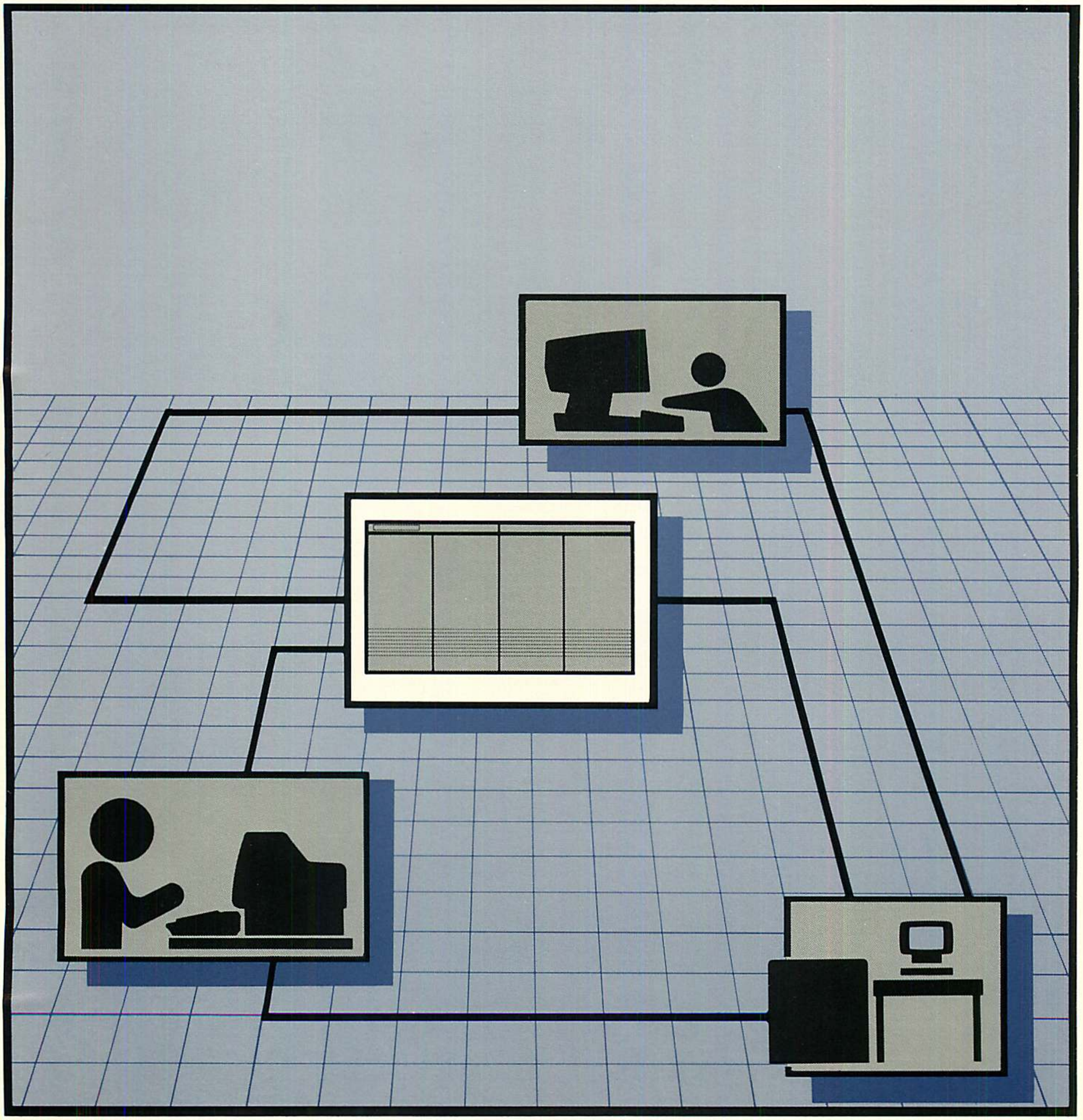
HP AdvanceNet

HEWLETT PACKARD

NetIPC3000/V

Programmer's Reference Manual



HP AdvanceNet

NetIPC3000/V

Programmer's Reference Manual

HEWLETT PACKARD

19420 HOMESTEAD ROAD CUPERTINO, CA 95014

Part No. 5958-8581
E0788

Printed in U.S.A. 07/88

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 . . . . MAY 1987. . . . . 32344A.00.04
Update 1. DEC 1987. . . . . 32344A.00.06
Edition 2. JUL 1988. . . . . 32344A.00.01

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
All.	JUL 1988

PREFACE

Network Interprocess Communication (NetIPC) is a set of programmatic calls that can be used to exchange data between processes executing on the same or different nodes in an HP NS network. NetIPC3000/V, in particular, is a version of NetIPC that can be used in programs written for MPE-V based systems.

NetIPC provides programmatic access to network protocols. NetIPC3000/V currently provides access to the Transmission Control Protocol (TCP), the Transport Layer protocol used by NS3000/V link products.

NetIPC3000/V is provided with the purchase of any NS3000/V link product. These products include:

- StarLAN/3000 Link (product number 30265A)
- ThinLAN/3000 (product number 30240A; includes ThickLAN option)
- NS Point-to-Point 3000/V Link (product number 30284A for MICRO 3000, MICRO 3000XE, Series 37, and Series 37XE computers; product number 30285A for other HP 3000s)
- Asynchronous SERIAL Network Link (product number 32003A)
- NS X.25 3000/V Link (product number 24405A)

NOTE

The material in this manual supercedes descriptions of NetIPC for the HP 3000 previously included in the *NS3000/V User/Programmer Reference Manual.*

Audience

As a NetIPC programmer, you should be familiar with MPE-V, the HP 3000 operating system on which NetIPC3000/V can be used. You should also be familiar with the TCP protocol. If you are using direct access to level 3 (X.25) you should be familiar with the X.25 protocol and the NS X.25 3000/V Link.

Organization of This Manