as QUERY/3000 commands.

The DATA BASE= prompt can be answered with a remote data base name or the data base-access file name. Note, however, that perfomance can be significantly improved if you run QUERY/3000 in remote session, thereby accessing the data base on the system where it resides, rather than running QUERY/3000 locally to access a remote data base.

Table 5-1. Modes of Access

If your mode is:	1	you may:	*find* (read), *replace*, *add*, and *delete* entries. (QUERY/3000 requests TurboIMAGE to lock and unlock the data base dynamically when accessing it.)
	2		*find* and *replace* entries.
	3* or 4		*find*, *replace*, *add*, and *replace* entries.
	5		*find* entries. (QUERY/3000 locks and unlocks.)
	6, 7*, or 8		*find* entries.

* These modes give you exclusive access to the data base. All other modes allow others to share the data base. Search and sort items cannot be replaced.

PROGRAM-TO-PROGRAM COMMUNICATIONS

SECTION 6

In the preceding sections, you have seen how you can establish communications links between several HP 3000 computers to form a telecommunications network and how you can execute programs in any of the HP 3000s from a single log-on terminal. Furthermore, you have seen that programs running within any HP 3000 in the network can, under the proper circumstances, obtain access to any of the hardware or software resources available throughout the network. At this point, you already have a powerful telecommunications network capability at your disposal.

But for many teleprocessing applications, it is essential that separate user programs be able to be run simultaneously in separate computers within the network and that they be able to communicate efficiently with one another.

Two capabilities answer that need: DS/3000 Program-to-Program (PTOP) Communications (described in this section) and MPE Interprocess Communications (IPC) (described in Section 8).

You might ask, "Why can't the normal process-handling capabilities of MPE be used for this purpose?" As you probably recall, the process-handling capabilities of MPE permit a user process (referred to as the "father" process), to create and activate one or more "son" processes that then run concurrently with the father process. Father and son processes can communicate efficiently with one another through the use of the SENDMAIL intrinsics, shared extra data segments, or a shared user file. Unfortunately, however, the process-handling capabilities of MPE were designed for use within a single processor. They cannot handle the intervention of a communications line between father and son processes.

Suppose you were to log on to an HP 3000, gain access to a DS line, and initiate a remote session. Within the local session, you use a STREAM command to initiate the execution of a program named PROGA; and within the remote session, you use a STREAM command to initiate execution of a program named PROGB. You now have two programs executing simultaneously: PROGA in your local HP 3000 and PROGB in the remote HP 3000.

At this point, the two programs are entirely independent of one another: neither knows the other exists. If you add a shared access disc file to the situation, PROGA and PROGB can now read from and write to that file, and thereby communicate indirectly with one another. This arrangement works well as long as the data being deposited in the shared file does not have to be retrieved, processed, and responded to within a finite period of time.

There are teleprocessing applications where this type of arrangement is not only adequate but makes a great deal of sense. For example, consider the case where a branch office is accumulating information that must be merged once a day into a data base residing at the main office. In this case, the two programs can make very effective use of the message file approach that IPC uses (see Section 8 for a description).

As soon as an application tries to be truly interactive, however, this arrangement falters because the two programs cannot communicate directly. Each must know whether or not the other program is trying to transmit data. The more dependent each program is upon receiving data from the other, the more likely it is that PTOP should be used for the application.

With the remote file access method of program-to-program communication (IPC), the two programs have no way of knowing if the other program is actually executing. With the POPEN intrinsic, the master program knows that the slave program is executing, because it created and activated the slave

program's process. Likewise, the slave program knows that the master program is executing, because without an active corresponding master program, the slave itself would not be executing.

The DS/3000 program-to-program communications facility provides nine intrinsics that make it possible for two or more user programs residing in separate HP 3000s to exchange data and control information directly (and efficiently) over DS/3000 communications links.

The nature of any two programs that are communicating with one another in this manner is not symmetrical. One of them (referred to as the "master" program) is always in control and is the one that initiates all activity between the two programs. The other (referred to as a "slave" program) always responds to requests received from the master. Those intrinsics used within a master program are summarized in Table 6-1, and those used within a slave program are summarized in Table 6-2.

Table 6-1. Master Program-to-Program Intrinsics.

Intrinsic Name	Function
POPEN	Initiates and activates a slave process in a remote HP 3000 and initiates program-to-program communication with the slave program.
PREAD	Sends a read request to the remote slave program asking the slave to send a block of data back to the master.
PWRITE	Sends a block of data to the remote slave program.
PCONTROL	Transmits a tag field (containing user-defined control information) to the remote slave program and receives a tag field back from the slave.
PCLOSE	Terminates (kills) the remote slave program's process.
PCHECK	Returns an integer code specifying the completion status of the most recently executed master program-to-program intrinsic.

Table 6-2. Slave Program-to-Program Intrinsics.

Intrinsic Name	Function
GET	Receives the next request from the remote master program.
ACCEPT	Accepts (and completes) the request received by the preceding GET intrinsic call.
REJECT	Rejects the request received by the preceding GET intrinsic call.
PCHECK	Returns an integer code specifying the completion status of the most recently executed slave program-to-program intrinsic.

Conceptually, the DS/3000 program-to-program intrinsics are very similar to the MPE process handling and file system intrinsics that are used for process-to-process communication within a single-system environment. Table 6-3 compares the intrinsics used for process-to-process communication within a single-system environment to those used for program-to-program communication within a distributed systems environment.

Table 6-3. Single System / Distributed Systems Comparison.

Function	Single System (Process Handling)	Distributed Systems (Program-to-Program)
Initiate another process.	CREATE ACTIVATE	POPEN
Communicate with the other process.	Mail Intrinsics: SENDMAIL RECEIVEMAIL . . . User Managed Extra Data Segment GETDSEG DMOVEIN DMOVEOUT . . . Shared User File: FOPEN FREAD FWRITE FCONTROL FCLOSE FCHECK . . .	Master (father) Requests: PREAD PWRITE PCONTROL PCHECK Slave (son) Responses: GET ACCEPT REJECT PCHECK
Terminate the other process.	Father: KILL (a son) TERMINATE (self and all sons)	Master (father): PCLOSE (a slave) TERMINATE (self and all slaves)

When a DS/3000 communications link exists between two HP 3000s, a user program (the master program) can create and activate a son process (a slave program) in the remote HP 3000. The POPEN intrinsic performs this function, in place of the standard MPE CREATE and ACTIVATE intrinsics.

After the master and slave programs are both executing, the master program can:

- Send data (PWRITE) or control information (PCONTROL) directly to the slave program
- Send a read request (PREAD) or control request (PCONTROL) to the slave program asking that the slave send data or control information back to the master
- Check status (PCHECK) and terminate (PCLOSE) a slave program.

Notice the striking similarity between this method of communication and the use of the MPE File System intrinsics FREAD and FWRITE. It is as though the master program is reading from or writing to a file -- a very intelligent file that is capable of making decisions, controlling input/output devices, and performing productive processing.

PTOP INTRINSICS

The following pages contain detailed descriptions of the PTOP intrinsics that were summarized in Tables 6-1 and 6-2. For convenience in locating specific items of information in this reference section, these detailed descriptions are presented in a format consistant with that used in the *MPE Intrinsics Reference Manual.* Also, since this part of the section will be used for repeated reference, the intrinsics are arranged in alphabetical sequence, rather than in the order of normal usage as they were presented in the summary tables.

These PTOP intrinsics are callable from SPL, FORTRAN, Pascal, or COBOL II. To call a DS/3000 PTOP intrinsic from a program, use the following procedure:

1. Refer to the intrinsic description to determine the parameter types and their positions in the parameter list.
2. Declare the variables or array names to be passed as parameters by type at the beginning of the program.
3. Include the name of the PTOP intrinsic in an INTRINSIC declaration statement, or the language's equivalent.
4. Issue the intrinsic call at the appropriate place in your program.

ACCEPT

(Slave callable)

Accepts (and completes) the requests received by the preceding GET intrinsic call and returns an optional tag field to the remote master program.

SYNTAX

```
ACCEPT([itag][,target][,tcount]);
```

PARAMETERS

itag — integer array

A twenty-word array used for transmitting a tag field. The format of the tag field is defined by the user's master and slave programs.

target — integer array

An array used for transmitting or receiving blocks of data.

For PREAD requests, this array contains the block of data to be transmitted to the master program.

For PWRITE requests, this array receives the block of data from the DS/3000 buffer.

For POPEN and PCONTROL requests, this parameter has no meaning and should be omitted.

tcount — integer by value

An integer specifying the number of words (if positive) or bytes (if negative) to be transmitted or received.

For PREAD requests, this parameter specifies how much data is to be transmitted from *target* to the master program.

For PWRITE requests, this parameter specifies how much data is to be moved from the DS/3000 buffer to *target*.

For POPEN and PCONTROL requests, this parameter has no meaning and should be omitted.

CONDITION CODES

CCE	Request completed successfully.
CCG	(Not returned.)
CCL	An error occured. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The ACCEPT intrinsic accepts the request received by the most recent GET intrinsic call, completes the requested operation, and transmits an optional tag field back to the remote master program.

In the case of a POPEN request, the ACCEPT call transmits an optional tag field (*itag*) to the remote master program.

In the case of a PREAD request, the ACCEPT call transmits the specified number of words or bytes (*tcount*) from *target* to the master program and transmits an optional tag field (*itag*) to the master program.

In the case of a PWRITE request, the ACCEPT call moves the specified number of words or bytes (*tcount*) from the DS/3000 buffer to *target* and transmits an optional tag field (*itag*) to the master program.

In the case of a PCONTROL request, the ACCEPT call transmits an optional tag field (*itag*) to the remote master program.

GET

(Slave callable)

Receives the next request from the remote master program.

SYNTAX

```
ifun:=GET([itag][,il][,ionumber]);
```

FUNCTIONAL RETURN

ifun integer

0 An error occured. This value is returned only when the condition code CCL is also returned. Issue a PCHECK intrinsic call (with a *dsnum* parameter of zero) to determine what happened.

1 POPEN request received.

2 PREAD request received.

3 PWRITE request received.

4 PCONTROL request received.

5 This value is returned only when the condition code CCG is also returned. It indicates that a pending MPE File System I/O without wait request was completed (instead of the expected remote DS/3000 I/O request). *ionumber* contains the file number associated with the completed I/O request.

PARAMETERS

itag integer array

A twenty-word array used for receiving a tag field. The format of the tag field is defined by the master and slave programs.

il integer

A word that has meaning only when a PREAD or PWRITE request is received from the master program.

For a PREAD request, *il* contains an integer specifying the number of words or bytes requested by the master program.

For a PWRITE request, *il* contains an integer specifying the number of words or bytes transmitted from the master program to the DS/3000 buffer on the remote system.

ionumber integer

A word that has meaning only when the condition code CCG and an *ifun* of 5 are returned. In that case, *ionumber* contains the MPE File System file number associated with the completed I/O without wait request.

Default: No file number is returned.

CONDITION CODES

CCE Request received successfully.

CCG The implicit IOWAIT(0) call issued by the GET intrinsic completed a pending remote MPE File System I/O without wait request instead of a remote DS/3000 I/O request. *ionumber* contains the file number associated with the completed File System request.

CCL An error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The GET intrinsic receives the next request from a local master program and accepts an optional tag field (available in *itag*). The GET intrinsic call implicitly issues an IOWAIT(0) intrinsic call. An *ifun* of 0 indicates that an IOWAIT error occurred. An *ifun* of 5 will occur only if you are executing MPE File System intrinsic calls without wait in your remote program and the implicit IOWAIT(0) call completes a pending File System I/O request instead of the expected DS/3000 I/O request (in this case, you will have to issue another GET call after processing the completed File System I/O request in order to receive the expected DS/3000 I/O request).

NOTE

You must not use IOWAIT calls within a program containing DS/3000 GET calls. If you were to use an IOWAIT(0) call and it responded to a DS/3000 I/O request, your program would not be able to make any sense out of the information returned by the IOWAIT call.

PCHECK

(Slave and Master callable)

Returns an integer code specifying the completion status of the most recently executed DS/3000 program-to-program intrinsic.

SYNTAX

```
icode:=PCHECK(dsnum);
```

FUNCTIONAL RETURN

When the PCHECK intrinsic executes, it returns to the calling program a number (*icode*) that specifies the completion status of the most recently executed DS/3000 program-to-program intrinsic. The various values of *icode* are shown in Appendix A under the heading "DS/3000 Functional Errors."

PARAMETERS

dsnum	integer by value	
	MASTER PROGRAM:	The link identifier returned by the particular POPEN intrinsic that initiated communication with the remote slave program.
	SLAVE PROGRAM:	0 (zero); no link identifier is returned to a slave program.

CONDITION CODES

CCE	PCHECK request successfully completed.
CCG	(Not returned.)
CCL	PCHECK request denied because *dsnum* was invalid.

OPERATION

The PCHECK intrinsic returns an integer value that specifies the completion status of the most recently executed DS/3000 program-to-program intrinsic.

(Master callable)

Terminates program-to-program communication with a remote slave program.

SYNTAX

```
PCLOSE(dsnum);
```

PARAMETERS

dsnum integer by value

The line number returned by the particular POPEN intrinsic call which initiated communication with the remote slave program.

CONDITION CODES

CCE Successful completion.

CCG (Not returned.)

CCL Request denied; an error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The PCLOSE intrinsic terminates the remote slave program associated with *dsnum*. The particular communications line remains open.

PCONTROL

(Master callable) Exchanges tag fields with the remote slave program.

SYNTAX

```
PCONTROL(dsnum[,itag]);
```

PARAMETERS

dsnum integer by value

The link identifier returned by the particular POPEN intrinsic call which initiated communication with the remote slave program.

itag integer array

A twenty-word array used for transmitting and receiving a tag field. The format of the tag field is defined by the master and slave programs and may serve any purpose you desire.

CONDITION CODES

CCE Request accepted by remote slave program.

CCG Request rejected by remote slave program.

CCL Request denied; an error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The PCONTROL intrinsic transmits a tag field to the remote slave program and accepts one in return. The remote slave program must issue a GET intrinsic call followed by either an ACCEPT or REJECT call to complete the PCONTROL operation. Both the ACCEPT and REJECT calls transmit a tag field back to the master program.

Although this intrinsic was designed specifically for the exchanging of tag fields, you will notice that *itag* is an optional parameter (it is also optional for the ACCEPT and REJECT slave program-to-program calls). If the master program did not transmit a tag field, the returned tag field (if any) is not accessible.

The PCONTROL activity is illustrated in Figure 6-1.

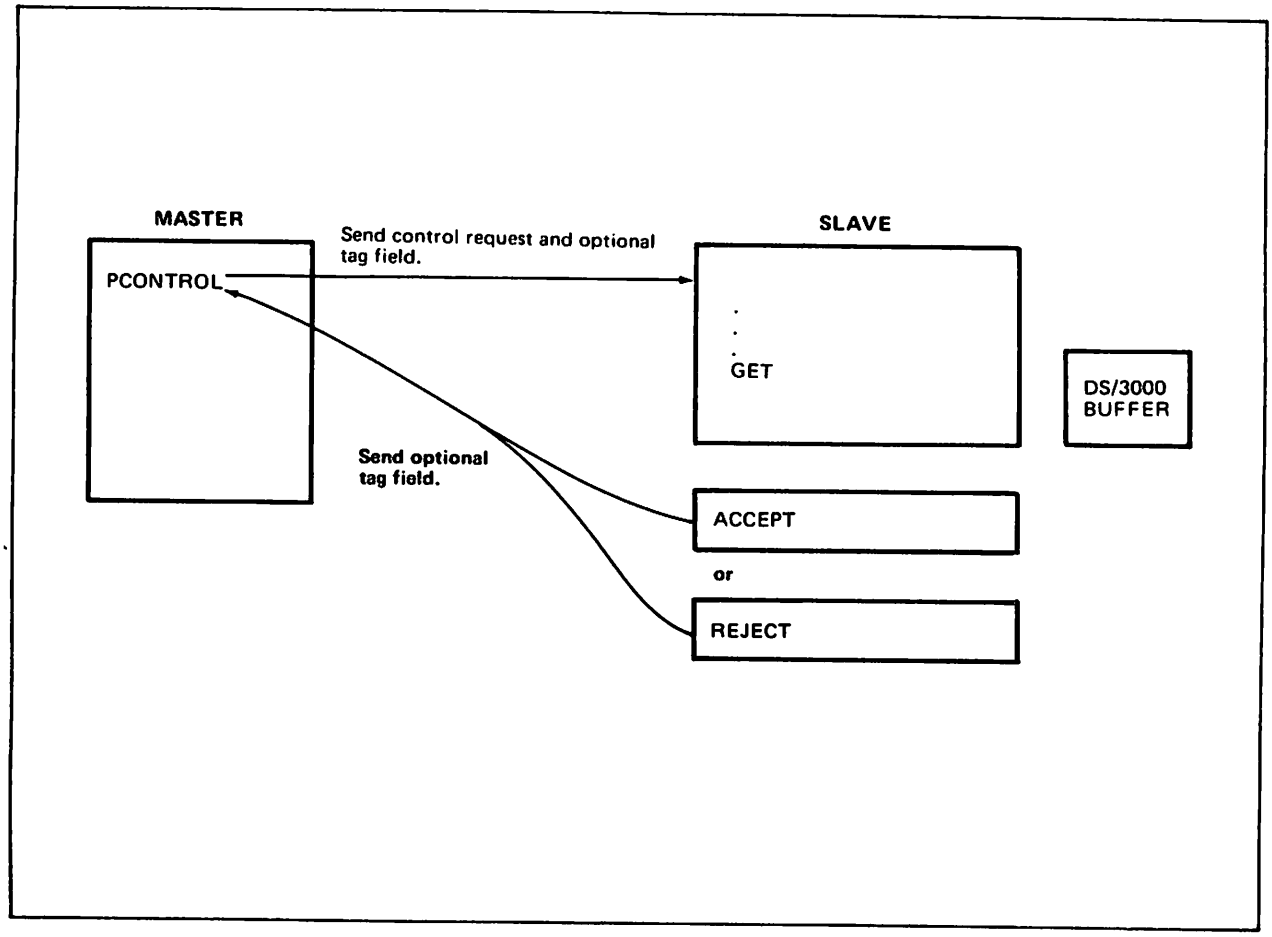


Figure 6-1. PCONTROL Activity.

POPEN

(Master callable)

Initiates program-to-program communication with a remote slave program.

SYNTAX

```
dsnum:=POPEN(dsdevice,progname[,itag][,entryname][,param]
             [,flags][,stacksize][,dlsize][,maxdata][,bufsize])
```

FUNCTIONAL RETURN

When the POPEN intrinsic executes, it returns to the master program a number (*dsnum*) by which DS/3000 uniquely identifies the particular communications link. This number is analgous to the file number returned by the MPE FOPEN intrinsic in that it is used in all subsequent master program-to-program intrinsic calls to reference the remote slave program.

PARAMETERS

dsdevice byte array

Contains a string of ASCII characters terminated by a space. This string must be the device class name, logical device number, or node name used in the :DSLINE or :REMOTE HELLO command that opened the communications line you will be using.

progname byte array

Contains a string of ASCII characters terminated by a space. This string is the name (with optional group and account names) of an MPE program file (residing on a disc connected to the remote HP 3000) containing the remote slave program.

itag integer array

A twenty-word array that is used for transmitting and receiving tag fields. The format of the tag field is defined as part of the user's application.

Default: A tag field of all zeros is sent; the returned tag field (if any) is not available to the master program.

entryname byte array

Contains a string of ASCII characters terminated by a space. This string is the name of the entry point (label) at which execution of the remote slave program is to begin.

Default: Primary entry point.

param integer by value

A word used to transfer control information to the new (remote) process. Any instruction in the outer block of code in the new process can access this information in location Q-4.

Default: Word is filled with zeros.

flags logical by value

A word whose bits, if on, specify the loading options for the slave program:

NOTE

Bit groups are denoted using the standard SPL notation. Thus bit(15:1) indicates bit 15, bits(10:3) indicates bits 10, 11, and 12.

Bit(15:1) - (Always set on.)

Bit(14:1) - LOADMAP bit. If on, a listing of the allocated (loaded) program is produced on the job/session list device. This map shows the Code Segment Table (CST) entries used by the new process. If off, no map is produced.

Default: Off.

Bit(13:1) - DEBUG bit. Bit must be off (0) -- no breakpoint can be set.

Default: Off.

Bit(12:1) - If on, the slave program is loaded in non-privileged mode. If this bit is off, the program is loaded in the mode specified when the program file was prepared.

Default: Off.

Bits(10:2) - LIBSEARCH bits. These bits denote the order in which remote libraries are to be searched for the slave program:

00 System Library

01 - Account Public Library, followed by System Library.

10 - Group Library, followed by Account Public Library and System Library.

Default: 00

Bit(9:1) - NOCB bit. If on, file system control blocks are established in an extra segment. If off, control blocks may be established in the Process Control Block Extension (PCBX) area.

Default: Off.

NOTE

This bit should be set on if the slave program uses a large stack.

Bits(7:2) - Reserved for MPE. Should be set to zero.

Bits(5:2) - STACKDUMP bits. Bits must be off (00).

Default: 00

Bit(4:1) - Reserved for MPE. Should be set to zero.

NOTE

The following bits (0:4) are ignored, because the bit pair (5:2) must be 00.

Bit(3:1) - DL to QI bit. If on, the portion of the stack from DL to QI is dumped. If off, this portion of the stack is not dumped.

Default: Off.

Bit(2:1) - QI to S bit. If on, the portion of the stack from QI to S is dumped. If off, this portion of the stack is not dumped.

Default: Off.

Bit(1:1) - Q-63 to S bit. If on, the portion of the stack from Q-63 to S is dumped. If off, this portion of the stack is not dumped.

Default: Off.

stacksize integer by value

An integer (Z - Q) denoting the number of words assigned to the local stack area bounded by the initial Q and Z registers.

Default: The same as that specified in the program file.

dlsize integer by value

An integer (DB - DL) denoting the number of words in the user-managed stack area bounded by the DL and DB registers.

Default: The same as that specified in the program file.

maxdata integer by value

The maximum size allowed for the process stack (Z - DL) area in words. When specified, this value overrides the one established at program-preparation time.

Default: If not specified, and not specified in program file either, MPE assumes that the stack will remain the same size.

bufsize integer by value

The size in words of the communications buffer (DS/3000 buffer) that is to be established by the remote DS/3000 software. It has a maximum value of 4096 words. Note that this parameter defines the maximum number of words of data that can be transmitted by a PWRITE or PREAD intrinsic call.

Default: Same size as the line buffer defined by the :DSLINE command (LINEBUF=) for the first :DSLINE issued to the *dsdevice*. Will never be smaller than 304 words.

If no LINEBUF= is specified by the first :DSLINE command, then the default configuration length is used. If X.25 is being used, the default configuration length will be 138.

CONDITION CODES

CCE	Request accepted by remote slave program.
CCG	Request rejected by remote slave program.
CCL	Request denied; an error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The POPEN intrinsic creates and activates a process in the remote HP 3000 for the specified remote slave program (*progname*) and optionally transmits a tag field (*itag*) to that remote slave program. The remote slave program must issue a GET intrinsic call followed by either an ACCEPT or REJECT call to complete the POPEN operation. The remote slave program may transmit a tag field back to the master program as part of an ACCEPT or REJECT call. If the master program transmitted a tag field, then the returned tag field (if any) is available in *itag*. If the master program did not transmit a tag field, then the returned tag field (if any) is not accessible.

The *bufsize* parameter specifies the length in words of an area to be established by the remote DS/3000 software as a communications buffer. This buffer is established implicitly as part of the GET call that receives the POPEN request. The value will be the maximum size of a PREAD or PWRITE data buffer.

NOTE

The master program is limited to one slave program on each line. Thus, only one POPEN (to a given node) is permitted, assuming only one line connects the two systems. After a POPEN intrinsic call, the remote slave program remains activated, and both the communications link and the DS/3000 buffer remain intact, even if the POPEN request is rejected by the remote slave program. The meaning of a POPEN reject by the remote slave program must be established as part of the design of the user's application.

The POPEN activity is illustrated in Figure 6-2.

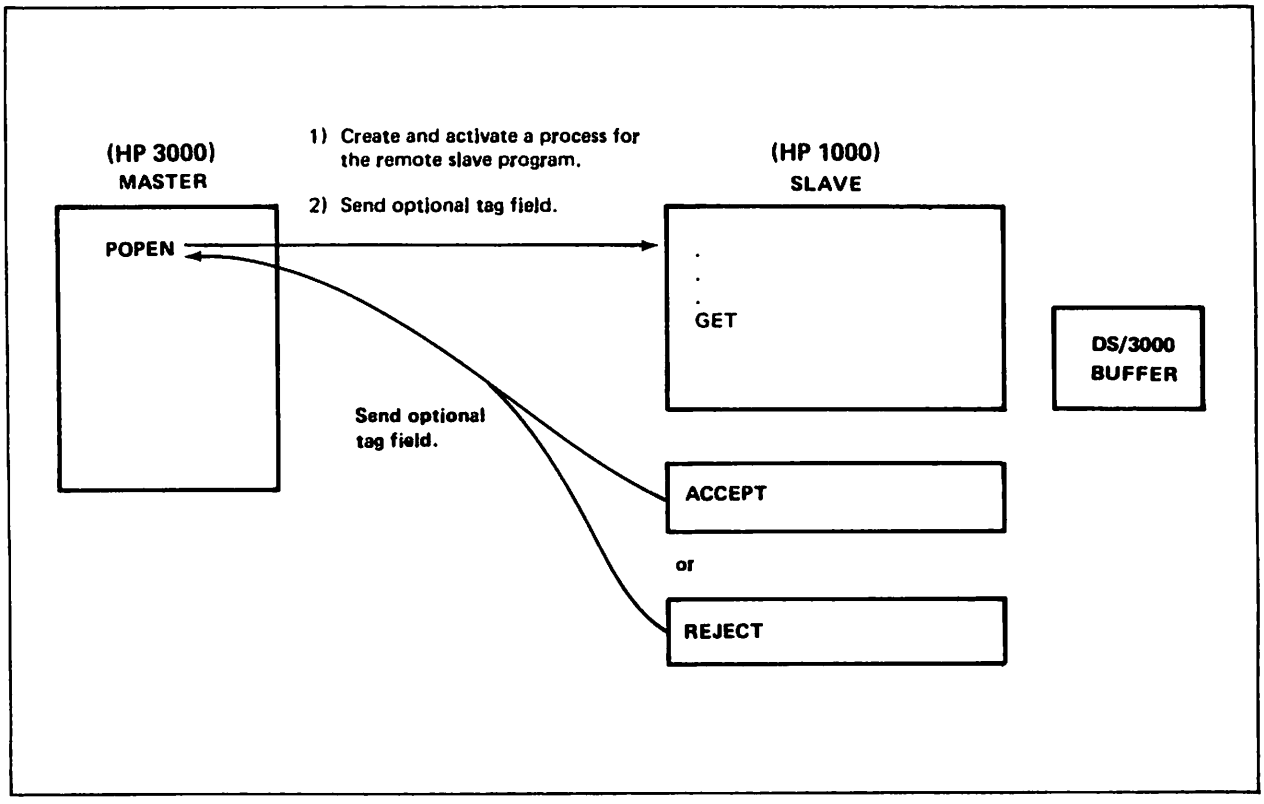


Figure 6-2. POPEN Activity.

PREAD

(Master callable) Asks the remote slave program to return a block of data.

SYNTAX

```
lgth:=PREAD(dsnum,target,tcount[,itag]);
```

FUNCTIONAL RETURN

The PREAD intrinsic returns a positive integer value showing the length (*lgth*) of the information transferred. If the *tcount* parameter in the PREAD call was positive, the positive value returned represents a word count; if the *tcount* parameter was negative, the positive value returned represents a byte count.

PARAMETERS

dsnum integer by value

The link identifier returned by the particular POPEN intrinsic call which initiated communication with the remote slave program.

target integer array

The array into which data received from the remote slave program will be deposited.

tcount integer by value

The requested number of words (if positive) or bytes (if negative) of data

itag integer array

A twenty-word array used for transmitting and receiving a tag field. The format of the tag field is defined by the master and slave programs and may serve any purpose the user desires.

CONDITION CODES

CCE Request accepted by remote slave program.

CCG Request rejected by remote slave program.

CCL Request denied; an error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The PREAD intrinsic transmits a read request to the remote slave program and optionally transmits a tag field from *itag* to the remote slave program. The remote slave program must issue a GET intrinsic call followed by either an ACCEPT or REJECT call to complete the PREAD operation. The GET call moves the tag field from the master program into the *itag* field provided in the remote slave program. The ACCEPT call moves the requested block of data from the remote program's data buffer into the target in the master program, and it also sends the optional *itag* back to the master program. The REJECT call transmits no data; it only returns the optional tag field. If the master program did not transmit a tag field, the returned field (if any) is not accessible.

The PREAD activity is illustrated in Figure 6-3.

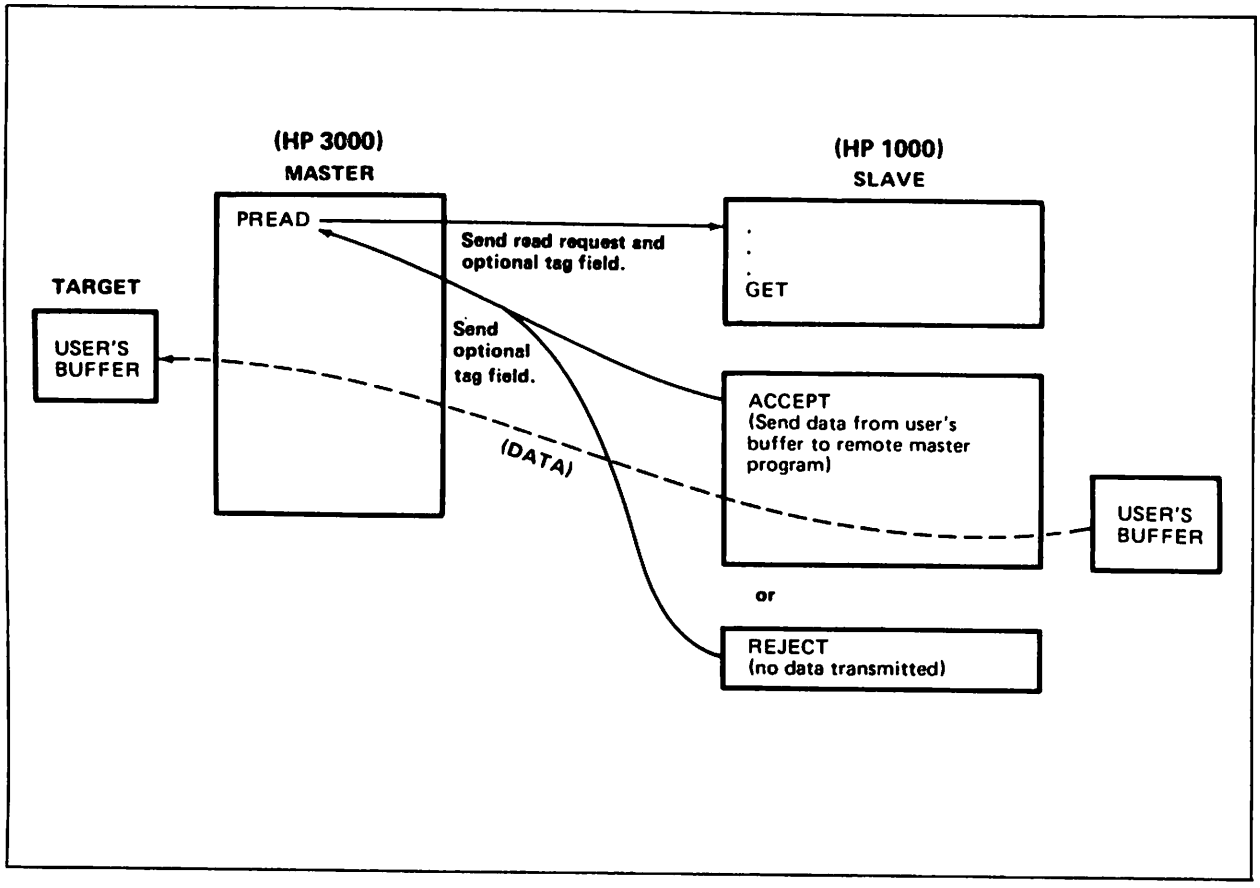


Figure 6-3. PREAD Activity.

PWRITE

(Master callable) Sends a block of data to the remote slave program.

SYNTAX

```
PWRITE(dsnum,target,tcount[,itag]);
```

PARAMETERS

dsnum integer by value

The link identifier returned by the particular POPEN intrinsic call which initiated communication with the remote slave program.

target integer array

The array from which data will be transmitted to a remote slave program.

tcount integer by value

The requested number of words (if positive) or bytes (if negative) of data.

itag integer array

A twenty-word array used for transmitting and receiving a tag field. The format of the tag field is defined by the master and slave programs and may serve any purpose the user desires.

CONDITION CODES

CCE Request accepted by remote slave program.

CCG Request rejected by remote slave program.

CCL Request denied; an error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The PWRITE intrinsic transmits a block of data (number of words or bytes = *tcount*) from *target* to the DS/3000 buffer in the remote HP 3000, and optionally transmits a tag field from *itag* to the remote slave program. The remote slave program must issue a GET intrinsic call followed by either an ACCEPT or REJECT call to complete the PWRITE operation. The GET call moves the tag field from the master program into the *itag* field provided in the remote slave program. The ACCEPT call moves the data from the remote DS/3000 buffer into the remote slave program's buffer, and it also sends the optional *itag* back to the master program. The REJECT call refuses the write request (the data in the

DS/3000 buffer is no longer accessible to the slave program) and returns the optional tag field to the master program.

The PWRITE activity is illustrated in Figure 6-4.

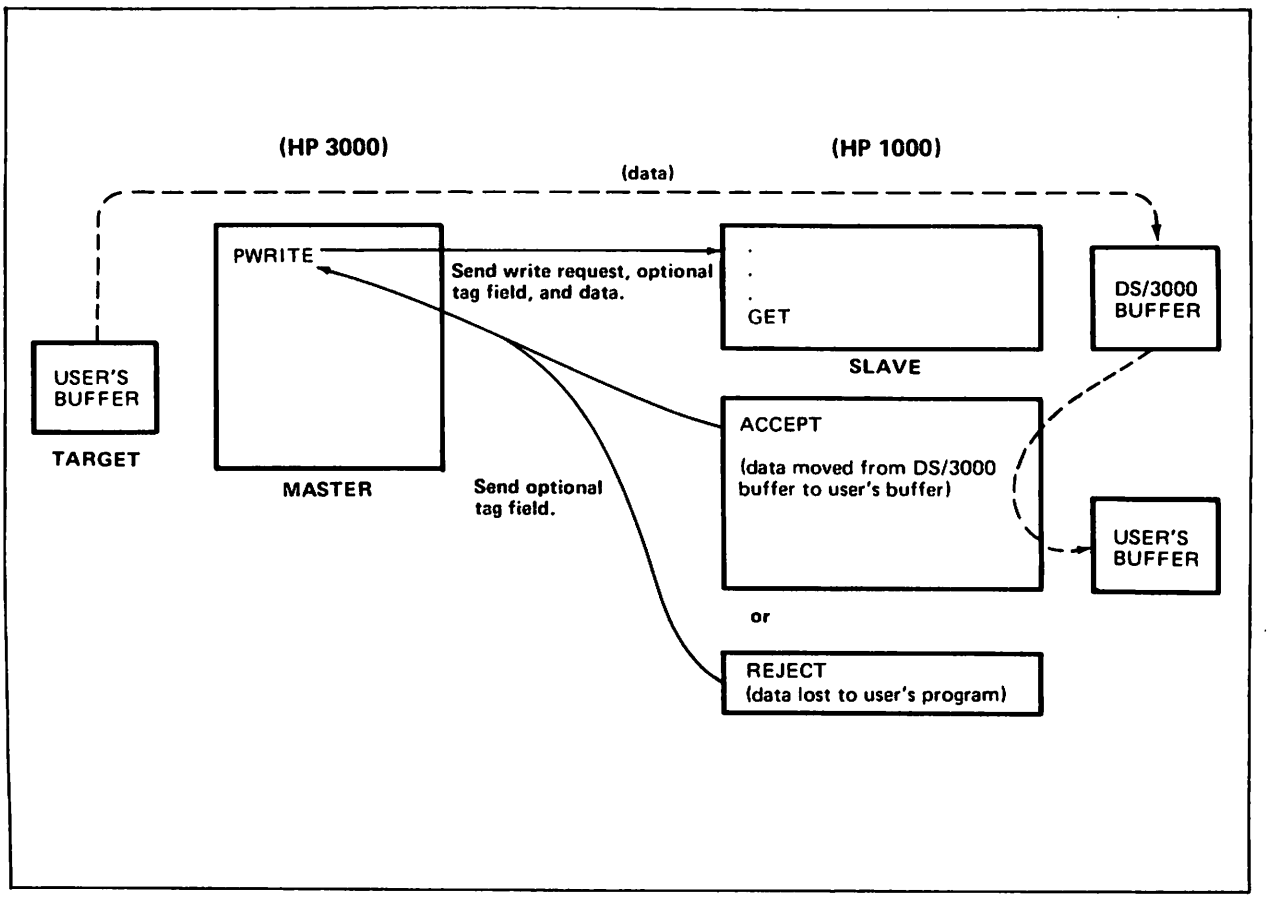


Figure 6-4. PWRITE Activity.

REJECT

(Slave callable)

Rejects the request received by the preceding GET intrinsic call and returns an optional tag field to the remote master program.

SYNTAX

```
REJECT([itag]);
```

PARAMETERS

itag integer array

A twenty-word array used for transmitting a tag field. The format of the tag field is defined by the user's master and slave programs.

CONDITION CODES

CCE	Response transmitted successfully to the remote master program.
CCG	(Not returned).
CCL	An error occurred. Issue a PCHECK intrinsic call to determine what happened.

OPERATION

The REJECT intrinsic rejects the request received by the most recent GET intrinsic call and transmits an optional tag field (*itag*) back to the remote master program.

INTERFACING WITH COBOL AND BASIC

Access to the program-to-program communications capability is available to ANS COBOL (COBOL/I) and BASIC users only through the interface routines described in Appendices B and C, respectively.

Programs written in SPL, FORTRAN, and COBOL II/3000 can use the PTOP intrinsics described in this section.

PTOP EXAMPLE

This example shows how two programs can communicate with one another by using the master and slave program-to-program intrinsics. The comments included within each program tell what is happening.

Master Program

```
 1 $CONTROL USLINIT,ADR,MAP,CODE
 2 BEGIN
 3
 4 COMMENT
 5    NAME OF PROGRAM IS MASTERP.
 6    THE SOURCE IS MASTERS.
 7    THIS PROGRAM IS TO BE RUN ON THE MASTER CPU.  IT WILL START
 8    THE "SLAVEP" PROGRAM ON THE SLAVE CPU. THE PROGRAM WILL THEN
 9    RECEIVE A KNOWN TEST PATTERN FROM THE USER TERMINAL, WRITE IT
10    TO THE REMOTE DISC FILE, READ IT BACK 5 TIMES, AND PRINT IT
11    ON THE LOCAL LP 5 TIMES.
12    THE TRANSFER OF DATA IS DONE THRU PTOP.;
13
14
15 INTEGER
16   ERROR,
17   LINE'NUM,
18   I,
19   J,
20   LPDEV'NUM;
21
22 BYTE ARRAY DS'DEVICE(0:6):=" ";
23 BYTE ARRAY LPDEV(0:2):="LP ";
24 BYTE ARRAY LPFILE(0:6):="LPFILE ";
25 BYTE ARRAY MSG(0:79);
26 BYTE ARRAY PROG'NAME(0:19):="SLAVEP.PUB.SUPPORT ";
27
28
29 LOGICAL ARRAY IOBUF(0:39);
30 LOGICAL ARRAY ITAG(0:19):=20(%020040);
31 LOGICAL ARRAY MSGW(*)=MSG;
32 LOGICAL ARRAY DS'DEVW(*)=DS'DEVICE;
33
```

```
34
35 INTRINSIC DEBUG,FCLOSE,FOPEN,FWRITE,PCONTROL;
36 INTRINSIC PCLOSE,POPEN,PREAD,PRINT,PWRITE,READ;
37
38 MOVE MSG:=" INPUT NAME OF DSDEVICE";
39 PRINT(MSGW,-28,0);
40 READ(DS'DEVW,-7);
41
42 MOVE MSG:=" POPEN ISSUED";
43 PRINT(MSGW,-18,0);
44
45 LINE'NUM:=POPEN(DS'DEVICE,PROG'NAME,ITAG);
46   IF <> THEN
47
48     BEGIN
49       PRINT(ITAG,20,0);
50       ERROR := 1;
51       GO TO ERR'PROC;
52     END
53     ELSE
54       PRINT(ITAG,20,0);
55
56
57 MOVE MSG:=" POPEN COMPLETED SUCCESSFULLY";
58 PRINT(MSGW,-33,0);
59
60 LPDEV'NUM:=FOPEN(LPFILE,4,1,40,LPDEV);
61   IF <> THEN BEGIN ERROR:=2;GO TO ERR'PROC; END;
62
63 MOVE MSG:="IN PUT TEST RECORD MAX. 80 CHAR";
64 PRINT(MSGW,-30,0);
65
66 MOVE IOBUF:=" "; <<CLEAR OUT BUFFER AREA>>
67 MOVE IOBUF(1):=IOBUF,(39);
68
69 READ(IOBUF,-80); <<GET RECORD TO WRITE>>
70
71 PWRITE(LINE'NUM,IOBUF,40); <<SEND RECORD TO REMOTE>>
72 IF <> THEN BEGIN ERROR:=3;GO TO ERR'PROC; END;
73
74 MOVE MSG:=" DISC FILES BEING XFERRED FROM REMOTE";
75 PRINT(MSGW,-41,0);
76 J:=-1; <<START READING FROM REMOTE>>
77 WHILE (J:=J+1)<5 DO
78   BEGIN
79     MOVE MSG:=" PREAD ISSUED";
80     PRINT(MSGW,-19,0);
81
82     MOVE IOBUF:=" ";
83     MOVE IOBUF(1):=IOBUF,(39);
84
85     I:=PREAD(LINE'NUM,IOBUF,40,ITAG);
86     IF = THEN
87       BEGIN
```

```
 88          IF J=4 THEN
 89            BEGIN
 90              MOVE MSG:=" ALL DISK RECORDS XFERRED";
 91              PRINT(MSGW,-29,0);
 92            END;
 93        END
 94          ELSE
 95        BEGIN ERROR:=4;GO TO ERR'PROC; END;
 96      FWRITE(LPDEV'NUM,IOBUF,I,0);
 97        IF <> THEN BEGIN ERROR:=4;GO TO ERR'PROC;END;
 98    END;
 99
100 FCLOSE(LPDEV'NUM,0,0);
101 PCLOSE(LINE'NUM);
102 IF <> THEN BEGIN ERROR:=5;GO TO ERR'PROC;END;
103 MOVE MSG:="END OF MASTER PROGRAM";
104 PRINT(MSGW,-21,0); GO TO END'IT;
105
106 ERR'PROC: <<HANDLE ERROR CONDITIONS>>
107   DEBUG;
108   FCLOSE(LPDEV'NUM,0,0);
109   PCLOSE(LINE'NUM);
110   MOVE MSG:="ERROR, END MASTER PROGRAM";
111   PRINT(MSGW,-25,0);
112
113 END'IT: END.
```

Slave Program

```
  1 $CONTROL USLINIT,ADR,MAP,CODE
  2
  3 BEGIN
  4 COMMENT
  5    THE NAME OF THIS PROGRAM IS SLAVEP.
  6    THE NAME OF THE SOURCE IS SLAVES.
  7    THIS PROGRAM IS TO BE COMPILED AND PREP'ED ON THE
  8    SLAVE HP3000 SYSTEM. IT WILL BE INITIATED FOR EXECUTION
  9    BY THE MASTER. THE FUNCTION OF THIS PROGRAM IS TO
 10    LOAD A DISC FILE WITH 5 KNOWN TEST PATTERNS THAT WILL
 11    BE TRANSFERRED TO THE MASTER 5 TIMES AND PRINTED ON THE
 12    MASTER'S LINE PRINTER 5 TIMES;
 13
 14
 15 INTEGER
 16   ERROR,
 17   DISK'FILENUM,
 18   I,
 19   IL,
 20   IONUMBER,
 21   J;
 22
 23 BYTE ARRAY MSG(0:79);
 24 BYTE ARRAY TEST(0:4):="TEST ";
```

```
25
26 LOGICAL ARRAY DISK'BUF(0:39);
27 LOGICAL ARRAY ITAG(0:19):=20(020040);
28 LOGICAL ARRAY MSGW(*)=MSG;
29
30 INTRINSIC FOPEN,DEBUG,FWRITEDIR,FREADDIR,FCLOSE;
31 INTRINSIC GET,ACCEPT,PRINT,READ,REJECT;
32
33
34 IL:=40;
35 MOVE MSG:="ISSUING A GET (REMOTE)";
36 PRINT(MSGW,-22,0);
37 I:=GET(ITAG); <<GET FOR POPEN>>
38 IF < THEN
39   BEGIN
40     MOVE ITAG:="ERROR ON GET;POPEN";
41     GO TO ERR'PROC;
42 END;
43
44 IF I=1 THEN
45   BEGIN
46     MOVE MSG:="POPEN RCVD...ISSUING AN ACCEPT (REMOTE)";
47     PRINT(MSGW,-39,0);
48     ACCEPT(ITAG); <<ACCEPT FOR POPEN>>
48 END;
50
51 MOVE ITAG:="POPEN ACCEPT SUCCESSFUL (REMOTE)";
52
53 DISK'FILENUM:=FOPEN(TEST,4,%104,-80,,,,1,1,10D);
54   IF <> THEN BEGIN ERROR:=1;GO TO ERR'PROC; END;
55
56 I:=GET; <<TEST REC FROM MASTER>>
57 IF <> THEN BEGIN ERROR:=2; GO TO ERR'PROC; END;
58 IF I=3 THEN <<PWRITE RECEIVED>>
59   BEGIN
60     ACCEPT( ,DISK'BUF);
61     IF <> THEN BEGIN ERROR:=3; GO TO ERR'PROC; END;
62 END;
63
64
65 I:=-1; <<START WRITING TEST FILE>>
66 WHILE(I:=I+1) < 5 DO
67   BEGIN <<WRITE REC TO DISK>>
68     FWRITEDIR(DISK'FILENUM,DISK'BUF,40,DOUBLE(I));
69       IF <> THEN BEGIN ERROR:=4; GO TO ERR'PROC; END;
70
71     END; <<END WRITING TEST FILE>>
72
73 J:=-1; <<SEND DISK FILE TO MASTER>>
74 WHILE(J:=J+1)<5 DO
75   BEGIN
76     MOVE MSG:="ISSUING A GET (REMOTE)";
77     PRINT(MSGW,-22,0);
78     I:=GET(ITAG,IL,IONUMBER);
```

```
 79     IF < THEN BEGIN ERROR:=5; GO TO ERR'PROC; END;
 80     IF I=2 THEN
 81       BEGIN
 82         MOVE MSG:="PREAD RCVD...ISSUING AN ACCEPT";
            (REMOTE)";
 83         PRINT(MSGW,-39,0);
 84       END
 85       ELSE
 86         BEGIN ERROR:=6;GO TO ERR'PROC; END;
 87     MOVE DISK'BUF:=%020040;
 88     MOVE DISK'BUF(1):=DISK'BUF(0),(39);
 89     FREADDIR(DISK'BUF,40,DOUBLE(J));
 90     IF <> THEN BEGIN ERROR:=7;GO TO ERR'PROC; END;
 91     ACCEPT(ITAG,DISK'BUF,40);
 92     IF <> THEN BEGIN ERROR:=8;GO TO ERR'PROC; END;
 93   END;
 94
 95 FCLOSE(DISK'FILENUM,0,0); GO TO END'IT;
 96
 97 ERR'PROC: <<HANDLE ERROR CONDITIONS>>
 98   DEBUG; <<WILL PROMPT AT MASTER SIDE TERMINAL>>
 99   REJECT;
100   I:=GET; <<ALLOW PCLOSE>>
101 GO TO ERR'PROC;
102
103 END'IT: END;
```

NETWORK FILE TRANSFER

SECTION 7

The Network File Transfer (NFT) program runs on an HP 3000 Computer System to provide the ability to copy disc files efficiently. When initiated over a DS/3000 communications link, the NFT program can copy a file to or from any other adjacent HP 3000 computer which also provides this service.

FEATURES OF NFT

- You can initiate copy operations from sessions, jobs, or programs.
- DSCOPY can be used to copy users' files and MPE system files, as well as data management files, such as KSAM/3000 files.
- There is only one NFT command to learn -- :DSCOPY.
- There are two intrinsics: DSCOPY and DSCOPYMSG. The intrinsics are callable from programs written in SPL/3000, COBOL, Pascal, FORTRAN, and BASIC.
- NFT can be used in Interactive Mode to submit a series of copy requests. When a DSCOPY command or intrinsic initiates Interactive Mode, users' requests are placed in a transaction file whose formal designator is DSCOPYI. The default for this file is $STDINX. This file must be unnumbered.
- NFT can record a history of all copy operations performed by DSCOPY requests. The history report can be printed to $STDLIST, as well as to a secondary file. The secondary file has the formal designator DSCOPYL.
- You can initiate a copy operation from a system other than the system(s) where the source and target files are located.
- NFT can efficiently copy disc files within your local HP 3000.
- The files referenced by a DSCOPY command (or intrinsic) may reside on system or private volumes.

File transfers can involve one or more computers. In all transfers, there are three distinct roles a system can play:

1. The **initiator** is always the system where the :DSCOPY command originates. The initiator functions only in an outgoing sense. It is similar to PTOP operation, where the PTOP master program always issues a POPEN out across a DS line to cause a slave to be created and activated on a remote system.
2. The **producer** is the source computer where the file that is to be copied resides.
3. The **consumer** is the target computer where the new file will reside.

You should remember that one system may be performing two or all three of these roles.

When a DSCOPY request names a remote source, the DS line to that computer must be open and a remote session must exist. The same is also true when a remote target is specified.

When DSCOPY is used to transfer files over two or more systems, the following restrictions apply:

1. DSCOPY must be initiated only from the master side of the DS line. The slave (remote) side cannot be the initiator of a DSCOPY command.

2. DSCOPY must not be initiated programmatically from either a master or a slave PTOP program in any direction.

SYNTAX

```
:DSCOPY [ [source] [ { TO } [target] ] ]
                     { ; }
```

where the command syntax has the following meanings:

```
source   --   sfile [ ,sdsdev [ { , } sdev ] ]
                                { # }

target   --   tfile [ tdsdev ] [ { , } [tdev] ] [ ;FCODE=sourcefilecode ]
                     ,*         { # }
```

To submit a series of transfer requests, omit all of the source and target parameters to initiate Interactive Mode. NFT prompts you for input and, after the transfer completes, prompts you again for the next transfer request.

Terminate Interactive Mode by typing // or (CONTROL)Y.

PARAMETERS

sfile

Identifies the file to be copied. The name can be written in the following format:

sfile[*lockword*] [.*groupname*] [.*accountname*]

If the source file is in a *group.account* different from the requestor's log-on *group.account*, the requestor must have read and lock access to the source file. (Lock access means that the file cannot be opened for writing while :DSCOPY is copying it.)

sdsdev

The device classname, logical device number, or node name that was used to open the communications link to the remote computer where the source file resides.

Default: The local system (that is, the system where the transfer request is submitted).

sdev

The classname or logical device number of the disc where the source file resides.

Default: DISC.

tfile

Specifies the file to receive the data. The name can be written in the following format:

tfile[*lockword*] [.*groupname*] [.*accountname*]

Default: The new file has the same filename as the source file. The default *groupname* and *accountname* are the log-on *groupname* and *accountname*. Security is on for the new file, even though the source file may have been released.

tdsdev

The device classname, logical device number, or nodename that was used to open the communications link to the remote computer where the target file will reside.

Default: :DSCOPY copies the sourcefile to the local computer and assigns the same filename as the sourcefile name. If the source computer is the local system, this default causes a file system error (because the file already exists).

*

Means the target *dsdevice* (the target computer) is the same as the source *dsdevice* (the source computer).

tdev

The device classname or logical device number of the disc where the new file should reside.

Default: DISC

FCODE= *sourcefilecode*

Applies to privilege mode files only. Specifies that the new file will have the same negative file code as the source file. System Manager or Privilege Mode capability is required for both the source and target file creators.

USE

Available	in Session?	YES
	in Job?	YES
	in Break?	NO
	Programmatically?	NO*
Breakable?		NO**

* Call the DSCOPY intrinsic rather than use the COMMAND intrinsic.

** Use (CONTROL)Y, rather than (BREAK).

OPERATION

NOTE

(BREAK) is disabled during DSCOPY.

If you enter (CONTROL)Y during a copy operation, DSCOPY prints the percentage of the transfer that is complete and prompts whether to cancel or continue the operation.

Source and Target Files

In a DSCOPY command, source and target files are referenced as defined by the systems upon which they reside.

There is no default for a sourcefile.

A default for a targetfile is derived from the sourcefile. The default consists of the first sequence of characters in the sourcefile name which constitutes a legal HP 3000 file name. For example:

```
:DSCOPY SFILE.SGROUP,SNODE
```

Here the source file is SFILE (in group SGROUP on a remote system). The targetfile is generated in the users' log-on group (on the local system) and is assigned the default name SFILE. The characteristics of the new file are the same as those of the source file.

In order to transfer a file with a negative file code (i.e., a privilege mode file), the FCODE= parameter must be included in the command string, and the log-on user on both the source and target systems must have System Manager (SM) or Privilege Mode (PM) capability. (Remember that the user requesting the transfer (the initiator) is not necessarily the log-on user at the target (the consumer).) After a successful copy operation, the new file has the same negative file code as the source file (as specified by *sourcefilecode*). For example:

```
:DSCOPY SFILE,SNODE TO TFILE;FCODE=-401
```

When copying KSAM files, both the data file and its key file are copied. The DSCOPY user can specifically name a data file/key file pair by enclosing the file names in quotes and separating them by a comma. For example:

```
:DSCOPY SFILE TO "DATAFILE,KEYFILE"
```

When a user specifies a source KSAM data file and the NFT subsystem must generate a default key file, it uses the data file name and appends a K. For example:

```
:DSCOPY SFILE, LINE1 TO TFILE, LINE2
```

In the case where SFILE is a KSAM data file, the new data file on the computer connected to LINE2 will be named TFILE and the associated key file will be named TFILEK by default.

NOTE

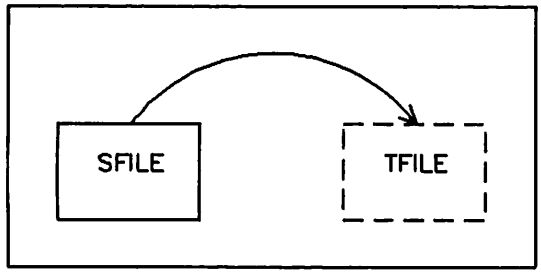
DSCOPY cannot copy to a back-referenced target file (e.g., the form DSCOPY SFILE TO *TFILE is not permitted).

EXAMPLES

Local Copy

To make a local copy of SFILE and name the new file TFILE, use either of the following:

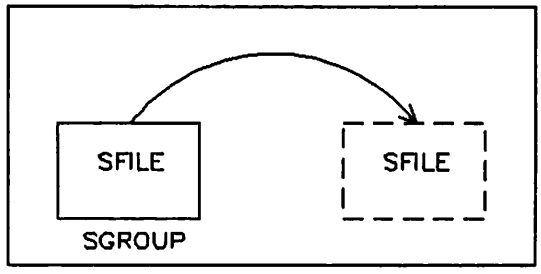
```
:DSCOPY SFILE TO TFILE or :DSCOPY SFILE; TFILE
```



LOCAL

The following example copies a file named SFILE from another group on the local system (SGROUP) into a file in the log-on group. The new file is also named SFILE.

```
:DSCOPY SFILE.SGROUP
```

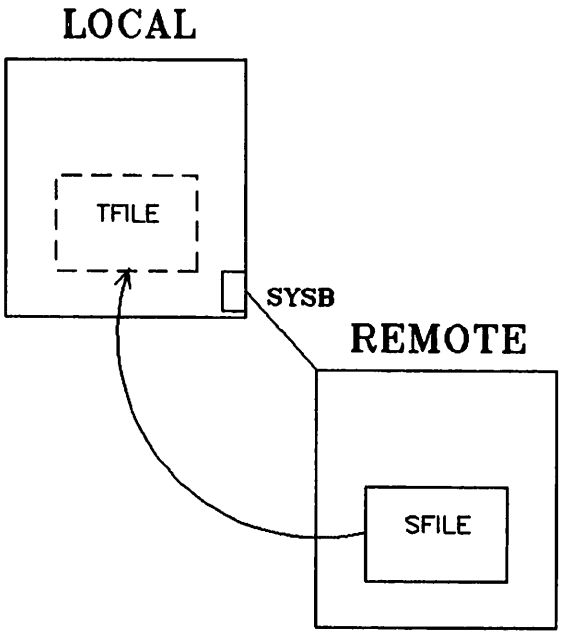


LOCAL

Remote-to-Local Copy

To copy a file from the computer connected to DS line SYSA into your log-on group (on the local system), enter:

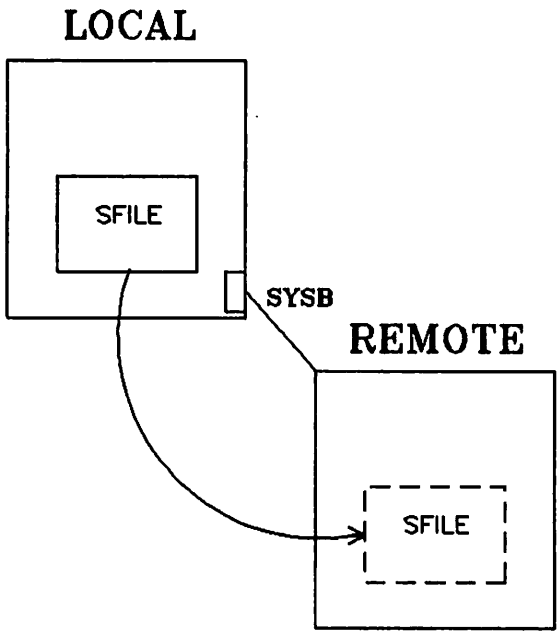
```
:DSCOPY SFILE,SYSA;TFILE
```



Local-to-Remote Copy

To copy a file named SFILE (on the local system) to the computer attached to DS line SYSB and name the new file SFILE, enter:

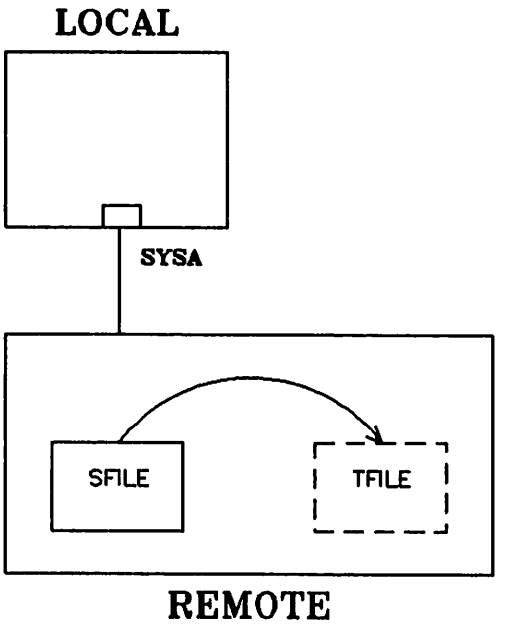
```
:DSCOPY SFILE TO ,SYSB
```



Remote Copy

An asterisk (*) means the target system is also the source system. The following example copies a file named SFILE to a new file named TFILE. Both files reside on the remote computer connected to the dsline named SYSA.

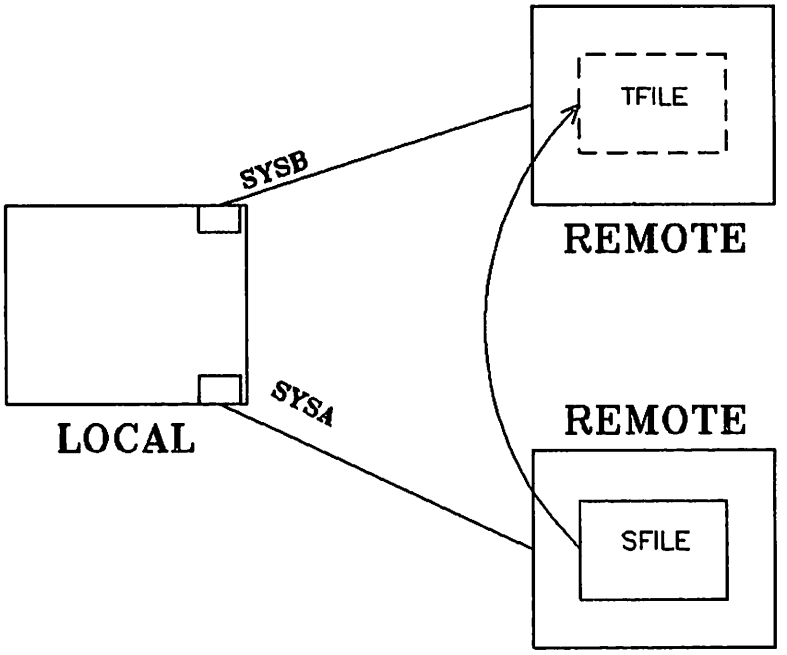
```
:DSCOPY SFILE,SYSA TO TFILE,*
```



Remote-to-Remote Copy

The next example illustrates a command that copies a file from one remote system to another. In this case, the communications lines to both remote computers must be open and a remote session must exist on each system.

```
:DSCOPY SFILE,SYSA TO TFILE,SYSB
```



INTERACTIVE MODE

To execute a series of transactions interactively, enter the :DSCOPY command without parameters. Now the system prompts you for input with the word DSCOPY and accepts your response from the file DSCOPYI (whose default is $STDINX).

The syntax required for your response follows the format already described for source and target parameters.

You can issue MPE comands while in Interactive Mode by entering a colon (:) before the command. The MPE commands allowed in Interactive Mode are those allowed by the COMMAND intrinsic.

Note the following about Interactive Mode:

- To continue your response on the next line, enter an ampersand (&) as the last non-blank character on the current line and press [RETURN]. A continuation prompt is printed so that you can continue your response.
- To cancel a response while entering a line, use [CONTROL]X.
- To terminate Intractive Mode, enter // or [CONTROL]Y.

Multiple Transactions

DSCOPY also allows you to perform multiple file transfers by redirecting DSCOPYI to a disc file containing the individual file transactions. To use this mode of DSCOPY, simply create an unnumbered ascii file, with one transfer per line, using the following format:

```
SFILE[,SDSDEV] TO TFILE[,TDSDEV]
```

where SFILE is the source file, TFILE is the target file, and SDSDEV or TDSDEV is the device class name, LDEV number, or node name that was used to open the communications link to the remote computer. Keep this file, unnumbered, using a filename of your choice. Next, enter an MPE file equation that references your transaction file to DSCOPYI, and then enter the :DSCOPY command without parameters. In the following example, user input is underscored, and the existence of a remote session is assumed.

NOTE

Imbedded MPE commands are not supported if DSCOPYI has been redirected to a file other than the default, $STDINX.

Example

```
EDITOR
HP 32201A.7.17 EDIT/3000 TUE, OCT 29, 1985,  8:35 AM
(C) HEWLETT-PACKARD CO. 1985
/ADD
   1  SFILEA,REMOTE1 TO TFILEA
   2  SFILEB,REMOTE1 TO TFILEB
   3  SFILEC,REMOTE1 TO TFILEC
   4  //
/K XFERFILE,UNN
/E

END OF PROGRAM
:FILE DSCOPYI=XFERFILE
:DSCOPY

Network File Transfer [HP32185B.52.00]  (C) Hewlett-Packard Co. 1980

DSCOPY SFILEA,REMOTE1 TO TFILEA
  New file created -- TFILEA.TGROUP.TACCT --
  Succeeded.

DSCOPY SFILEB,REMOTE1 TO TFILEB
  New file created -- TFILEB.TGROUP.TACCT --
  Succeeded.

DSCOPY SFILEC,REMOTE1 TO TFILEC
  New file created -- TFILEC.TGROUP.TACCT --
  Succeeded.

END OF SUBSYSTEM
:
```

Event Recording

DSCOPY produces printed output to document user input and copy results. This output may be sent to a primary file and/or a secondary file, either of which may be disabled. The primary file is $STDLIST and the secondary file has the formal designator DSCOPYL. All user requests and DSCOPY prompts are printed on $STDLIST and echoed on the secondary file (and on the primary, if not duplicative). Primary output is enabled by a :DSCOPY command, or by the DSCOPY intrinsic with the OPT parameter set to 4, 5, or 6 (refer to the parameters of the DSCOPY intrinsic). Output for the secondary file, DSCOPYL, defaults to $NULL so that secondary output is disabled by default. It can be enabled by using a :FILE command to equate DSCOPYL to a file or a line printer, or to $STDLIST.

PROGRAMMATIC MODE

Programs can use the DSCOPY intrinsic to copy disc files.

Programs can also print a message which corresponds to the result code returned by a DSCOPY intrinsic call. The DSCOPYMSG intrinsic is used for this purpose.

(BREAK) is disabled during DSCOPY. After you finish with the DSCOPY intrinsic, you must call FCONTROL 15 to set (BREAK) back on. See the *MPE Intrinsics Manual* for details.

The rules for using the intrinsics are consistent with those for using other MPE intrinsics. Specifically, the following rules apply.

- Both intrinsics can be called from programs written in the SPL/3000, COBOL, Pascal, FORTRAN, and BASIC languages.
- Calling sequences for all of the languages are basically the same.
- All parameters are passed by reference.
- The intrinsics are not option variable.
- Neither of the intrinsics are typed (returns a parameter as its value).
- Neither returns a condition code (they both return a result).
- Split stack calls are not allowed.
- For COBOL, data types should be defined as follows:

Data Type	Data Description
Numeric	PICTURE S9(4) COMPUTATIONAL
Alphanumeric	PICTURE X(*n*) or PICTURE A(*n*)
Numeric Array	PICTURE S9(4) COMPUTATIONAL SYNCHRONIZED OCCURS *n* TIMES

- For Pascal, the following special data type should be defined:

Data Type	Data Description
SHORTINT	-32768..32767

DSCOPY Intrinsic

Allows programmatic use of DSCOPY

SYNTAX

```
DSCOPY (opt, spec, result);
```

PARAMETERS

opt logical

opt controls the primary output (i.e. output to $STDLIST) and specifies the type of copy operation.

Bits 0 through 12 are reserved for future use and should be set to zero. The remaining bits can be set to indicate the following:

Value	Meaning
0	Single transaction; primary output disabled.
1	Multiple transactions; return after first unsuccessful transaction; primary output disabled.
2	Multiple transactions; return after all transactions have been attempted or after an internal error occurs; primary output disabled.
4	Single transaction; primary output enabled.
5	Multiple transactions; return after first unsuccessful transaction; primary output enabled.
6	Multiple transactions; return after all transactions have been attempted or after an internal error occurs; primary output enabled.

spec logical array

The logical array should contain ASCII text terminated by an 8-bit binary zero. In the single transaction case, the syntax required is the same as for the DSCOPY command parameters.

In the multiple transaction case, the array should contain only a zero. Zero causes NFT to read the copy request from the DSCOPYI file (whose default

is $STDIN). This file must be unnumbered.

result logical array

A two-word array returned to the caller that indicates the outcome of the intrinsic call.

result(0) Indicates the copy operation was successful. Any other value represents an error as defined in "DSCOPY Error Messages" listed in Appendix A.

result(1) Shows the number of files that were successfully copied.

OPERATION

Simultaneous DSCOPY requests cannot be issued from two processes in the same session.

The only valid values for the *opt* parameter are: 0, 1, 2, 4, 5, or 6.

The ASCII text passed by the *spec* parameter must be terminated by a binary zero.

The values passed in the parameters are verified as being in bounds and valid.

The system creates the NFT process and passes the contents of *opt* and *spec* to it.

The specified files are copied by the NFT process.

The intrinsic returns the result to the user.

EXAMPLES

COBOL Calling Sequence

```
CALL "DSCOPY" USING OPT, SPEC, RESULT.
```

OPT Numeric data item.

SPEC Alphanumeric data item.

RESULT Numeric array of two or more data items.

FORTRAN Calling Sequence

```
CALL DSCOPY (OPT, SPEC, RESULT)
```

OPT	INTEGER*2 variable
SPEC	CHARACTER array
RESULT	An array of two or more INTEGER*2 variables

BASIC Calling Sequence

```
CALL BDSCOPY (O, S$, R)
```

O	Numeric variable
S$	A string variable
R	An array of two or more numeric variables

Pascal Calling Sequence

```
DSCOPY (OPT, SPEC, RESULT);
```

OPT	SHORTINT variable
SPEC	PACKED ARRAY OF CHAR array
RESULT	INTEGER variable

SPL Calling Sequence

```
DSCOPY (OPT, SPEC, IRESULT);
```

OPT	Logical
SPEC(0:30)	Logical Array
IRESULT(0:4)	Logical Array

DSCOPYMSG Intrinsic

Prints the result code returned by DSCOPY.

SYNTAX

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

PARAMETERS

result — logical array

The two-word result returned by the DSCOPY intrinsic.
0 = DSCOPY was successful.
n = An error occurred. Refer to the Error Messages in Appendix A.

fnum — integer by value

When *fnum*=0, the message associated with *result* is printed on $STDLIST.

When *fnum* contains a file number returned by an FOPEN call, the message associated with *result* is written to the file.

r — integer

Result returned by this DSCOPYMSG call.
0 = Successful call
n = Unsuccessful call. Refer to the Error Messages in Appendix A.

EXAMPLES

COBOL Calling Sequence

```
CALL "DSCOPYMSG" USING RESULT, FNUM, R.
```

RESULT — An array of two or more data items.

FNUM — A numeric data item.

R — A numeric data item.

FORTRAN Calling Sequence

```
CALL DSCOPYMSG (RESULT, FNUM, R)
```

RESULT — An array of two or more INTEGER*2 variables.

FNUM INTEGER*2 variable

R INTEGER*2 variable

BASIC Calling Sequence

```
CALL BDSCOPYMSG (R, F, RO)
```

R An array of two or more numeric variables

F An integer variable

RO An integer variable

Pascal Calling Sequence

```
DSCOPYMSG (RESULT, FNUM, R);
```

RESULT INTEGER variable

FNUM SHORTINT variable

R SHORTINT variable

SPL Calling Sequence

```
DSCOPYMSG (IRESULT, FNUM, R);
```

IRESULT(0:4) Logical Array

FNUM Integer

R Integer

PROGRAMMATIC EXAMPLES

A very simple example of a programmatic DSCOPY request is shown coded in the COBOL, FORTRAN, BASIC, Pascal and SPL/3000 languages.

The example copies a file (NFTTEST) to a new file (TEMP1). The source file resides on the local machine, and the new file will be created on a remote machine connected to line "HDS."

DSCOPY COBOL Example

```
1 $CONTROL CODE
1.1 $TITLE " DSCOPY INTRINSIC TEST"
1.2 IDENTIFICATION DIVISION.
1.3 PROGRAM-ID.  DSCOPY00.
1.4 AUTHOR.  SUZANNE FLAHERTY.
1.5 DATE-WRITTEN.  APRIL 1980.
1.6 DATE-COMPILED.
1.7 REMARKS.
1.8 THIS PROGRAM DOES A SIMPLE DSCOPY INTRINSIC CALL.
1.9 ENVIRONMENT DIVISION.
2 CONFIGURATION SECTION.
2.1 SOURCE-COMPUTER. HP3000
2.2 OBJECT-COMPUTER. HP3000
2.3 DATA DIVISION.
2.4 WORKING-STORAGE SECTION.
2.5 01 OPT PIC S9(4) COMP VALUE 0.
2.6 01 STRING1.
2.7 02 ASCIIPART PIC X(24) VALUE "NFTTEST TO TEMP1,HDS".
2.8 02 TERMINATOR PIC S9(4) COMP VALUE 0.
2.9 01 RESULT1.
3 02 RESULT2 PIC S9(4) COMP OCCURS 2 TIMES.
3.1 PROCEDURE DIVISION.
3.2 BEGINLABEL.
3.3 CALL INTRINSIC "DSCOPY" USING OPT, STRING1, RESULT1.
3.4 STOP RUN.
```

DSCOPY FORTRAN Example

```
25 $CONTROL MAP,LIST,CODE,CROSSREF,LOCATION,STAT
26        PROGRAM DSCOPY
27        CHARACTER*40 STRING1
28        INTEGER*2 FNUM, R
29        INTEGER*2 OPT
30        INTEGER ARRAY IRESULT(4)
31        SYSTEM INTRINSIC DSCOPY, DSCOPYMSG
32 C
33        DATA STRING1/20HNFTTEST TO TEMP1,HDS/
34 C
35 C THIS PROGRAM DOES A DSCOPY INTRINSIC REQUEST
36 C
37        OPT=0
```

```
38        FNUM=0
39        CALL DSCOPY(OPT, STRING1, IRESULT)
40        IF (IRESULT .GT. 0) CALL DSCOPYMSG(IRESULT, FNUM, R)
41        STOP
42        END
```

DSCOPY BASIC Example

```
 10 REM THIS WILL DO A SIMPLE DSCOPY REQUEST
 20 DIM A$[30],R[4]
 30 0=R2=Z=0
 40 MAT R=ZER
 50 A$=" NFTTEST TO TEMP1,HDS "
 60 PRINT A$
 70 CALL BDSCOPY(0,A$,R[*])
 80 IF R[1] <>0 THEN PRINT " ERROR IN DSCOPY. ERROR= ",R[1]
 90 IF R[1] <>0 THEN CALL BDSCOPYMSG(R[*],Z,R2)
100 STOP
110 END
```

DSCOPY Pascal Example

```
PROGRAM DSCOPY;
TYPE
   SHORTINT = -32768..32767;
VAR
   OPT,FNUM,R :  SHORTINT;
   RESULT     :  INTEGER;
   SPEC       :  PACKED ARRAY [1..127] OF CHAR;
PROCEDURE DSCOPY; INTRINSIC;
PROCEDURE DSCOPYMSG; INTRINSIC;
BEGIN
   SPEC:='NFTTEST TO TEMP1,HDS';
   OPT:=0;
   FNUM:=0;
   DSCOPY(OPT,SPEC,RESULT);
   IF (RESULT > 0) THEN DSCOPYMSG(RESULT, FNUM,R);
END.
```

DSCOPY SPL/3000 Example

```
1  $CONTROL USLINIT,CODE
2  BEGIN
4     INTEGER FNUM:=0,R:=0;
3     LOGICAL OPT:=0;
5     LOGICAL ARRAY SPEC(0:30), IRESULT(0:4);
6     INTRINSIC DSCOPY, DSCOPYMSG;
7     MOVE SPEC:="NFTTEST TO TEMP1, HDS";
8     DSCOPY(OPT,SPEC,IRESULT);
9     IF IRESULT > 0 THEN DSCOPYMSG(IRESULT,FNUM,R);
10 END.
```

DS APPLICATION DESIGN

SECTION 8

DS/3000 is particularly useful in applications that involve transaction processing and that are geographically or functionally dispersed. Any local-system command can be executed remotely through a simple extension to that command. Many operating system intrinsics are also extended in a similar fashion. No knowledge of the communication protocol or physical link being used is required of the terminal user or application programmer. Every application-level capability operates transparently across each connection-level alternative.

DS/3000 on the HP 3000 provides facilities for point-to-point connection between processors. These connections can be made on a variety of types of communications lines, including switched (dial-up), leased, or hardwired, and they can also be mixed throughout the network. Applications can easily obtain access to systems more than one "hop" away, through multiple :REMOTE HELLO log-ons. In addition, HP 3000 computers can connect to X.25 packet-switched networks and communicate across those networks with HP 1000 or other HP 3000 computers using the DSN/X.25 software. In fact, DS/3000 can maintain concurrent connections to multiple remote systems, and/or multiple connections to the same remote system, over a single physical link to the X.25 network.

DS/3000 requires users to pass all of the security checks imposed by MPE (such as passwords) when logging on to a remote system. DS/3000 also provides additional security features applicable only to a network environment. For example, the operator can restrict incoming or outgoing access to the communications link, and incoming host-to-host calls from an X.25 network are accepted only if the remote host is configured in the local system's network data base.

DS/3000 offers the tools to facilitate the sharing of resources within a network. Examples of such resources are programs, data structures, or physical hardware elements of the network. You can access these resources in any of several modes:

- Remote command execution allows you to direct commands to any CPU in the network.
- Remote File Access (RFA) permits the application of processing power to files and devices remote from the CPU. RFA also provides the means for extending Interprocess Communications (IPC) across a DS link.
- Program-to-program (PTOP) communication permits direct communications between master and slave programs, each resident in its own CPU within the network.
- Remote Data Base Access (RDBA) gives the capability for direct and indirect access of data bases on any HP 3000 computer in the network. Combining the distributed processing capability of DS/3000 with the use of data management subsystems such as VPLUS/3000, KSAM/3000, and TurboIMAGE makes possible the sharing of data
- Network File Transfer (NFT) is a more efficient mechanism than FCOPY for transferring disc files across a communications link.

The chief advantages of Program-to-Program (PTOP) communication are coprocessing capabilities and control of data transmission blocking. Coprocessing master-slave programs execute in multiple systems. Program-to-Program communication allows decision making to be distributed within the master-slave relationship. The exchange of data and control information between the executing programs can be used

to alter program flow to adjust to current conditions in the network. Coprocessing capabilities assume importance in networks where synchronization of modifications to related data structures is important.

There exist limitations on certain PTOP application designs. If your PTOP application intends to have one master program and multiple slave programs, the SYSTEM BREAK key should be disabled by calling the following procedure at the beginning of the master program:

```
PROCEDURE DISABLE BREAK;
BEGIN
   INTEGER STDIN;(file number)
   LOGICAL ANYINFO;(required by fcontrol)
   STDIN:=FOPEN( ,%244);
   FCONTROL(STDIN,14,ANYINFO);(disable system break)
END
```

If you opt to keep the SYSTEM BREAK key feature, the master program should be structured so that it will be made up of one father process and multiple son processes. Each of the son processes will be a master PTOP process and communicate with a slave PTOP process on a separate remote system. See Figure 8-1.

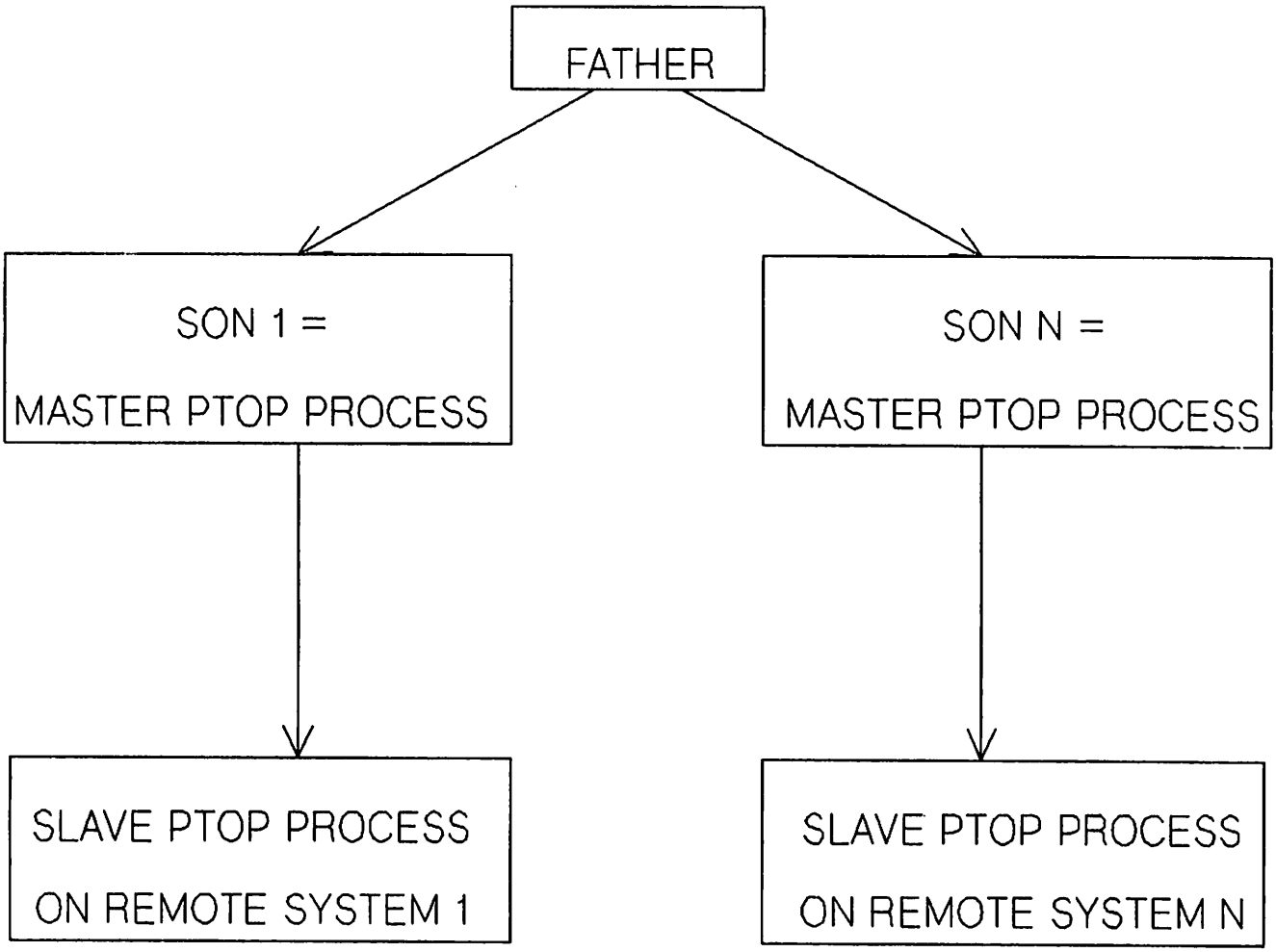


Figure 8-1. Master Program Structure.

Blocking control can be utilized in such a manner as to decrease the number of transmissions to move a specific amount of data. Since transmission time on a high-speed link is a negligible factor in communications performance, the required number of transmissions is the key to performance. Reducing the number of transmissions correspondingly reduces the number of line turnarounds. This may become a significant performance factor in half-duplex networks or satellite communication links where propagation delay affects response time.

TRANSMISSIONS BETWEEN SYSTEMS

Underlying all modes of utilizing DS/3000 is the transmission of data from one system to another. Now compare the building of these transmission units for remote file access and for PTOP.

DS/3000 is supported on three controllers: the Intelligent Network Processor (INP), the Hardwired Serial Interface (HSI), and the Synchronous Single-Line Controller (SSLC). To configure any of these devices into the system, you must specify a buffer length. The buffer length value that you specify represents the maximum number of words to be transmitted between systems in one transmission and it is the system's default buffer size. When you activate DS/3000 with a :DSLINE command, a *linebuf* parameter may be specified to override the configured buffer size. Only the first user to activate the line may use *linebuf* to alter the data communication buffer size. This buffer size may not be respecified until all concurrent DS/3000 users have closed their links. In this way, the pertinent buffer limiting factors for inter-CPU transmissions are set.

How is *linebuf* utilized in accessing remote files and remote peripheral devices? The basic unit for file system operations is the record (or block of records). In remote file access and remote command execution, file system blocking limits the transmission unit to a single record or sequential multiple records. File system's FREAD is satisfied by moving a logical record from a file to the user's buffer. An FREAD on a file open with multi-record access is satisfied by a byte count that can be specified to be block-sized. Thus, an FREAD on a remote file will pack *linebuf* with a record or a specified byte count of sequential records.

Contrast this record-orientation to the array-orientation of PTOP communications. PTOP's PREAD is satisfied by the transmission between programs of the contents of a user-defined buffer. The PTOP programmer must construct the buffer by packing it with array(s), record(s), or fields of records. The records in one transmission need not even come from the same file.

In addition to transmitting specified data, DS/3000 attaches a header of varying lengths. The header always contains eight words transmitted in a fixed format and can contain additional words in an appendage area. For remote command execution and remote file and peripheral device access, the data field is usually preceded by a header of 14 words. Some intrinsics, such as FREAD (multirecord), require a longer header to convey all parameter information. The header for PTOP communications includes the 20-word tag field in the appendage; thus the typical PTOP header is 34 words long. The ideal *linebuf* size will allow the user's data field plus DS/3000 header information to fit into *linebuf*.

To illustrate: Assume that you want to read six 80-byte records from a remote file and have specifed a *linebuf* of 304 words.

a. If the remote file is defined as REC=-80,1,F, then Remote File Access must retrieve a block of one record from the disc, FREAD one record, and transmit one record. The complete data transfer requires six disc accesses, six FREADs, and six data transmissions

b. If the remote file is defined as REC=-80,6,F, then Remote File Access must access the disc to retrieve a block of six records, satisfy an FREAD with one record, and transmit one record. The complete transfer requires one disc access, six FREADs, and six data transmissions.

c. If the remote file is defined as REC=-80,6,F, and opened with the NOBUF and multirecord option, then Remote File Access must access the disc to retrieve a block of six records, satisfy an FREAD of 480 bytes with six records, and transmit the six records. The complete data transfer requires one disc access, one FREAD, and one data transmission.

d. In PTOP, the master program can issue a PREAD. The slave program can pack the buffer with all six records, utilizing any of the above three methods. Note that a *linebuf* of 304 words is ample to permit transmission of 480 bytes (240 words) of data plus 34 words of DS/3000 header information in one transmission.

COORDINATING MASTER AND SLAVE PROGRAMS

PTOP communication programming requires synchronizing two separate programs at specific points in time. For this reason, it is often helpful to block diagram the transmissions and their contents on a simulated time line.

Where the PTOP programmer wants to loop on certain PTOP operations, the loop's terminating condition must, of course, be defined. The master program has direct control over the interprogram communications and can terminate a loop under conditions defined locally. More difficult are the situations when the slave must communicate to the master that the terminating condition has been met. To do this, the slave might send a REJECT response. A REJECT does not allow transmission of data, and so requires a terminating exchange of transmissions after all data has been transmitted.

Another method is to utilize the 20-word *itag* field (the *itag* parameter) of the PTOP intrinsics. This field is not accessible by the slave unless designated as a parameter in the corresponding master's PTOP operation. For example:

	Master Program	Slave Program
Example A.	`PREAD(dsnum, target, tcount);`	`GET(itag);`
Example B.	`PREAD(dsnum, target, tcount, itag);`	`GET(itag);`

In example A, PREAD does not utilize the *itag* field. The slave program cannot access *itag* on this transaction. The second PREAD (Example B) might not even initialize the *itag* array, but the array has been specified as a parameter. The slave program can now return control information to the master via this field. The master program logic can inspect *itag* and take corresponding action.

A PCONTROL from the master will also cause an exchange of *itag* fields and may be used for passing control information. This intrinsic will not pass a data field, however.

The control information passed between programs may terminate a loop, may branch to another part of the program, may transmit an index to be used in a CASE statement, or may serve any other purpose the programmer desires.

It is important to bear in mind the accessibility of transmitted data When the master program PWRITEs, the slave program cannot process the received data until the ACCEPT intrinsic has moved the data into the slave process stack. The slave program can, however, examine the *itag* array before doing the ACCEPT or REJECT. After examining the *itag*, the slave can then alter the *itag* array. The ACCEPT or REJECT will transmit the slave's *itag* to the master. Slave local processing can then proceed.

INTERPROCESS COMMUNICATION AND PTOP

Interprocess Communication (IPC) is a capability of the MPE file system that is very beneficial in the DS/3000 environment. For some applications, IPC may be easier to implement than Program-to-Program Communications (PTOP) and may provide other advantages as well. A basic description of the use of IPC and the changes made to the file system is included in the *MPE Intrinsics Reference Manual* and the *MPE File System Reference Manual.*

Message Files

Message files are used by IPC to transfer requests from one process to another. Each message file is a queue, with records acted upon in the order received. Message files are opened with FOPEN, read with FREAD, written to with FWRITE, and closed with FCLOSE.

When a message file is opened, the file system assigns a unique 16-bit ID number to the process that opens the file. Each record the process writes to the message file is associated with this number. When the writer closes the file, the ID number no longer applies, and may be reused.

When the files are opened with FOPEN, the user process accesses the file as either a reader or a writer; readers access the top of the message file, while writers access the tail. Readers access the file with FREAD. The record is copied to the reader, and is deleted from the message queue. Writers access the file with FWRITE. FWRITE appends one record to the tail of a message file. See Figure 8-2.

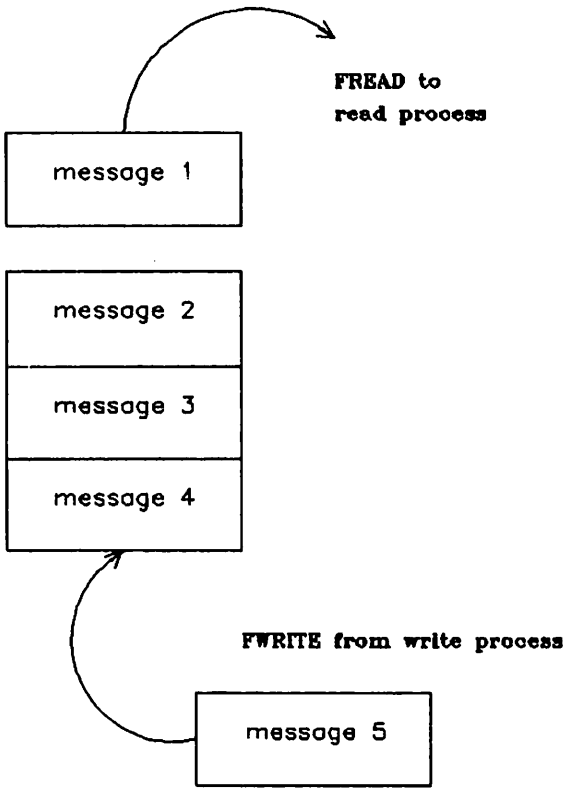


Figure 8-2. Reading and Writing to a Message File.

Message files can be created using the :BUILD command or the FOPEN intrinsic.

Example

A simple example of the use of IPC for communication between two remote sessions is presented in Figure 8-3. User Bill establishes a local session on Node A and a remote session on Node B. His application, called BILLPROG, opens a local MSGFILE as a reader and a remote MSGFILE as a writer. Then, user Jack establishes a local session on Node B and a remote session on Node A. Jack's application, called JACKPROG, opens a local MSGFILE as a reader and a remote MSGFILE as a writer. Now these two unrelated processes can communicate with each other through the IPC capability.

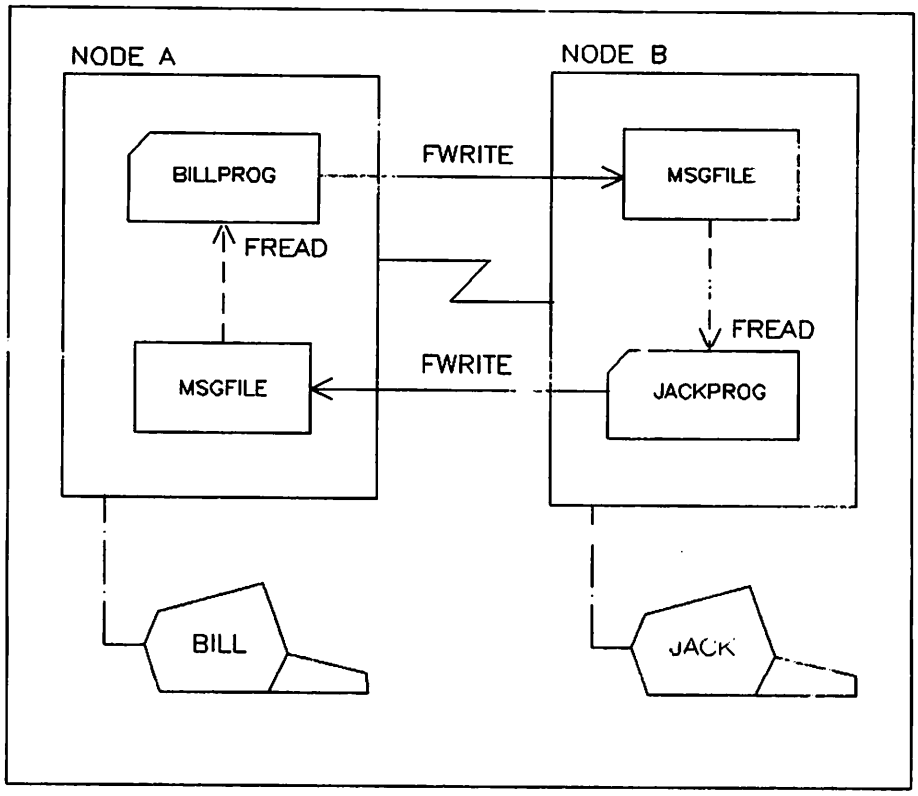


Figure 8-3. Two-node IPC Communication

If PTOP had been used in the example in Figure 8-3, a PTOP master program would need to be executing in one node and a slave program would have been initiated by the master in the other node. The master-slave programs would also require coordination because of their relationship.

The advantage of IPC becomes more dramatic when two or more processes desire to communicate with each other, or when the network is more complex than two nodes. Figure 8-4 shows a network consisting of three nodes and a solution that seems very useful in the general DS/3000 applications environment.

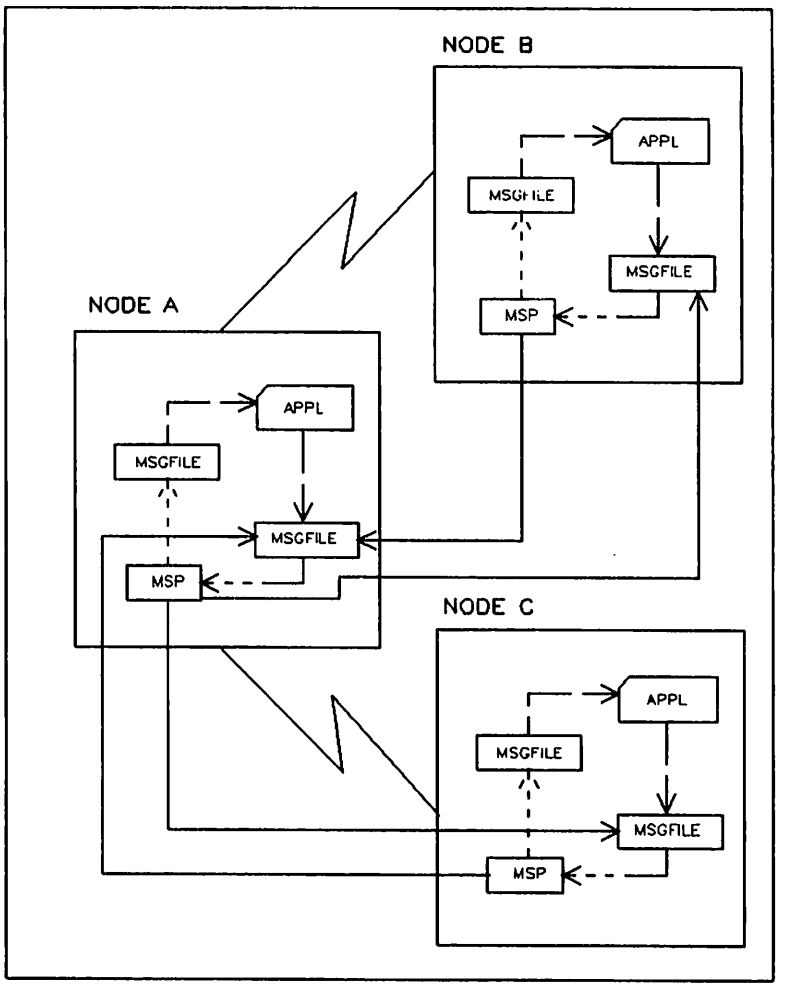


Figure 8-4. Three-node IPC Communication

In Figure 8-4, a general application program called a Message Switching Procedure (MSP) is written and executed on each node. The MSP performs the following functions:

- Opens a local message file as a reader.
- Opens any local applications message files as a writer.
- Opens all DS lines to adjacent nodes.
- Establishes a remote session on each of these nodes.
- Opens a message file on each adjacent node as a writer to be used for communication with each adjacent MSP.

The MSP handles all outgoing requests by forwarding them to the MSP programs on adjacent nodes. The MSP also handles all incoming requests by routing them to a local application program or by passing them on to the next node in the network.

If the network is complex and it is desirable to shift the responsibility for routing from the user to the MSP, a solution might include addressing within the user's data buffer and the use of a directory file in conjunction with the MSP. The MSP would then use the directory file to determine to what node it should forward the message. A more advanced directory file could provide alternate routes in case of downed lines. If alternate routes were not available, the unserviceable requests could be stored in a disc file and then be rewritten to the MSP's MSGFILE when the downed lines are restored.

In a simple network, it may not be desirable to design an MSP; but it is still possible that using IPC may be more advantageous than using PTOP. In this case, each user application could set up one or more remote sessions on the appropriate node(s) and communicate with other processes using the normal file intrinsics (FOPEN, FREAD, FWRITE, and FCLOSE) and message files. Also, by using the :FILE command, it can be transparent to the user or to the application program that the MSGFILE is located on a remote node.

The advantage of using an MSP is that several users on a system can communicate with a number of remote processes, but only one remote session is required per node. Since fewer remote sessions are necessary, the amount of memory required is decreased.

The major advantage of IPC versus PTOP is that there is no limitation to the number of local or remote processes with which a single process can communicate. The processes are fully bilateral with IPC making it easier to implement and expand the application for more complex networks. Also, activities such as development, testing, and debugging can all be done on one node, and then the resulting application can be distributed.

DEBUGGING

Where the amount of local processing in a PTOP application is significant, it may be helpful to debug the master and slave programs as local programs. MOVEs on dummy arrays or FREADs on dummy files can be substituted for communication operations to simplify debugging of the local processing.

When the time arrives to run the programs in master-slave fashion, a :RUN PROG;DEBUG is sufficient to invoke the Debug Utility for the master. This will not, however, allow the programmer to set a break-point in the slave program or to examine the slave process stack. To facilitate debugging a slave program, the first executable statement of the slave program should be the DEBUG intrinsic.

LINE BUFFERS/CONTINUATION BUFFERS

DS/3000 is designed to send across the line, in a single transfer operation, the amount of data configured as the PREFERRED BUFFER SIZE for the line controller (INP, SSLC, or HSI). The first person to use a DS line can override the configured line buffer size by specifying a different value with the *linebuf* parameter of a :DSLINE command.

When a user specifies *linebuf*=xxxx, the xxxx value tells the Communication Subsystem (CS/3000) the maximum amount of data DS/3000 will ever send across the line in a single request. For example, if you say *linebuf*=1074, you are saying the largest buffer DS/3000 can pass to the Communication Subsystem is 1074 words.

The 1074 words will always consist of both user data and DS/3000 fixed header and variable-length appendage characters. The additional characters (approximately 20 to 50 words) give to and from information, intrinsic names, etc., and vary for RFA and PTOP operations.

When a DS/3000 user requests the transfer of more data in a single operation than the line buffer can accomodate, DS/3000 automatically fills the line buffer, CS/3000 makes the transfer, then DS/3000 refills it, and transfers again -- until all of the user's data has been sent. When a user's single request causes DS/3000 to make several transfer operations. the additional buffers of data are known as "continuation buffers." As stated before, the ideal line buffer should be large enough to eliminate the need for continuation buffers.

COMPRESSION

Compression of data on the communications link may be specified in order to achieve higher throughput. DS/3000 initialization procedures allow compression only if both systems are capable of performing compression. Compression is handled on an individual basis, so that on a non-exclusive line, some users will compress while others will not.

Compression is most helpful in applications using line speeds up to 56 K bps. It is generally neither helpful nor desirable in applications that use higher data rates. It can be specified by the user in three different ways:

- At generation time by use of SUBTYPE=1 while configuring IODS0 or IODSX (refer to Section 1 in the *DS/3000 HP 3000 to HP 3000 Network Administrator Manual.* This configured subtype sets the default for the line.

- While executing the :DSCONTROL console command. A console operator uses the parameter to override the configured line default or to reset to the configured state.

- While executing a :DSLINE command in a session or job. Use of the :DSLINE parameter allows individual users to control whether or not their data will be compressed, and will override the current line default.

Compression generally increases throughput by reducing redundancy in the data, which results in a reduction in the number of characters being transmitted over the communications link. The compression technique compresses any occurance of three or more consecutive characters. Compression does not take place in the data only. It can occur in the fixed part or the appendage of the DS/3000 header.

The amount of redundancy in data or files may vary significantly. Source or listing files may be compressed by as much as 75 percent, but a more typical random assortment of HP 3000 files may be reduced by an amount closer to 25 percent. Obviously, the actual reduction will vary from application to application. Comparative tests with and without compression will indicate the benefits.

In some cases, however, compression could actually result in an increase in the number of characters to be transmitted. For this reason, DS/3000 examines each case when compression is specified. If a situation is found where compression would be detrimental to performance, DS/3000 sends the data uncompressed.

Using compression and decompression increases the system overhead at both ends of the link. The decision on whether to use compression depends on the communications link data rate, system load, and the amount of redundancy in the data being transmitted. Often, a test of relative throughput with normal system load and "typical" data will provide an indication of the benefits of using compression.

Formats for Inserted Compression Characters

Octal Value	Meaning
xx nnn nnn	xx = compression type 00 = uncompressed character string 10 = repeated blanks 11 = repeated non-blank characters (next byte is the ASCII value for the repeated character) nnn nnn = octal character count 1 to 77.

Examples:

036	36 (octal) non-compressed characters
217	17 (octal) blank characters
323.052	23 (octal) compressed * characters

For example, sending the string 1111111111 uncompressed resulted in the following string:

```
031.061  031.061  031.061  031.061  031.061
```

Sending the string compressed resulted in the following string:

```
312.061
```

where the 3 shows that there is a repeated non-blank character, the 12 means 12 octal (10 decimal) characters, and the 061 is the ASCII representation of 1.

PERFORMANCE

The performance achieved while using the DS/3000 link may vary widely, and it depends on many factors.

Computer System Dependent

The activity mix on the respective HP 3000 will affect performance. It depends upon the character of the simultaneous activity: such as the number of jobs, number of CPU-bound jobs and their relative priority, contention for disc, memory size and amount of swapping, quantum size, etc.

Communication Links

The choice of the communications link will provide an upper limit to the performance. Generally, a full-duplex line will outperform a half-duplex line by reducing line turnaround delays. A half-duplex line with a smaller request-to-send/clear-to-send delay will be faster (such as a 208B at 50 milliseconds versus a 208B at 150 milliseconds).

Line quality can result in wide variations in performance at times when line errors are high. A leased line is generally better and more predictable than a dial-up line. Some telephone offices provide cleaner lines depending on the age and nature of their switching gear.

Applications

For a given amount of data, the buffer size selected will affect performance. The smaller the number of requests required to transmit a given quantity of data, the higher the throughput. This also includes continuation requests. The data may be packed into larger buffers while using PTOP applications. The data may also be blocked into larger records for RFA applications. (RFA and FCOPY handle one record at a time, even though the file may use blocking)

As described earlier, use an appropriate line buffer size. Use a line buffer large enough to contain the full record or buffer, plus the DS/3000 fixed blocks and appendage header words. (The "rule-of-thumb" is 50 words larger than the data size.)

For applications to be run on dial-up lines, the line errors normally suggest a reasonable *linebuf* maximum of 1024 words. Analysis of `:SHOWCOM xx ;ERRORS` output and trace listings for error rates will allow modification of this recommendation for "typical" conditions. (This suggested maximum value of 1024 could be either increased or decreased when an SSLC is being used; but the value could only be decreased when the communications interface is an INP, since the maximum buffer size for the INP is 1024 words.)

PTOP applications allow both the master and slave programs to do a larger share of data searching, qualification, and manipulation at each local computer, thus reducing the quantity of data which must be sent across the line.

Remote Listing

Where data must be sent to a remote device (such as a line printer or a magnetic tape) it may be possible to send the program that generates the data to the remote computer for execution. For example, since a compiler listing can be quite large, it might be more efficient to transmit the source across the line and do a remote compilation and remote list, rather than doing a remote list for a local compilation.

MULTIPLE REMOTE ACCESS

While presenting the basic concepts of DS/3000 in the tutorial sections of this manual, the examples were intentionally limited to simple networks. From those somewhat simplistic illustrations, it might appear as though the only way your local computer can talk to more than one other computer is through additional parallel communication lines from your local system to the additional remote systems. Actually, it is possible to communicate with other remote computers in the network that have no direct connection with your local computer. This communication is made possible by going through one remote computer (to which you do have a direct communication line) to reach another remote computer to which the first remote computer is connected. To reach a second remote system through a first remote system, a multiple REMOTE command is used. The syntax is as follows:

:REMOTE [*xxx*] [REMOTE [*xxx*]] ... [*command*]

In this way, the local user can initiate a session sequentially on each remote system. See Figure 8-5.

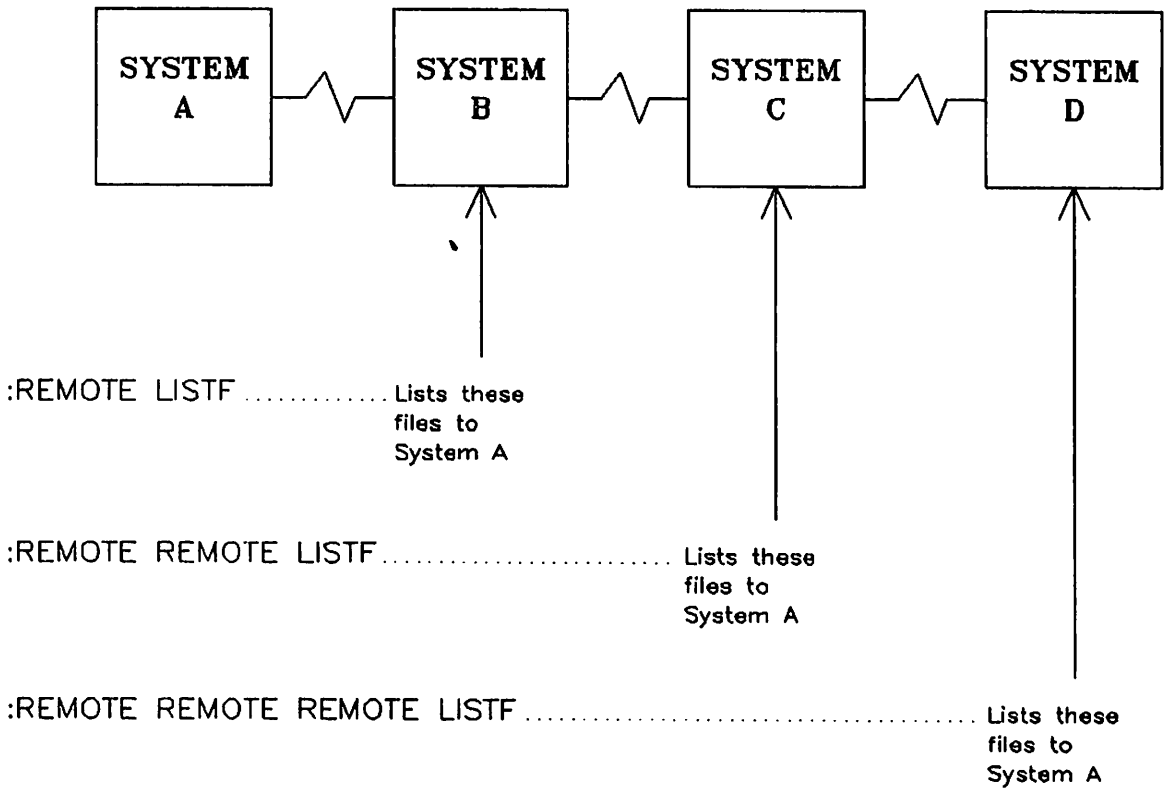


Figure 8-5. Multiple Remote Accessing Example

Figure 8-5 shows how your local system (System A) can obtain a list of the files in the first remote system (System B) by issuing the command:

:REMOTE LISTF

To obtain a similar list of files from System C in this kind of network (where the communications link is through an intermediate remote computer), use the command:

```
:REMOTE REMOTE LISTF
```

Likewise, you can route your request through to System D by expanding the command to:

```
:REMOTE REMOTE REMOTE LISTF
```

Using this compound command accomplishes the same result as if you had issued the following series of separate commands:

```
:REMOTE
#REMOTE
#REMOTE
#LISTF
```

There is an important difference in the way of returning to your local system, however. When you reach System D (Figure 8-4) by entering the compound command

```
:REMOTE REMOTE REMOTE
#
```

the # prompt is coming from the Command Interpreter (CI) on System D. If you now type a colon (:)

```
#:
:
```

you are being switched back to your local CI (System A). But if you were to use the alternative method of reaching System D with a series of separate commands

```
:REMOTE
#REMOTE
#REMOTE
#
```

and then you typed a colon as before, you would be switched to System C. To get back to your local system (System A), you must return a step at a time (just as you went out to System D a step at a time) as follows:

```
:REMOTE
#REMOTE
#REMOTE
#:
#:
#:
:
```

DSCONTROL CONSOLE COMMAND

SECTION 9

Before establishing a DS/3000 communications link, the console :DSCONTROL command must be used to OPEN a line, so that it is available to DS/3000 users. The :DSCONTROL command allows you to enable or disable the DS/3000 subsystem on a specific communications link.

For easy reference, this command is shown in the following format:

- SYNTAX — Shows the format of the command.
- PARAMETERS — Describes the variables in the command.
- OPERATION — Describes the command in detail.
- EXAMPLES — Shows the command in use.

:DSCONTROL

SYNTAX

```
:DSCONTROL dsdevice [function [... function]]
```

where the parameter *function* has the following options:

```
[;CANCEL]

[ ;OPEN  [,MASTER] [,[SPEED=]speed ] ]
  ;SHUT  [,SLAVE ]

[ ;TRACE {,ON  [,ALL][,mask][,numentries][,WRAP][,filename] } ]
         {,OFF                                              }

[ ;MON  [,DS] ]
  ;MOFF [,CS] ]

[ ;COMP   ]
  ;NOCOMP ]

[ ;RETRY={DEFAULT} ]
         {count   }
```

PARAMETERS

dsdevice

The logical device number or the device class name of the DS/3000 communications device (IODS0 or IODSX). On your system's I/O configuration listing, the device is back referenced by a pound sign (#) to a previously defined INP, HSI, or SSLC. (Required parameter.)

OPEN

Makes the line available for remote communication via the DS/3000 Subsystem. For bisync DS/3000 (IODS0), the DS subsystem is enabled, but no activity is initiated on the communications link. For DS/3000-X.25 (IODSX), the DS

subsystem establishes a communications link with the PDN. (Required parameter.)

CANCEL

Applicable only to X.21. Cancels all queued outgoing call requests. Sends an abort request to the communications device. Validates that the device is X.21 related and that there is a queued request. CANCEL is executed before any other :DSCONTROL parameter.

SHUT

Initiates an orderly line shutdown. Refer to OPERATION for details about the line closing procedure.

MASTER

Limits DS/3000 line activity to outgoing requests only. No incoming sessions are allowed.

SLAVE

Limits DS/3000 line activity to incoming requests only; no outgoing activity is allowed.

Default: Both MASTER and SLAVE processing are allowed.

linespeed

Transmission rate in characters per second (Bit Rate/8). This parameter is effective only if your system generation for the line selected SPEED CHANGEABLE. Specify *linespeed* if yours is a European installation with modems running at half speed, or if the line is hardwired and you want to override the configured default. It may be necessary to include this parameter if the length of cables used for HSI communications has been changed since the system was configured.

HSI speed: 250 000 (cable lengths less than 1000 ft.)

125 000 (cable lengths greater than 1000 ft.)

INP or SSLC speed: 250, 300, 600, or 1200

INP only speed: 2400 or 7200

The SPEED= keyword in the open option may be omitted from a :DSCONTROL command. For example, the following two commands have exactly the same effect:

```
:DSCONTROL 60;OPEN,MASTER,SPEED=250000

:DSCONTROL 60;OPEN,MASTER,250000
```

Remember, both ends of the line must operate at the same speed.

Default: System configuration values.

TRACE,ON	Activates the TRACE facility to provide a record of communications activities. Trace parameters are positional. The line must already be open, or the OPEN keyword must also be included (to open the line).
ALL	Generates trace records for all line activity. Default: Records are written only for transmission errors.
mask	An octal number preceded by a percent sign (*%nn*). Used to select type of trace entries generated. Refer to Sections 3 and 4 in the *DS/3000 Network Administrator's Manual* for an explanation of the mask bits. Default: %37 (all except PSTN).
numentries	Decimal integer for the maximum number of entries in a trace record, not greater than 248. Default: 24. (See OPERATION.)
WRAP	Trace entries that overflow the trace record overlay the prior trace record entries. Default: Overflow entries are discarded.
filename	A name for the trace file. Default: DSTRCxxx.PUB.SYS (where xxx is the LDEV of the *dsdevice*).
TRACE,OFF	Deactivates the TRACE facility, so that no records are kept of DS/3000 actions, states, and events. Also closes the trace file.
COMP	Sets data compression as the default mode of operation for all line users. The line need not be open to use COMP.
NOCOMP	Sets uncompressed data as the default mode of operation for all line users. The line need not be open to use NOCOMP.
MON [,DS ,CS]	Activates internal communication monitoring activity to give additional information on a subsequent cold dump of the system. The line must be open for the use of MON.

MON	Requests monitoring of all levels of activity.
MON,DS	Requests monitoring at the DS/3000 level of internal software operation.
MON,CS	Requests monitoring at the Communication System level of internal software operation.

Default: No monitoring.

Used only for system troubleshooting.

MOFF

Deactivates internal DS/3000 monitor records. Line must be open for the use of MOFF.

RETRY=[DEFAULT | *count*]

Changes the communications error retry count to the specified value. The retry counter controls the number of times the system attempts to send or receive a message across the communications link.

DEFAULT	Specifies a limit of 15 retries after a line error occurs.
count	Can be any value within the range of 0 to 255.

Default: 15.

OPERATION

Unless :DSCONTROL is issued from the master console, this command requires the user to have CS and ND capability. In addition, all users except the console operator are granted access to :DSCONTROL only if they are ALLOWed to use the command and are ASSOCIATEd with the specified DS device.

Only one DS/3000 communications device can be active (OPEN) on a controller at any given time. Once opened (with the :DSCONTROL command), a communications link can be shared by multiple DS/3000 users. It cannot, however, be shared by users of other communications subsystems supported by your system (for example, DSN/MRJE). Thus, you must SHUT the DS/3000 communications device before the controller can be opened for use by another subsystem.

Before issuing a :DSCONTROL command, use the :SHOWDEV command to check whether a communications link is already established. The LDEV for the INP, SSLC, or HSI port will be UNAVAILable if the communications link is in use by any subsystem; the LDEV for a DS/3000 communication device, driver IODS0 or IODSX, will be AVAILable if it is currently OPEN for use by DS/3000 users.

If a DS device class includes more than one DS device, the functions specified in the :DSCONTROL command apply to all devices in that class. If you have not been ALLOWed to use this command, you can only control those devices in the device class with which you have been ASSOCIATEd (if any).

:DSCONTROL

If you include more than one function in a :DSCONTROL command, each function (with its subparameter list) must be separated by a semicolon. A function that duplicates or conflicts with a previous function overrides that function. Functions can appear in any order but are executed in the following order:

1. CANCEL
2. OPEN/SHUT
3. TRACE
4. MON/MOFF
5. COMP/NOCOMP

The default name of the trace file is:

DSTRCxxx.PUB.SYS

where xxx is the logical device number of the *dsdevice*.

If no trace file exists when you turn on the trace facility and you do not specify *numentries*, the system creates a file to hold 24 entries in each record.

When using the bisync protocol, the SHUT parameter initiates an orderly line closing procedure. If no sessions or applications are using the line when you shut it, line disconnection occurs immediately. If any user (including applications) has the line open, the line remains connected until all sessions and applications CLOSE the line, or until those accessing the line terminate or are aborted. Once CLOSEd by the console operator, no new users may access the line until the operator reopens it. When using the X.25 capability of DS/3000, the SHUT parameter disconnects the line immediately, even if there are current users on the line.

NOTE

Occasionally you may not be able to SHUT a standard (non-X.25) DS line. This could happen, for example, if a DS user forgot to issue a :DSLINE xxx;CLOSE command but still has a local session. It could also happen if a remote session is "hung." In such a situation, you can "kill" all activity across the line by issuing an :ABORTIO xxx (where xxx is the logical device number of the *dsdevice*). Following the use of the :ABORTIO xxx command, a second :DSCONTROL xxx;SHUT command will complete successfully.

EXAMPLES

To open X.25 line number 55, thereby making it available for use by DS/3000 users, enter:

```
:DSCONTROL 55;OPEN
```

To permit the local HP 3000 to process only master (outgoing) requests on DS line number 55, enter:

```
:DSCONTROL 55;OPEN,MASTER
```

To activate the CS Trace facility for DS line 55 (the line is already open), enter:

```
:DSCONTROL 55;TRACE,ON,ALL
```

To open X.25 line 55 and activate CS Trace with a maximum of 250 entries in a trace record, enter:

```
:DSCONTROL 55;OPEN;TRACE,ON,,,250
```

To open the line named REMSYS and establish compression as a default and enable internal monitoring, enter:

```
:DSCONTROL REMSYS;OPEN;COMP;MON
```

ERROR CODES AND MESSAGES

APPENDIX A

The following is a summary of the error code numbers and messages that may be encountered. The messages, as listed here, have been grouped into several categories. For example, the first group contains all messages pertaining to :DSLINE syntax problems, while the second group contains the messages that report a DS/3000 functional problem. Each group is identified with an explanatory heading, and the messages are listed in numerical sequence within each category for easy reference.

:DSLINE SYNTAX ERRORS

These messages are sent to the terminal user to point out an error in syntax or to warn of the consequences of a request.

```
1300 REMOTE JOBS ARE NOT ALLOWED !.  (CIERR 1300)

1301 DSLINE CANNOT CONTAIN BOTH OPEN AND CLOSE.  (CIERR 1301)

1302 DSLINE REQUIRES AT LEAST ONE PARAMETER.  (CIERR 1302)

1303 DSNUMBER SPECIFICATION MUST BE A NUMBER FROM 1 THRU 255.
     (CIERR 1303)

1304 DSLINE #1! DOES NOT IDENTIFY AN OPEN DS LINE.  (CIERR 1304)

1305 EXPECTED LINEBUF, PHNUM, IOCID, REMID, OPEN, CLOSE,
     QUIET, COMP, NOCOMP, OR EXCLUSIVE.  (CIERR 1305)

1306 MULTIPLE USE OF ! IS NOT ALLOWED.  (CIERR 1306)

1307 THE SYNTAX FOR ! REQUIRES AN = SIGN FOLLOWED BY DATA.
     (CIERR 1307)

1308 PHNUM IS 1 TO 20 DIGITS AND DASHES.  (CIERR 1308)

1309 ! LIST CAN CONTAIN ONLY ONE ELEMENT.  (CIERR 1309)

1310 THE SPECIFIED LOGICAL DEVICE IS NOT OPEN.  (CIERR 1310)

1311 THE FIRST CHARACTER OF AN ID SEQUENCE MUST BE A " OR A (
     (CIERR 1311)

1312 THE ID SEQUENCE MUST TERMINATE WITH A ).  (CIERR 1312)

1313 THE ID SEQUENCE MUST TERMINATE WITH A ".  (CIERR 1313)

1314 A NUMERIC ID SEQUENCE ELEMENT MUST BE 1 THRU 255 (OR
     %377).  (CIERR 1314)
```

1315 LINEBUF MUST BE A NUMERIC VALUE FROM 304 THRU 4096.
(CIERR 1315)

1316 UNABLE TO COMPLETE THE REMOTE COMMAND. (CIERR 1316)

1317 NOT A CURRENTLY AVAILABLE DSLINE. (CIERR 1317)

1318 USE OF EXCLUSIVE REQUIRES BOTH NS AND CS CAPABILITY.
(CIERR 1318)

1319 THE DS LINE #L! IS IN USE BY A PROGRAM OR SUBSYSTEM AND
CANNOT BE CLOSED. (CIERR 1319)

1320 EXPECTED A RESPONSE OF YES, Y, NO, OR N. (CIERR 1320)

1321 UNABLE TO OPEN THE DS LINE ON DEVICE !. (CIERR 1321)

1322 @ IS INVALID IN THIS CONTEXT. (CIERR 1322)

1323 A DSLINE OPEN REQUIRES A VALID DS DEVICE NAME AS THE
FIRST PARAMETER. (CIERR 1323)

1324 FROM ADDRESS MUST BE BETWEEN 1 AND 14 CHARACTERS INCLUSIVE.
(CIERR 1324)

1325 TO ADDRESS MUST BE BETWEEN 1 AND 14 CHARACTERS INCLUSIVE.
(CIERR 1325)

1326 FROM AND TO ADDRESS MUST BE A DECIMAL NUMBER. (CIERR 1326)

1389 INVALID OR MISSING DELIMETER FOR SELECTION SIGNAL SEQUENCE.
(CIERR 1389)

1390 SELECTION SIGNAL SEQUENCE MUST BE FROM 1 TO 30 CHARACTERS.
(CIERR 1390)

1391 BOTH QUEUE AND NOQUEUE SPECIFIED; NOQUEUE USED. (CIWARN 1391)

1392 ONLY ! WORDS WERE ALLOCATED FOR THE LINE BUFFER.
(CIWARN 1392)

1393 COMPRESSION REQUEST NOT HONORED. REMOTE DOES NOT SUPPORT
THIS FEATURE. (CIWARN 1393)

1394 COMPRESSION PARAMETER RESPECIFIES AND OVERRIDES PREVIOUS
COMPRESSION PARAMETER. (CIWARN 1394)

1395 OPEN PARAMETERS ENTERED ON A CLOSE REQUEST ARE IGNORED
(CIERR 1395)

1396 AN ID LIST MUST CONTAIN 255 OR LESS ELEMENTS.
(CIWARN 1396)

1397 AN UNNECESSARY DELIMITER IS IGNORED. (CIWARN 1397)

1398 THERE ARE NO DS LINES OPEN. (CIWARN 1398)

1399 MULTIPLE USE OF ! IS REDUNDANT AND IGNORED. (CIWARN 1399)

DS/3000 FUNCTIONAL ERRORS

These messages report a functional problem within the system.

201 REMOTE DID NOT RESPOND WITH THE CORRECT REMOTE ID.
(DSERR 201)

202 SPECIFIED PHONE NUMBER IS INVALID. (DSERR 202)

203 REMOTE ABORT/RESUME NOT VALID WHEN DOING PROGRAM-TO-PROGRAM COMMUNICATION. USE LOCAL ABORT/RESUME.
(DSWARN 203)

204 UNABLE TO ALLOCATE AN EXTRA DATA SEGMENT FOR DS/3000.
(DSERR 204)

205 UNABLE TO EXPAND THE DS/3000 EXTRA DATA SEGMENT.
(DSERR 205)

206 SLAVE PTOP FUNCTION ISSUED FROM A MASTER PROGRAM.
(DSERR 206)

207 SLAVE PTOP FUNCTION OUT OF SEQUENCE. (DSERR 207)

208 MASTER PTOP FUNCTION ISSUED BY A SLAVE PROGRAM.
(DSERR 208)

209 SLAVE PROGRAM DOES NOT EXIST OR IS NOT PROGRAM FILE.
(DSERR 209)
Creation of Slave Program failed, possibly because of invalid stack size.

210 WARNING -- INVALID MAXDATA OR DLSIZE FOR A SLAVE PROGRAM. SYSTEM DEFAULTS ARE IN EFFECT. (DSWARN 210)

211 SLAVE ISSUED A REJECT TO A MASTER PTOP OPERATION.
(DSWARN 211)

212 FILE NUMBER FROM IOWAIT NOT A DS LINE NUMBER. (DSWARN 212)

213 EXCLUSIVE USE OF A DS LINE REQUIRES BOTH ND AND CS CAPABILITY. (DSERR 213)

214 THE REQUESTED DS LINE HAS NOT BEEN OPEN WITH A USER :DSLINE COMMAND OR A REQUIRED :REMOTE HELLO HAS NOT BEEN DONE.
(DSERR 214)

215 DSLINE CANNOT BE ISSUED BACK TO THE MASTER COMPUTER.
(DSERR 215)

216 MESSAGE REJECTED BY THE REMOTE COMPUTER. (DSERR 216)

217 INSUFFICIENT AMOUNT OF USER STACK AVAILABLE. (DSERR 217)

218 INVALID PTOP FUNCTION REQUESTED. (DSERR 218)

219 MULTIPLE POPEN. ONLY ONE MASTER PTOP OPERATION CAN BE
ACTIVE ON A DS LINE. (DSERR 219)

220 PROGRAM EXECUTING GET WAS NOT CREATED BY POPEN. (DSERR 220)

221 INVALID DS MESSAGE FORMAT. INTERNAL DS ERROR. (DSERR 221)

222 MASTER PTOP FUNCTION ISSUED PRIOR TO A POPEN. (DSERR 222)

223 REQUEST TO SEND MORE DATA THAN SPECIFIED IN POPEN.
(DSERR 223)

224 FILE EQUATIONS FOR A REMOTE FILE CONSTITUTE A LOOP.
(DSERR 224)

225 CANNOT ISSUE POPEN TO A SLAVE SESSION IN BREAK MODE.
(DSERR 225)

226 SLAVE PROGRAM HAS TERMINATED BEFORE EXECUTING "GET".
(DSERR 226)

227 REMOTE HELLO MUST BE DONE TO INITIATE REMOTE SESSION.
(DSERR 227)

228 EXCEEDED MAXIMUM NUMBER OF VIRTUAL CHANNELS PER JOB.
(DSERR 228)

231 INVALID FACILITY IN CONNECTION REQUEST. (DSERR 231)

232 THE REMOTE COMPUTER IS NOT OBTAINABLE. (DSERR 232)

233 VIRTUAL CIRCUIT IS NOT AVAILABLE. (DSERR 233)

234 QUEUEING IS REQUIRED TO COMPLETE THE REQUEST. (DSERR 234)

235 DS MESSAGE SEQUENCING ERROR. (DSERR 235)

236 COMMUNICATIONS HARDWARE HAS DETECTED AN ERROR. (DSERR 236)

237 CANNOT CURRENTLY GAIN ACCESS TO THE TRACE FILE. (DSERR 237)

238 COMMUNICATIONS INTERFACE ERROR. INTERNAL FAILURE.
(DSERR 238)

239 COMMUNICATIONS INTERFACE ERROR. TRACE MALFUNCTION.
(DSERR 239)

240 LOCAL COMMUNICATION LINE WAS NOT OPENED BY OPERATOR.
(DSERR 240)

241 DS LINE IN USE EXCLUSIVELY OR BY ANOTHER SUBSYSTEM.
(DSERR 241)

242 INTERNAL DS SOFTWARE ERROR ENCOUNTERED. (DSERR 242)

243 REMOTE OR PDN IS NOT RESPONDING. (DSERR 243)

244 COMMUNICATIONS INTERFACE ERROR. LINE RESET OCCURRED.
(DSERR 244)

245 COMMUNICATIONS INTERFACE ERROR. RECEIVE TIMEOUT.
(DSERR 245)

246 COMMUNICATIONS INTERFACE ERROR. REMOTE DISCONNECTED.
(DSERR 246)

247 COMMUNICATIONS INTERFACE ERROR. LOCAL TIME OUT. (DSERR 247)

248 COMMUNICATIONS INTERFACE ERROR. CONNECT TIME OUT.
(DSERR 248)

249 COMMUNICATIONS INTERFACE ERROR. REMOTE REJECTED
CONNECTION. (DSERR 249)

250 COMMUNICATIONS INTERFACE ERROR. CARRIER LOST. (DSERR 250)

251 COMMUNICATIONS INTERFACE ERROR. LOCAL DATA SET FOR THE
DS LINE WENT NOT READY. (DSERR 251)

252 COMMUNICATIONS INTERFACE ERROR. HARDWARE FAILURE.
(DSERR 252)

253 COMMUNICATIONS INTERFACE ERROR. NEGATIVE RESPONSE TO THE
DIAL REQUEST BY THE OPERATOR. (DSERR 253)

254 COMMUNICATIONS INTERFACE ERROR. INVALID I/O CONFIGURATION.
(DSERR 254)

255 COMMUNICATIONS INTERFACE ERROR. UNANTICIPATED CONDITION.
(DSERR 255)

256 REQUEST QUEUED BEHIND PREVIOUS REQUEST.

:DSCONTROL INFORMATORY MESSAGES

These messages convey status information.

```
300 DS DEVICE !: MASTER AND SLAVE ACCESS SHUT.

301 DS DEVICE !: SLAVE ACCESS OPENED; MASTER ACCESS SHUT.

302 DS DEVICE !: MASTER ACCESS OPENED; SLAVE ACCESS SHUT.

303 DS DEVICE !: MASTER AND SLAVE ACCESS OPENED.

304 DS DEVICE !: TRACE ACTIVATED USING TRACE FILE !.

305 DS DEVICE !: TRACE DEACTIVATED.

306 DS DEVICE !: MONITORING ACTIVATED.

307 DS DEVICE !: MONITORING DEACTIVATED.

308 DS DEVICE !: DEBUG MODE ACTIVATED.

309 DS DEVICE !: DEBUG MODE DEACTIVATED.

310 DS DEVICE !: SPECIAL DEBUG MODE ACTIVATED.

311 DS DEVICE !: DEFAULT MODE IS NO COMPRESSION.

312 DS DEVICE !: DEFAULT MODE IS COMPRESSION.

313 DS DEVICE !: RETRY COUNT NOW EQUALS !.

314 DS DEVICE !: CALL REQUEST CANCELED.
```

:DSCONTROL ERROR MESSAGES

These messages point out an error in syntax or warn of the consequences of a request.

```
4100 NUMBER OF PARAMETERS EXCEEDS MAXIMUM OF !.  (CIERR 4100)

4101 EXPECTED AT LEAST TWO PARAMETERS:  A DS DEVICE
     CLASS/NUMBER AND A FUNCTION KEYWORD.  (CIERR 4101)

4102 EXPECTED A DEVICE CLASS NAME OR LOGICAL DEVICE NUMBER FOR
     ONE OR MORE DS DEVICES.  (CIERR 4102)

4103 USER IS NOT ASSOCIATED WITH DS DEVICE !.  NO CONTROL
     FUNCTIONS EXECUTED FOR THIS DEVICE.  (CIWARN 4103)
```

4104 USER IS NOT ALLOWED TO USE :DSCONTROL AND IS NOT ASSOCIATED WITH THE DS DEVICE(S). (CIERR 4104)

4105 EXPECTED ONE OR MORE OF THE CONTROL FUNCTIONS: OPEN, SHUT, MON, MOFF, COMP, NOCOMP, TRACE, OR DEBUG. (CIERR 4105)

4106 INVALID CONTROL FUNCTION. EXPECTED ONE OF: OPEN, SHUT, MON, MOFF, COMP, NOCOMP, TRACE, OR DEBUG. (CIERR 4106)

4107 MASTER OVERRIDES PREVIOUS MASTER/SLAVE OPTION. (CIWARN 4107)

4108 SLAVE OVERRIDES PREVIOUS MASTER/SLAVE OPTION. (CIWARN 4108)

4109 SPEED OPTION OVERRIDES PREVIOUS SPEED OPTION. (CIWARN 4109)

4110 OPEN OVERRIDES PREVIOUS OPEN/SHUT FUNCTION. (CIWARN 4110)

4111 SHUT OVERRIDES PREVIOUS OPEN/SHUT FUNCTION. (CIWARN 4111)

4112 TRACE OVERRIDES PREVIOUS TRACE FUNCTION(S). (CIWARN 4112)

4113 DEBUG OVERRIDES PREVIOUS DEBUG FUNCTION(S). (CIWARN 4113)

4114 MON OVERRIDES PREVIOUS MON/MOFF FUNCTION. (CIWARN 4114)

4115 MOFF OVERRIDES PREVIOUS MON/MOFF FUNCTION. (CIWARN 4115)

4116 COMP OVERRIDES PREVIOUS COMP/NOCOMP FUNCTION. (CIWARN 4116)

4117 NOCOMP OVERRIDES PREVIOUS COMP/NOCOMP FUNCTION. (CIWARN 4117)

4118 EXPECTED A ";" , "," , OR RETURN AS DELIMITER. (CIERR 4118)

4119 EXPECTED EITHER A ";" OR RETURN AS DELIMITER. (CIERR 4119)

4120 EXPECTED A "=" AS DELIMITER FOR SPEED OPTION. (CIERR 4120)

4121 EXPECTED A "," AS DELIMITER BETWEEN OPTIONS. (CIERR 4121)

4122 ILLEGAL OPEN/SHUT OPTION. EXPECTED ONE OF: MASTER, SLAVE, SPEED, OR LINESPEED VALUE. (CIERR 4122)

4123 EXPECTED A POSITIVE DOUBLE VALUE FOR LINESPEED. (CIERR 4123)

4124 CS CAPABILITY REQUIRED TO USE :DSCONTROL. (CIERR 4124)

4125 PM CAPABILITY REQUIRED TO USE DEBUG FUNCTION. (CIERR 4125)

4126 DEBUG FUNCTION MAY ONLY BE USED BY SYSTEM CONSOLE.
(CIERR 4126)

4127 EXPECTED NO OPTION FOR DEBUG OR ONE OF THE FOLLOWING:
ON, OFF, OR POSITIVE INTEGER VALUE. (CIERR 4127)

4128 EXPECTED NO OPTION FOR MON/MOFF OR ONE OF THE FOLLOWING:
CS OR DS. (CIERR 4128)

4129 COMP/NOCOMP FUNCTIONS HAVE NO OPTIONS. (CIERR 4129)

4130 SPEED OPTION IGNORED FOR SHUT FUNCTION. (CIWARN 4130)

4131 EXTRANEOUS ";" IGNORED. POSSIBLE MISSING FUNCTION?
(CIWARN 4131)

4132 EXTRANEOUS "," IGNORED. POSSIBLE MISSING OPTION?
(CIWARN 4132)

4133 CREATION OF DS MONITOR PROCESS FAILED. (CIERR 4133)

4134 PROGRAM FILE "DSMON.PUB.SYS" MISSING. (CIERR 4134)

4135 DS MONITOR UNABLE TO RUN AS A SYSTEM PROCESS. (CIERR 4135)

4136 CS DEVICE ! IS UNAVAILABLE FOR USE. (CIERR 4136)

4137 DS DEVICE MUST BE OPEN PRIOR TO USE. (CIERR 4137)

4138 USER SPECIFIED TRACE FILE NOT ALLOWED WHEN MORE THAN ONE
DEVICE IN DEVICE CLASS. (CIERR 4138)

4139 DS DEVICE ! CURRENTLY CONTROLLED ELSEWHERE. (CIWARN 4139)

4140 DS DEVICE !: OPEN/SHUT NOT EXECUTED DUE TO ABOVE.
(CIWARN 4140)

4141 DS DEVICE !: TRACE NOT EXECUTED DUE TO ABOVE. (CIWARN 4141)

4142 DS DEVICE !: MON/MOFF NOT EXECUTED DUE TO ABOVE.
(CIWARN 4142)

4143 DS DEVICE !: COMP/NOCOMP NOT EXECUTED DUE TO ABOVE.
(CIWARN 4143)

4144 DS DEVICE !: DEBUG NOT EXECUTED DUE TO ABOVE. (CIWARN 4144)

4145 NO DS DEVICES REMAINING TO BE CONTROLLED. (CIWARN 4145)

4146 RETRY OVERRIDES PREVIOUS RETRY FUNCTION. (CIWARN 4146)

4147 EXPECTED AN "=" AS DELIMITER FOR RETRY FUNCTION.
(CIERR 4147)

4148 INVALID RETRY COUNT, MUST SPECIFY "DEFAULT" OR A NUMBER BETWEEN 0 AND 255 INCLUSIVE. (CIERR 4148)

4149 DS DEVICE !: RETRY NOT EXECUTED DUE TO ABOVE. (CIWARN 4149)

4150 DS INTERNAL FIX NUMBERS DIFFER. (CIWARN 4150)

4151 INCOMPATIBLE OR MISSING NONCRITICAL DS MODULE: DSCOPY, DSTEST, DS2026, OR DS2026CN. (CIWARN 4151)

4152 CRITICAL DS MODULES ARE INCOMPATIBLE, NO CONTROL FUNCTIONS EXECUTED. (CIERR 4152)

4153 MISSING CRITICAL DS SOFTWARE, NO CONTROL FUNCTIONS EXECUTED. (CIERR 4153)

4155 PROMPT OVERRIDES PREVIOUS PROMPT FUNCTION(S). (CIERR 4155)

4180 REDUNDANT SPECIFICATION OF CANCEL OPTION IGNORED. (CIWARN 4180)

4181 CANCEL OPTION HAS NO PARAMETERS. (CIERR 4181)

4182 DS DEVICE ! IS NOT AN X.21 DEVICE; CANCEL NOT EXECUTED. (CIWARN 4182)

4183 NO CALL REQUEST FOR DS DEVICE !. (CIWARN 4183)

:DSCOPY GENERAL ERROR MESSAGES

0 SUCCEEDED.

1 SUCCESSFULLY INITIATED.

4 UNABLE TO OPEN TRANSACTION FILE. (NFTERR 4)

5 UNABLE TO OPEN LIST FILE (DSCOPYI). (NFTERR 5)

6 IC ERROR ON TRANSACTION FILE. (NFTERR 6)

7 TRANSACTION RECORD > 200 CHARS LONG. (NFTERR 7)

9 TEMPORARY TRANSACTION FILE FULL. (NFTERR 9)

10 PARAMETERS IMPLY CONFLICTING MODES. (NFTERR 10)

11 CAN'T "RUN" COPY PROCESS IN THIS MODE. (NFTERR 11)

13 UNRECOGNIZED PARAMETER. (NFTERR 13)

14 CONFLICTING OPTIONS HAVE BEEN SPECIFIED. (NFTERR 14)

16 UNIMPLEMENTED FEATURE. (NFTERR 16)

17 CANNOT CONTACT REMOTE NODE. (NFTERR 17)

18 FILE SYSTEM ERROR ON SOURCE FILE. (NFTERR 18)

19 FILE SYSTEM ERROR ON TARGET FILE. (NFTERR 19)

21 ILLEGAL DSLINE NAME. (NFTERR 21)

24 UNSUPPORTED STANDARD DEVICE TYPE. (NFTERR 24)

25 CAN'T FIND OR OPEN THE SOURCE FILE. (NFTERR 25)

26 CAN'T CREATE OR OPEN THE TARGET FILE. (NFTERR 26)

27 CANNOT CONTACT REMOTE SYSTEM. (NFTERR 27)

28 SOURCE AND TARGET FILES CANNOT BE ACCESSED THROUGH REMOTE FILE ACCESS. (NFTERR 28)

29 COMMUNICATION IO ERROR. (NFTERR 29)

30 INSUFFICIENT CAPABILITIES. (NFTERR 30)

33 NO SOURCE FILE WAS SPECIFIED. (NFTERR 33)

36 DS/3000 HAS NOT BEEN INSTALLED ON THIS SYSTEM. (NFTERR 36)

37 REMOTE SYSTEM UNABLE TO USE TRANSPARENT MODE. (NFTERR 37)

38 CAN'T FIND THE EXTRA DATA SEGMENT, USE THE DSCOPY INTRINSIC TO INVOKE NFT. (NFTERR 38)

39 INVALID EXTRA DATA SEGMENT CONTENTS, USE THE DSCOPY INTRINSIC TO INVOKE NFT. (NFTERR 39)

40 NEGOTIATIONS FAILED, NO COPY CAN BE PERFORMED. (NFTERR 40)

41 FILE TRANSFER ABORTED. (NFTERR 41)

42 COPY CANCELLED BY USER. (NFTERR 42)

:DSCOPY INTRINSIC ERROR RETURNS

80 BOUNDS VIOLATION. (NFTERR 80)

81 SPLITSTACK MODE CALLS NOT ALLOWED. (NFTERR 81)

82 FIRST PARAMETER VALUE IS OUT OF RANGE (-1:6). (NFTERR 82)

83 SECOND PARAMETER TOO SHORT TO CONTAIN VERSION STRING. (NFTERR 83)

84 NFT PROCESS IS BUSY, CAN'T START NEW TRANSACTION. (NFTERR 84)

85 NFT PROCESS IS NOT RUNNING. (NFTERR 85)

86 ILLEGAL BASIC CALLING SEQUENCE. (NFTERR 86)

:DSCOPY INTERNAL ERRORS

101 INTERNAL ERROR ON REMOTE SYSTEM. (NFTERR 101)

102 REMOTE SYSTEM NFT VERSION IS INCOMPATIBLE. (NFTERR 102)

103 INTERNAL - STRING STORAGE OVERFLOW. (NFTERR 103)

104 UNABLE TO CREATE TEMPORARY TRANSACTION FILE. (NFTERR 104)

105 AN UNEXPECTED MESSAGE WAS RECEIVED. (NFTERR 105)

106 AN ILLEGAL VALUE WAS RECEIVED IN A MESSAGE. (NFTERR 106)

107 A MESSAGE RECEIVED IN INVALID FORMAT. (NFTERR 107)

108 A REQUIRED ELEMENT WAS MISSING FROM A RECEIVED MESSAGE. (NFTERR 108)

109 NFT PROCESS CREATE FAILED. (NFTERR 109)

110 ATTEMPT TO GET EXTRA DATA SEGMENT FAILED. (NFTERR 110)

X.21 MESSAGES

Set 1: Call Progress Signals

1 ! LDEV !/CPS 001 TERMINAL CALLED

2 ! LDEV !/CPS 002 REDIRECTED CALL

3 ! LDEV !/CPS 003 CONNECT WHEN FREE

20! LDEV !/CPS 020 NO CONNECTION

21! LDEV !/CPS 021 NUMBER BUSY

22! LDEV !/CPS 022 SELECTION SIGNALS PROCEDURE ERROR

23! LDEV !/CPS 023 SELECTION SIGNAL TRANSMISSION ERROR

41! LDEV !/CPS 041 ACCESS BARRED

42! LDEV !/CPS 042 CHANGED NUMBER

43! LDEV !/CPS 043 NOT OBTAINABLE

44! LDEV !/CPS 044 OUT OF ORDER

45! LDEV !/CPS 045 CONTROLLED NOT READY

46! LDEV !/CPS 046 UNCONTROLLED NOT READY

47! LDEV !/CPS 047 DCE POWER OFF

48! LDEV !/CPS 048 INVALID FACILITY REQUEST

49! LDEV !/CPS 049 NETWORK FAULT IN LOCAL LOOP

61! LDEV !/CPS 061 NETWORK CONGESTION

71! LDEV !/CPS 071 LONG TERM NETWORK CONGESTION

81! LDEV !/CPS 081 REGISTRATION/CANCELLATION CONFIRMED

82! LDEV !/CPS 082 REDIRECTION ACTIVATED

83! LDEV !/CPS 083 REDIRECTION DEACTIVATED

Set 2: DCE Provided Information

1 ! LDEV !/NPI 001 CHARGE ADVICE - MONETARY CHARGES !

2 ! LDEV !/NPI 002 CHARGE ADVICE - DURATION (SECONDS) !

3 ! LDEV !/NPI 003 CHARGE ADVICE - UNITS !

10! LDEV !/NPI 010 LINE IDENTIFICATION !

20! LDEV !/NPI 020 SIGNAL FORMATTING ERROR !

DS/3000 COBOL INTERFACE

APPENDIX B

CONVENTIONS

To call an external procedure from ANS COBOL (COBOL/I), the parameters must be passed by word reference. This requirement effectively prevents the COBOL/I user from calling system-level intrinsics in general, and specifically, the DS/3000 Program-to-Program intrinsics. The following interface routines are provided to allow the ANS COBOL user access to the program-to-program communications capability. The user of COBOL II/3000 need not use these interface intrinsics, since the call-by-value capability can access the Program-to-Program intrinsics (as outlined in the *COBOL/II Reference Manual*).

The parameters in the COBOL calling sequences must be of the following types:

If the parameter is an integer, it must be a COBOL picture 9 through 9(4) or S9(3) computational, synchronized.

If the parameter is a character string, it must be defined as COBOL picture X(*n*) or A(*n*), where *n* is large enough for the required number of characters.

In the following parameters, those not specifically defined as characters will be assumed to be integers.

It is assumed that the user is already familiar with DS/3000, in general, and the program-to-program intrinsics, specifically. Information regarding formal usage or content of the interface intrinsic parameters can be found in Section 6.

COMMON PARAMETERS

Parameters whose use is the same through all the procedures are:

ccode — integer (required)
The condition code returned by the PTOP intrinsic.
-1 = CCL
0 = CCE
1 = CCG

Refer to Section 6 for the meaning associated with these codes in individual PTOP calls.

dsnum — integer (required)
The number returned by CPOPEN, and which is required for all subsequent master PTOP calls. The number is always 0 for slave programs.

itag — A 20-word integer field (optional).

target — A character field used for reading or writing data (required).

tcount integer (required).
The number of words or bytes to be read or written. Words are a positive integer; bytes are negative.

INTERFACE INTRINSICS

CPOPEN

This procedure is the COBOL callable interface to POPEN.

Calling Sequence:

```
CALL "CPOPEN" USING ccode,dsnum, dsdevice, progname, itag, entryname, parm,
flags, stacksize, dlsize, maxdata, buffsize
```

Where:

dsdevice is a character field containing the node name, device class, or logical device number of the desired DS line.

progname is a character field containing the name (terminated by a space) of the remote slave program.

itag is the 20-word integer field sent to and received from the remote program.

entryname is the character field specifying the secondary entry point (or spaces) where the remote program will begin execution. It is ignored if the slave program runs on an HP 1000.

parm is an integer value to be placed in Q-4 of the slave program. It is ignored if the slave program runs on an HP 1000.

flags
stacksize
dlsize
maxdata
are all MPE parameters used to specify slave program loading options. See Section 6 of this manual or the *MPE Intrinsics Reference Manual* for usage. It is ignored if the slave system is an RTE system.

buffsize is an integer specifying the maximum number of words which will be transferred by any of the PTOP intrinsics.

CPREAD

This procedure is the COBOL callable interface to PREAD.

Calling Sequence:

```
CALL "CPREAD" USING ccode,dsnum,length,target,tcount,itag
```

Where:

length is the actual number of words or bytes (depending on the value of *tcount*) read into *target*. (Required.)

CPWRITE

This procedure is the COBOL callable interface to PWRITE.

Calling Sequence:

```
CALL "CPWRITE" USING ccode,dsnum,target,tcount,itag
```

CPCONTROL

This procedure is the COBOL callable interface to PCONTROL.

Calling Sequence:

```
CALL "CPCONTROL" USING ccode,dsnum,itag
```

CPCLOSE

This procedure is the COBOL callable interface to PCLOSE.

Calling Sequence:

```
CALL "CPCLOSE" USING ccode,dsnum
```

CGET

This procedure is the COBOL callable interface to GET.

Calling Sequence:

```
CALL "CGET" USING ccode,ifun,itag,il,ionumber
```

Where:

ifun is the function number of the current pending PTOP operation. (Required.)
- 0 = An error occurred. This value is returned only when the condition code CCL is also returned. Issue a PCHECK intrinsic call (with a dsnum parameter of zero) to determine what happened.
- 1 = POPEN request received.
- 2 = PREAD request received.
- 3 = PWRITE request received.
- 4 = PCONTROL request received.
- 5 = This value is returned only when the condition code CCG is also returned. It indicates that a pending MPE File System I/O without wait request was completed (instead of the expected remote DS/3000 I/O request). *ionumber* contains the file number associated with the completed I/O request.

il is the number of words sent by a PWRITE or the number of words requested by a PREAD.

ionumber is the file number of a non-DS file which completed an I/O without wait.

CACCEPT

This procedure is the COBOL callable interface to ACCEPT.

Calling Sequence:

```
CALL "CACCEPT" USING ccode,itag,target,tcount
```

CREJECT

This procedure is the COBOL callable interface to REJECT.

Calling Sequence:

```
CALL "CREJECT" USING ccode,itag
```

CPCHECK

This procedure is the COBOL callable interface to PCHECK.

Calling Sequence:

```
CALL "CPCHECK" USING ccode,dsnum,icode
```

Where:

dsnum integer (required for master, optional for slave).
For a master program, this number is returned by the CPOPEN call. For a slave program, this number is always 0.

iocode is an integer identifying the last error encountered. The error code meanings are given in Appendix A, under "DS/3000 Functional Errors."

EXAMPLE

The following example illustrates how two COBOL programs, residing on two HP 3000 computers, pass data back and forth. These two programs demonstrate and test the intrinsics available to the user of the COBOL Program-to-Program facility of DS/3000. The Slave program must be entered on the remote system, compiled, and PREPed before the test. The PREPed file must then be made a permanent file. In this example, the MPE commands were:

```
:COBOL COBOLSS (COBOL Slave Source)
:PREP $OLDPASS, COBOLS
:SAVE COBOLS
```

The Master program must then be entered, compiled, PREPed, and run on the local system.

A brief outline of the test follows:

1. The Master program opens the Slave program with CPOPEN. The *itag* array is filled with the value of the subscript of each array element, and the CPOPEN intrinsic is called. The Slave displays certain parameters involved in the opening. Then the Master also displays the value of the parameters used for opening the remote program. After each call is made to a COBOL intrinsic, the status of the call is checked in the STATUS-CK-RTN paragraph.

2. The Master next tests the CPREAD intrinsic by requesting that a message from the Slave be sent back.

3. CPWRITE is tested by sending a message to the Slave. The Slave then displays the message as it was received to demonstrate the validity of the text.

4. The CREJECT-TEST paragraph of the Master is used to test the CPREJECT intrinsic available to the Slave as well as the CPCONTROL intrinsic of the Master. The value 14 is moved into the first element of ITAG and CPCONTROL is called. Within the paragraph that handles a call to CPCONTROL, the Slave tests this value and rejects the request.

5. The master then calls CPCLOSE to close the remote program before terminating itself.

The individual programs are shown on the following pages.

Master PTOP Program

```
001000$CONTROL USLINIT,SOURCE,MAP
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. MASTER-COBOL.
001300 ENVIRONMENT DIVISION.
001400 DATA DIVISION.
001500*************************************
001600 WORKING-STORAGE SECTION.
001700 77 CCODE        PIC S99    COMP  VALUE 0.
001800 77 DSNUM        PIC S99    COMP  VALUE 0.
001900 77 PARAM        PIC S99    COMP  VALUE 0.
002000 77 FLAGS        PIC S99    COMP  VALUE 33.
002100 77 STACKSIZE    PIC S9(4)  COMP  VALUE IS -1.
002200 77 DLSIZE       PIC S9(4)  COMP  VALUE IS -1.
002300 77 MAXDATA      PIC S9999  COMP  VALUE IS 8000.
002400 77 BUFSIZE      PIC S999   COMP  VALUE IS 304.
002500 77 LGTH         PIC S99    COMP  VALUE 0.
002600 77 ICODE        PIC S99    COMP  VALUE 0.
002700 77 TCOUNT       PIC S99    COMP  VALUE IS 33.
002800 77 I            PIC S99    COMP  VALUE 0.
002900 77 DATA-BUF     PIC X(66)        VALUE SPACES.
003000*********************************************
003100 77 A-DOLLAR     PIC X(12) VALUE "<< ACCEPT >>".
003200 77 K-DOLLAR     PIC X(12) VALUE "## PCHECK ##".
003300 77 M-DOLLAR     PIC X(26) VALUE "######### MASTER #########".
003400 77 O-DOLLAR     PIC X(11) VALUE "## POPEN ##".
003500 77 R-DOLLAR     PIC X(11) VALUE "## PREAD ##".
003600 77 S-DOLLAR     PIC X(18) VALUE "## STATUS CHECK ##".
003700 77 W-DOLLAR     PIC X(12) VALUE "## PWRITE ##".
003800 77 CO-DOLLAR    PIC X(14) VALUE "## PCONTROL ##".
003900 77 C9-DOLLAR    PIC X(12) VALUE "## PCLOSE ##".
```

```
004000 77 D1-DOLLAR   PIC XXXX  VALUE "INDY".
004100 77 E0-DOLLAR   PIC XX    VALUE "  ".
004200 77 P0-DOLLAR   PIC X(7)  VALUE "COBOLS ".
004300 77 R0-DOLLAR   PIC X(12) VALUE "<< REJECT >>".
004400*********************************************
004500 01 ITAG-ARRAY.
004600     02 ITAG-ARRAY-MEM PIC 99 OCCURS 20 TIMES.
004700*********************************************
004800*            PROCEDURE DIVISION             *
004900*********************************************
005000 PROCEDURE DIVISION.
005100 DRIVER-PARA.
005200     PERFORM ISSUE-AN-OPEN.
005300     PERFORM ISSUE-A-READ.
005400     PERFORM ISSUE-A-WRITE.
005500     GO TO CREJECT-TEST.
005600     GO TO PCLOSE-CALL.
005700 ISSUE-AN-OPEN.
005800     DISPLAY M-DOLLAR.
005900     DISPLAY O-DOLLAR.
006000     PERFORM LOOPI VARYING I FROM 1 BY 1 UNTIL
006100      I IS GREATER THAN 20.
006200     DISPLAY " TAG TO BE SENT: ".
006300     DISPLAY ITAG-ARRAY.
006400     CALL "CPOPEN" USING CCODE, DSNUM, D1-DOLLAR, P0-DOLLAR,
006500      ITAG-ARRAY, E0-DOLLAR, PARAM, FLAGS, STACKSIZE, DLSIZE,
006600      MAXDATA, BUFSIZE.
006700     PERFORM STATUS-CK-RTN.
006800     DISPLAY O-DOLLAR.
006900     DISPLAY " CCODE=", CCODE, " DSNUM=", DSNUM,
007000      " PARAM=", PARAM.
007100     DISPLAY " FLAGS=", FLAGS, " STACKSIZE=", STACKSIZE.
007200     DISPLAY " DLSIZE=", DLSIZE, " MAXDATA=", MAXDATA.
007300     DISPLAY " BUFSIZE=", BUFSIZE, " LGTH=", LGTH.
007400     DISPLAY " PROGNAME=", P0-DOLLAR.
007500     DISPLAY " ITAG-ARRAY RECEIVED: ".
007600     DISPLAY ITAG-ARRAY.
007700 ISSUE-A-PREAD.
007800     DISPLAY R-DOLLAR.
007900     PERFORM LOOPI VARYING I FROM 1 BY 1 UNTIL
008000      I IS GREATER THAN 20.
008100     DISPLAY "ITAG TO BE SENT: ".
008200     DISPLAY ITAG-ARRAY.
008300     CALL "CPREAD" USING CCODE, DSNUM, LGTH, DATA-BUF,
008400      TCOUNT, ITAG-ARRAY.
008500     PERFORM STATUS-CK-RTN.
008600     DISPLAY " CCODE=", CCODE, " DSNUM=", DSNUM,
008700      " LGTH=", LGTH.
008800     DISPLAY " DATA RECEIVED FROM SLAVE: ".
008900     DISPLAY DATA-BUF.
009000     DISPLAY " ITAG RECEIVED: ".
009100     DISPLAY ITAG-ARRAY.
009200 ISSUE-A-WRITE.
009300     DISPLAY W-DOLLAR.
```

```
009400     PERFORM MULTIPLY-LOOP VARYING I FROM 1 BY 1 UNTIL
009500      I IS GREATER THAN 20.
009600     DISPLAY " ITAG TO BE SENT: ".
009700     DISPLAY ITAG-ARRAY.
009800     MOVE "THIS IS THE DATA FROM PWRITE TEST." TO DATA-BUF.
009900     CALL "CPWRITE" USING CCODE, DSNUM, DATA-BUF, TCOUNT,
010000      ITAG-ARRAY.
010100     PERFORM STATUS-CK-RTN.
010200     DISPLAY " CCODE=", CCODE, " DSNUM=", DSNUM.
010300     DISPLAY " ITAG RECEIVED: ".
010400     DISPLAY ITAG-ARRAY.
010500 CREJECT-TEST.
010600     DISPLAY C0-DOLLAR.
010700     MOVE SPACES TO ITAG-ARRAY.
010800     MOVE 14 TO ITAG-ARRAY-MEM(1).
010900     DISPLAY "ITAG TO BE SENT: ".
011000     DISPLAY ITAG-ARRAY.
011100     CALL "CPCONTROL" USING CCODE, DSNUM, ITAG-ARRAY.
011200     PERFORM STATUS-CK-RTN.
011300     STOP RUN.
011400 LOOPI.
011500     MOVE I TO ITAG-ARRAY-MEM(I).
011600 MULTIPLY-LOOP.
011700     MULTIPLY 2 BY ITAG-ARRAY-MEM(I).
011800 STATUS-CK-RTN.
011900     IF CCODE IS LESS THAN ZERO GO TO
012000      SOMETHING-WENT-WRONG.
012100     IF CCODE IS GREATER THAN ZERO GO TO
012200      REQUEST-REJECTED.
012300     DISPLAY S-DOLLAR, "EVERYTHING OKAY".
012400 REQUEST-REJECTED.
012500     DISPLAY S-DOLLAR, "REQUEST REJECTED BY SLAVE".
012600     GO TO PCLOSE-CALL.
012700 SOMETHING-WENT-WRONG.
012800     DISPLAY S-DOLLAR, "CCL--SOMETHING IS WRONG".
012900     CALL "CPCHECK" USING CCODE, DSNUM, ICODE.
013000     DISPLAY K-DOLLAR, " CCODE=", CCODE,
013100      " ICODE=", ICODE.
013200 PCLOSE-CALL.
013300     DISPLAY C9-DOLLAR.
013400     CALL "CPCLOSE" USING CCODE, DSNUM.
013500     DISPLAY " CCODE=", CCODE, " DSNUM=", DSNUM.
013600     STOP RUN.
```

Slave PTOP Program

```
001000$CONTROL USLINIT,SOURCE
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. SLAVE-COBOL.
001300 ENVIRONMENT DIVISION.
001400 DATA DIVISION.
001500*********************************************
001600 WORKING-STORAGE SECTION.
001700 77 T        PIC S99  USAGE COMP.
001800 77 I        PIC S99  COMP VALUE 0.
001900 77 CCODE      PIC S99     COMP VALUE 0.
002000 77 IFUN       PIC S9      COMP VALUE 0.
002100 77 IL         PIC S99     COMP VALUE 0.
002200 77 IONUMBER   PIC S99     COMP VALUE 0.
002300 77 ICODE      PIC S99     COMP VALUE 0.
002400 77 DSNUM      PIC S99     COMP VALUE 0.
002500*********************************************
002600 77 C-DOLLAR  PIC X(11)  VALUE "## CHECK ##".
002700 77 G-DOLLAR  PIC X(9)   VALUE "## GET ##".
002800 77 A-DOLLAR  PIC X(12)  VALUE "## ACCEPT ##".
002900 77 R-DOLLAR  PIC X(12)  VALUE "## REJECT ##".
003000 77 S-DOLLAR  PIC X(25)  VALUE "######### SLAVE #########".
003100*********************************************
003200 77 OO-DOLLAR  PIC X(11)  VALUE "<< POPEN >>".
003300 77 CO-DOLLAR  PIC X(14)  VALUE "<< PCONTROL >>".
003400 77 SO-DOLLAR  PIC X(18)  VALUE "## STATUS CHECK ##".
003500 77 RO-DOLLAR  PIC X(11)  VALUE "<< PREAD >>".
003600 77 WO-DOLLAR  PIC X(12)  VALUE "<< PWRITE >>".
003700*********************************************
003800 01 DATA-ARRAY.
003900     02 DATA-ARRAY-MEM  PIC 99 OCCURS 33 TIMES.
004000 01 ITAG-ARRAY.
004100     02 ITAG-ARRAY-MEM  PIC 99 OCCURS 20 TIMES.
004200*****************************************************
004300*            PROCEDURE DIVISION                     *
004400*****************************************************
004500 PROCEDURE DIVISION.
004600 START-OF-SLAVE.
004700     DISPLAY S-DOLLAR.
004800     CALL "CGET" USING CCODE, IFUN, ITAG-ARRAY, IL, IONUMBER.
004900     PERFORM PRINT-GET-PARAMS THROUGH CHECK-RETURN-CC.
005000     IF CCODE IS NOT EQUAL TO ZERO GO TO CREJECT-PAR.
005100     GO TO POPEN, PREAD, PWRITE, PCONTROL DEPENDING ON IFUN.
005200 POPEN.
005300     DISPLAY S-DOLLAR.
005400     DISPLAY OO-DOLLAR.
005500     MOVE ZEROES TO ITAG-ARRAY.
005600     GO TO CACCEPT-PAR.
005700 PREAD.
005800     DISPLAY S-DOLLAR.
005900     DISPLAY RO-DOLLAR.
006000     PERFORM LOOP1 VARYING I FROM 1 BY 1 UNTIL
006100      I IS GREATER THAN 20.
```

```
006200      PERFORM INCREASE-LOOP VARYING T FROM 1 BY 1 UNTIL
006300       T IS GREATER THAN IL.
006400      GO TO CACCEPT-PAR.
006500 LOOP7.
006600      MOVE 7 TO ITAG-ARRAY-MEM(I).
006700 LOOP1.
006800      MOVE 1 TO ITAG-ARRAY-MEM(I).
006900 LOOP2.
007000      MOVE 2 TO ITAG-ARRAY-MEM(I).
007100 INCREASE-LOOP.
007200      MOVE T TO DATA-ARRAY-MEM(T).
007300 PWRITE.
007400      DISPLAY S-DOLLAR.
007500      DISPLAY WO-DOLLAR.
007600      PERFORM LOOP7 VARYING I FROM 1 BY 1 UNTIL
007700       I IS GREATER THAN 20.
007800 CACCEPT-PAR.
007900      CALL "CACCEPT" USING CCODE, ITAG-ARRAY, DATA-ARRAY, IL.
008100      PERFORM CHECK-RETURN-CC.
008200      IF IFUN = 3 PERFORM PRINT-DATA.
008300      GO TO START-OF-SLAVE.
008400 CREJECT-PAR.
008500      CALL "CREJECT" USING CCODE, ITAG-ARRAY.
008600      PERFORM CHECK-RETURN-CC.
008700      GO TO START-OF-SLAVE.
008800 PCONTROL.
008900      DISPLAY CO-DOLLAR.
009000      IF ITAG-ARRAY-MEM(1) = 14 GO TO CREJECT-PAR.
009100      PERFORM LOOP2 VARYING I FROM 1 BY 1 UNTIL
009200       I IS GREATER THAN 20.
009300      GO TO CACCEPT-PAR.
009400 PRINT-DATA.
009500      DISPLAY "DATA RECEIVED FROM THE MASTER: ".
009600      DISPLAY DATA-ARRAY.
009700 PRINT-GET-PARAMS.
009800      DISPLAY " CCODE=", CCODE, " IFUN=", IFUN, " IL=", IL,
009900       " IONUMBER=", IONUMBER.
010000      DISPLAY "ITAG RECEIVED: ".
010100      DISPLAY ITAG-ARRAY.
010200 CHECK-RETURN-CC.
010300      IF CCODE IS NOT EQUAL TO ZERO
010400       PERFORM SOMETHING-WENT-WRONG.
010500      DISPLAY SO-DOLLAR, "EVERYTHING OKAY".
010600 SOMETHING-WENT-WRONG.
010700      CALL "CPCHECK" USING CCODE, DSNUM, ICODE.
010800      DISPLAY " CCODE=", CCODE, " ICODE=", ICODE.
010900 ERROR-EXIT.
011000      DISPLAY "## SLAVE PROGRAM EXITING ##".
011100      STOP RUN.
```

DS/3000 BASIC INTERFACE

APPENDIX C

CONVENTIONS

When parameters are specified in the CALL statement the BASIC/ 3000 Interpreter (and compiled BASIC) sets up a parameter address table. The parameter address table consists of:

- The number of parameters.
- A code word for each parameter, specifying data type and dimensioning.
- A reference pointer to each parameter.

See Appendix F of the *BASIC/3000 Interpreter Manual.*

Because the DS/3000 intrinsics are program-to-program, the BASIC/3000 slave must be a compiled and PREPed program. The master program may be either running on the Interpreter or run as a compiled program.

It is assumed that the user is already familiar with DS/3000 in general and the program-to-program intrinsics specifically. Information regarding formal usage or content of the interface intrinsic parameters can be found in Section 6.

COMMON PARAMETERS

Parameters whose usage is the same throughout the procedures are:

ccode — integer.
Condition code returned by the DS/3000 Program-to-Program intrinsic.

-3 = not enough user stack for data transfer.
-2 = CCL
0 = CCE
1 = CCG

dsnum — integer.
The DS/3000 communication line number. (analagous to FOPEN file number)

itag — integer.
A 20-word array.

Parameters — The BASIC-DS/3000 interface routines pack and unpack the data specified in the parameter lists of the master and the slave programs. The user must insure that the number of parameters specified on the master and the slave sides are the same and that the data types correspond. If the sending and receiving data types are not the same, the resulting data will be unpredictable.

Special care must be taken when passing string variables (simple strings or string arrays). String variables have both a physical (i.e. maximum) length and a logical (i.e. current) length. The logical length is never greater than the physical length and may, in fact, be smaller. Since PTOP packs string variable parameters according to their logical lengths, but unpacks according to the physical length of the receiving string variables, the same string variable cannot be used for both sending and receiving if its physical length exceeds its logical length.

For example, suppose N0$ is a 10 character string which currently has the value JOHN. N's physical length is 10 but its logical length is only 4. If a master PTOP program were to BPWRITE N0$, only 4 characters of data will be sent across the line. If the slave PTOP program were to call BACCEPT with another 10 character string, say M0$, the value of M0$ after the BACCEPT would be JOHNxxxxxx, where the xs are undefined. The logical length of M0$ would now be equal to its physical length, which is 10.

As another example, suppose the master PTOP program were to pass two 10 character strings to BPWRITE (N0$ and N1$), and the slave PTOP program called BACCEPT with two 10 character strings (M0$ and M1$). If N0$ = JOHN and N1$ = JANE DOE, the resulting values of M0$ and M1$ would be JOHN JANE and DOExxxxxxx, respectively. (The xs are undefined characters.)

These anomalies are consistent in BASIC with the rules governing access to other media in which the length of strings is not inherent (for example, binary files). There are several ways to avoid these problems. The first method, which is illustrated in the example BASIC PTOP programs in this section, is to pad all strings with blanks before transmitting them (i.e. before BPWRITE or before BPACCEPT of a BPREAD). Another, more sophisticated approach, would be for the sending side to pass the logical lengths of the strings in the *itag* array. The receiving side would supply only one parameter, a string which is large enough to hold all of the strings which were sent. This string could then be unpacked by the user program using the information in the tag array and the substring functions of BASIC.

There is one more anomaly involved in passing strings. Since strings are always unpacked starting on word boundaries, it is not possible to unpack an even length string into two odd length strings.

INTERFACE INTRINSICS

BPOPEN

This procedure is the BASIC callable interface to POPEN.

Calling Sequence:

```
CALL BPOPEN(ccode,dsnum,dsdevice,progname,{itag}{0},{entryname}{0},{param}{0},{flags}{0},
            {stacksize}{0},{dlsize}{0},{maxdata}{0},
            buffsize)
```

Where:

dsdevice — string.
The DS line class, node name, or logical device number (string must have at least one trailing blank).

progname — string.
Name of remote slave program (terminated with a blank).

entryname — string.
Secondary entry point into the slave program (terminated with a blank).

param — integer.
Value placed in Q-4 of the slave program stack.

flags, *stacksize*, *dlsize*, *maxdata* — MPE parameters used to control slave program loading. See Section 6 of this manual or the *MPE Intrinsics Reference Manual* for usage.

bufsize — integer.
Maximum number of words per PTOP transfer.

BPREAD

This procedure is the BASIC interface routine to PREAD.

Calling Sequence:

CALL BPREAD(*ccode*,*dsnum*,*lgth*,$\left\{\begin{matrix} itag \\ 0 \end{matrix}\right\}$,*param list*)

Where:

lgth integer.
Number of words received in transfer.

param list ::= *param* [,*param list*] where *param* is any BASIC supported data type (such as STRING, INTEGER, REAL ARRAY)

NOTE

The BASIC interface routines: BPREAD, BPWRITE, and BACCEPT differ significantly from the PTOP intrinsics in the *target/tcount* vs. parameter list data schemes. The parameter list allows the BASIC user to send and receive heterogeneous data items. A contiguous buffer is built on the User's stack for the transfers.

BPWRITE

This procedure is the BASIC callable interface to PWRITE.

Calling sequence:

CALL BPWRITE(*ccode*,*dsnum*,$\left\{\begin{matrix} itag \\ 0 \end{matrix}\right\}$,*param list*)

BPCONTROL

This procedure is the BASIC interface routine to PCONTROL.

Calling Sequence:

CALL BPCONTROL(*ccode*,*dsnum*,*itag*)

BPCLOSE

This procedure is the BASIC interface routine to PCLOSE.

Calling Sequence:

CALL BPCLOSE(*ccode*,*dsnum*)

BGET

This procedure is the BASIC interface routine to GET.

Calling Sequence:

CALL BGET(*ccode*,*ifun*,$\left\{\begin{matrix} itag \\ 0 \end{matrix}\right\}$,$\left\{\begin{matrix} il \\ 0 \end{matrix}\right\}$,$\left\{\begin{matrix} ionumber \\ 0 \end{matrix}\right\}$)

Where:

ifun — integer.
Receives the function code from the request issued by the remote master program. (Refer to Section 6 for *ifun* meanings.)

il — integer.
The number of words expected or sent on BPREAD or BPWRITE.

ionumber — integer.
Valid if both *ccode*=1 and *ifun*=5. File number of completed non-DS I/O without wait.

BACCEPT

This is the BASIC callable interface routine to ACCEPT.

Calling Sequence:

CALL BACCEPT(*ccode*,*ifun*,$\left\{\begin{matrix} itag \\ 0 \end{matrix}\right\}$,*param list*)

BREJECT

This is the BASIC callable interface routine to REJECT.

Calling Sequence:

```
CALL BREJECT(ccode,itag)
```

BPCHECK

This is the BASIC callable routine to PCHECK.

Calling Sequence:

```
CALL BPCHECK(ccode,dsnum,icode)
```

Where:

dsnum integer.
For a master program, this number is returned by BPOPEN. For a slave program, this number is always 0.

icode integer.
The number returned identifies the last error encountered. Refer to Appendix B, under the heading "DS/3000 Functional Errors", for meaning.

EXAMPLES

Master PTOP Program

```
MASTER
    1 REM:*****************************************************
    2 REM:
    3 REM:                 MASTER PTOP PROGRAM
    4 REM:
    5 REM:*****************************************************
    6 REM:
    7 REM:  THIS PROGRAM ISSUES A BPOPEN TO THE SLAVE PROGRAM AND
    8 REM:  USES THE TAG FIELD TO SEND SUBTYPES FOR THE BREAD/BWRITE
    9 REM:  OPERATIONS.
   10 REM:
   11 REM:*****************************************************
  110 REM:                   DATA DECLARATIONS
  120 REM:*****************************************************
  130 REM:   BASIC PTOP INTRINSIC PARAMETERS--
  140 INTEGER C,D,F,IO,L
  150 REM: C=CC, D=DSNUM, F=FLAGS, IO=ICODE, L=LENGTH
  160 INTEGER I1[20]
  165 MAT I1=ZER
```

```
170 REM: I1[*]=TAG FIELD
180 DIM DO$[10],PO$[10]
190 REM: DO$=DSLINE, PO$=REMOTE PROGRAM
200 REM: ------LOCAL VARIABLES-----
210 DIM N1$[20],N2$[20],A1$[20],R$[40]
220 REM: N1$,N2$=NAMES; A1$[20]=ADDRESS; R$=RECORD(NAME,ADDRESS)
221 DIM B$[20]
222 REM: B$= BLANK PADDING STRING
230 DIM C$[20],N$[22]
240 REM: C$=USER COMMAND/TEXT LINE
250 REM: N$=ENTER NAME MESSAGE
260 REM:*******************************************************
270 REM:
280 REM:                START OF PROCESSING
290 REM:
300 REM:*******************************************************
310 REM: -----INITIALIZATION OF VALUES AND BPOPEN-----
320 PRINT "ENTER DSLINE CLASS NAME"
325 INPUT DO$
330 PRINT "ENTER SLAVE PROGRAM NAME"
335 INPUT PO$
340 N$="ENTER NAME "
345 B$="                    "
360 CALL BPOPEN(C,D,DO$,PO$)
370 IF C=0 THEN GOTO 410
380 PRINT "### ERROR ON BPOPEN ###"
390 GOSUB 7000
400 STOP
410 REM:*******************************************************
420 REM:            WELCOME MESSAGE AND MENU
430 REM:*******************************************************
440 PRINT "MASTER AND SLAVE PTOP RUNNING"
450 PRINT " "
460 PRINT "*** OPERATIONS MENU ***"
470 PRINT "    N  -  NAME CHANGE"
480 PRINT "    A  -  ADDRESS CHANGE"
490 PRINT "    I  -  INSERT PERSON"
500 PRINT "    D  -  DELETE PERSON"
510 PRINT "   LN  -  LIST NAME AND ADDRESS"
520 PRINT "   LA  -  LIST ALL NAMES AND ADDRESSES"
530 PRINT "   EX  -  EXIT PROGRAM"
540 REM:*******************************************************
550 REM:                OPERATION REQUEST
560 REM:*******************************************************
570 PRINT " "
580 PRINT "ENTER OPERATION"
590 LINPUT C$
600 IF C$[1,1]="N" THEN GOTO 1000
610 IF C$[1,1]="A" THEN GOTO 2000
620 IF C$[1,1]="I" THEN GOTO 3000
630 IF C$[1,1]="D" THEN GOTO 4000
640 IF C$[1,2]="LN" THEN GOTO 5000
650 IF C$[1,2]="LA" THEN GOTO 6000
660 IF C$[1,2]="EX" THEN GOTO 8000
```

```
 670 PRINT "*** UNRECOGNIZED OPERATION ***"
 680 GOTO 450
1000 REM:*****************************************************
1010 REM:                    NAME CHANGE
1020 REM:*****************************************************
1030 PRINT N$
1040 LINPUT N1$
1045 IF N1$="" THEN GOTO 450
1050 IF I1[1]<0 OR I1[1]>1 THEN GOSUB 7000
1060 PRINT "ENTER NEW NAME"
1070 LINPUT N2$
1080 I1[1]=1
1084 REM: PAD STRINGS WITH BLANKS BEFORE XMITTING
1085 N1$=N1$+B$
1086 N2$=N2$+B$
1090 CALL BPWRITE(C,D,I1[*],N1$,N2$)
1100 GOSUB 7000
1110 GOTO 1000
2000 REM:*****************************************************
2010 REM:                  ADDRESS CHANGE
2020 REM:*****************************************************
2030 PRINT N$
2040 LINPUT N1$
2050 IF N1$="" THEN GOTO 450
2060 PRINT "ENTER NEW ADDRESS"
2070 LINPUT A1$
2080 I1[1]=2
2084 REM: PAD STRINGS WITH BLANKS
2085 N1$=N1$+B$
2086 A1$=A1$+B$
2090 CALL BPWRITE(C,D,I1[*],N1$,A1$)
2100 GOSUB 7000
2200 GOTO 2000
3000 REM:*****************************************************
3010 REM:                   INSERT NAME
3020 REM:*****************************************************
3030 PRINT N$
3040 LINPUT N1$
3050 IF N1$="" THEN GOTO 450
3060 PRINT "ENTER ADDRESS"
3070 LINPUT A1$
3080 I1[1]=3
3084 REM: PAD STRINGS WITH BLANKS
3085 N1$=N1$+B$
3086 A1$=A1$+B$
3090 CALL BPWRITE(C,D,I1[*],N1$,A1$)
3100 GOSUB 7000
3110 GOTO 3000
4000 REM:*****************************************************
4010 REM:                    DELETE PERSON
4020 REM:*****************************************************
4030 PRINT N$
4040 LINPUT N1$
4050 IF N1$="" THEN GOTO 450
```

```
4060 I1[1]=4
4065 REM: PAD NAME WITH BLANKS
4066 N1$=N1$+B$
4070 CALL BPWRITE(C,D,I1[*],N1$)
4080 GOSUB 7000
4090 GOTO 4000
5000 REM:*********************************************************
5010 REM:                LIST NAME AND ADDRESS
5020 REM:*********************************************************
5030 PRINT N$
5040 LINPUT N1$
5050 IF N1$="" THEN GOTO 450
5060 I1[1]=1
5067 REM: PAD NAME WITH BLANKS
5068 N1$=N1$+B$
5070 CALL BPWRITE(C,D,I1[*],N1$,N1$)
5080 IF C=0 THEN CALL BPCONTROL(C,D,I1[*])
5090 IF C<>0 OR I1[1]<>1 THEN GOTO 5000
5100 L=-80
5110 CALL BPREAD(C,D,L,I1[*],R$)
5120 GOSUB 7000
5130 PRINT R$
5140 GOTO 5000
6000 REM:*********************************************************
6010 REM:                 LIST WHOLE LIST
6020 REM:*********************************************************
6030 L=-80
6040 I1[1]=2
6050 CALL BPREAD(C,D,L,I1[*],R$)
6060 GOSUB 7000
6070 PRINT R$
6080 IF I1[1]=0 THEN GOTO 6000
6090 GOTO 450
7000 REM:*********************************************************
7010 REM:          CONDITION CODE AND STATUS CHECK
7020 REM:*********************************************************
7030 IF C>0 THEN GOTO 7120
7040 IF C<0 THEN GOTO 7150
7050 CALL BPCONTROL(C,D,I1[*])
7060 IF I1[1]<0 OR I1[1]>1 THEN GOTO 7090
7065 REM: REMINDER--CHECK ABOVE LINE
7070 REM: *** EVERYTHING OKAY ***
7080 RETURN
7090 REM: *** BAD RECORD ***
7100 PRINT "### NON-EXISTENT RECORD ###"
7110 RETURN
7120 REM: *** CCG ***
7130 PRINT "### REQUEST REJECTED BY SLAVE ###"
7140 RETURN
7150 REM: *** CCL ***
7160 CALL BPCHECK(C,D,I0)
7170 PRINT "### PTOP ERROR:";I0;" ###"
7180 RETURN
8000 REM:*********************************************************
```

```
8010 REM:                          EXIT
8020 REM:*******************************************************
8030 CALL BPCLOSE(C,D)
8040 GOSUB 7000
8050 END
```

Slave PTOP Program

```
SLAVEBF
    1 REM:*****************************************************
    2 REM:
    3 REM:                SLAVE PTOP PROGRAM
    4 REM:
    5 REM:*****************************************************
    6 REM:
    7 REM:  THIS PROGRAM ACCEPTS DATA FROM THE MASTER AND
    8 REM:  ACCORDINGLY CHANGES, INSERTS, OR DELETES ENTRIES.
    9 REM:  IT ALSO TRANSMITS NAME/ADDRESS RECORDS TO THE MASTER
   10 REM:
  100 REM:*****************************************************
  110 REM:                DATA DECLARATIONS
  120 REM:*****************************************************
  130 REM: -----BASIC PTOP INTRINSIC PARAMETERS-----
  140 INTEGER C,F,L,IO,I,D
  150 REM: C=CC; F=FUNCTION; L=IL; IO=I/O; I=ICODE; D=DSNUM
  155 D=0
  160 INTEGER I1[20]
  170 REM: I1[*]=TAG FIELD
  180 REM: -----LOCAL VARIABLES-----
  190 DIM N1$[20],N2$[20],A1$[20]
  200 REM: N1$,N2$=NAME; A1$=ADDRESS
  210 DIM NO$[20,20],AO$[20,20]
  220 REM: NO$,AO$=LIST OF NAMES AND ADDRESSES
  225 DIM B$[20]
  226 REM: B$= BLANK PADDING STRING
  230 INTEGER P,S
  240 REM: P=LAST RECORD POINTER; S=STATUS
  250 INTEGER X
  255 X=0
  256 S=1
  800 REM:*****************************************************
  810 REM:                START OF PROGRAM
  820 REM:*****************************************************
  830 CALL BGET(C,F,I1[*])
  845 IF C<>0 THEN GOTO 7000
  850 GOSUB 5000
  870 GOSUB F OF 1000,2000,3000,4000
  880 GOTO 800
 1000 REM:*****************************************************
 1010 REM:                      BPOPEN
 1020 REM:*****************************************************
 1030 CALL BACCEPT(C,F)
 1050 GOSUB 5000
 1055 REM: INITIALIZE NAME AND ADDRESS ARRAYS
 1056 B$="                    "
 1060 FOR P=1 TO 9
 1070   READ NO$[P],AO$[P]
 1074   REM: PAD STRINGS WITH BLANKS
 1075   NO$[P]=NO$[P]+B$
 1076   AO$[P]=AO$[P]+B$
```

```
1080 NEXT P
1085 P=9
1090 RETURN
2000 REM:*****************************************************
2010 REM:                          READ
2020 REM:*****************************************************
2030 IF I1[1]=2 THEN GOTO 2090
2040 REM: *** LIST SINGLE RECORD ***
2050 I1[1]=S=1
2060 CALL BACCEPT(C,F,I1[*],N0$[P0],A0$[P0])
2070 GOSUB 5000
2080 RETURN
2090 REM: ***LIST ALL RECORDS ***
2100 IF S<>0 THEN P0=0
2110 S=0
2120 P0=P0+1
2130 IF P0=P THEN S=1
2140 I1[1]=S
2150 CALL BACCEPT(C,F,I1[*],N0$[P0],A0$[P0])
2160 GOSUB 5000
2170 RETURN
3000 REM:*****************************************************
3010 REM:                         WRITE
3020 REM:*****************************************************
3030 GOSUB I1[1] OF 3040,3160,3290,3410
3035 RETURN
3040 REM: ***ENTRY: NAME CHANGE ***
3050 CALL BACCEPT(C,F,I1[*],N1$,N2$)
3055 GOSUB 5000
3060 P0=0
3070 P0=P0+1
3080 IF N0$[P0]=N1$ THEN GOTO 3120
3090 IF P0<>P THEN GOTO 3070
3100 S=-1
3110 RETURN
3120 REM: *** NAME FOUND, CHANGE IT ***
3130 N0$[P0]=N2$
3140 S=1
3150 RETURN
3160 REM: ***ENTRY: ADDRESS CHANGE ***
3170 CALL BACCEPT(C,F,0,N1$,A1$)
3180 GOSUB 5000
3190 P0=0
3200 P0=P0+1
3210 IF N0$[P0]=N1$ THEN GOTO 3250
3220 IF P0<>P THEN GOTO 3200
3230 S=-1
3240 RETURN
3250 REM: *** RECORD FOUND, CHANGE IT ***
3260 A0$[P0]=A1$
3270 S=1
3280 RETURN
3290 REM: ***ENTRY: INSERT NAME
3300 CALL BACCEPT(C,F,0,N1$,A1$)
```

```
3310 GOSUB 5000
3320 IF P=20 THEN GOTO 3380
3330 P=P+1
3340 N0$[P]=N1$
3350 A0$[P]=A1$
3360 S=1
3370 RETURN
3380 REM: ***LIST ALREADY FULL ***
3390 S=-1
3400 RETURN
3410 REM: *** ENTRY: DELETE NAME ***
3420 CALL BACCEPT(C,F,0,N1$)
3430 GOSUB 5000
3440 P0=0
3450 P0=P0+1
3460 IF N0$[P0]=N1$ THEN GOTO 3500
3470 IF P0<>P THEN GOTO 3450
3480 S=-1
3490 RETURN
3500 REM: ***FOUND RECORD, DELETE IT ***
3510 N0$[P0]=B$
3515 A0$[P0]=B$
3520 S=1
3530 RETURN
4000 REM:*******************************************************
4010 REM:                      BPCONTROL
4020 REM:*******************************************************
4030 I1[1]=S
4040 CALL BACCEPT(C,F,I1[*])
4050 RETURN
5000 REM:*******************************************************
5010 REM:       CONDITION CODE AND STATUS CHECK
5020 REM:*******************************************************
5030 IF C>0 THEN GOTO 5070
5040 IF C<0 THEN GOTO 5100
5050 REM: ***EVERYTHING OKAY ***
5060 RETURN
5070 REM: *** CCG ERROR ***
5080 PRINT "### SLAVE: PTOP(CCG) ###"
5090 RETURN
5100 REM: *** CCL ERROR ***
5110 I=0
5120 CALL BPCHECK(C,I,I0)
5130 PRINT "### SLAVE: PTOP ERROR:";I0;" ###"
5140 RETURN
6000 DATA "CHRISTINE","BRISTOL"
6010 DATA "MEL","CAMBRIDGE"
6020 DATA "CAROL","PALO ALTO"
6030 DATA "LISA, MISA & RICHIE","BRISTOL"
6040 DATA "LISBET","ZURICH"
6050 DATA "JOHN","BERKELEY"
6060 DATA "CAROLYN","ST. PAUL"
6070 DATA "TODD","SANTA CLARA"
6080 DATA "GARY","SAN FRANCISCO"
```

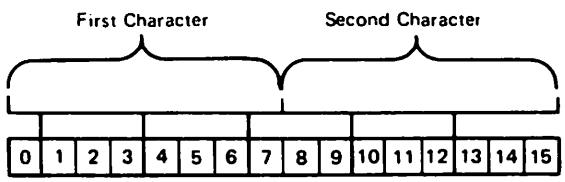
```
7000 CALL REJECT(C,I1[*])
7010 GOTO 830
```

ASCII CHARACTER SET

APPENDIX D

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000110
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
a	060400	000141
b	061000	000142
c	061400	000143
d	062000	000144
e	062400	000145
f	063000	000146
g	063400	000147
h	064000	000150
i	064400	000151
j	065000	000152
k	065400	000153
l	066000	000154
m	066400	000155
n	067000	000156
o	067400	000157
p	070000	000160
q	070400	000161
r	071000	000162
s	071400	000163
t	072000	000164
u	072400	000165
v	073000	000166
w	073400	000167
x	074000	000170
y	074400	000171
z	075000	000172
0	030000	000060
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
NUL	000000	000000
SOH	000400	000001
STX	001000	000002
ETX	001400	000003
EOT	002000	000004
ENQ	002400	000005

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
ACK	003000	000006
BEL	003400	000007
BS	004000	000010
HT	004400	000011
LF	005000	000012
VT	005400	000013
FF	006000	000014
CR	006400	000015
SO	007000	000016
SI	007400	000017
DLE	010000	000020
DC1	010400	000021
DC2	011000	000022
DC3	011400	000023
DC4	012000	000024
NAK	012400	000025
SYN	013000	000026
ETB	013400	000027
CAN	014000	000030
EM	014400	000031
SUB	015000	000032
ESC	015400	000033
FS	016000	000034
GS	016400	000035
RS	017000	000036
US	017400	000037
SPACE	020000	000040
!	020400	000041
"	021000	000042
#	021400	000043
$	022000	000044
%	022400	000045
&	023000	000046
'	023400	000047
(	024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
,	026000	000054
-	026400	000055
.	027000	000056
/	027400	000057
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
[	055400	000133
\	056000	000134
]	056400	000135
Δ	057000	000136
_	057400	000137
`	060000	000140
{	075400	000173
|	076000	000174
}	076400	000175
~	077000	000176
DEL	077400	000177



INDEX

A

B

C

D

E

F

G

H

I

I/O configuration, and compression, 8-10

K

L

M

N

O

P

Q

R

S

T

U

V

X

MANUAL UPDATE

MANUAL IDENTIFICATION
Title: HP 3000 to HP 3000 User/Programmer Reference Manual
Part Number: 32185-90001
Edition Date: DECEMBER 1985

UPDATE IDENTIFICATION
Update Number: 1
Update Date: JULY 1987

THE PURPOSE OF THIS MANUAL UPDATE
is to accumulate all the changes to the latest edition of the manual. Earlier updates to the latest edition which have not been incorporated are contained herein. This update package consists of all new and changed pages (backup pages are provided when necessary) plus this cover letter.

CHANGED PAGES
have the date of the update at the bottom of the page. Changes are marked with a vertical bar in the margin; when an update is incorporated in a subsequent reprinting of the manual, these bars are removed. "New" pages are those which were not present in the latest edition of the manual.

TO UPDATE YOUR MANUAL
replace existing pages in the latest edition of the manual with corresponding pages from this update package. Destroy all replaced pages. In addition, insert any new pages from this update.

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
19420 HOMESTEAD ROAD, CUPERTINO, CA 95014

32185-90001
U0787

Part No. 32185-90001
Printed in U.S.A. 12/85
E1285

HEWLETT PACKARD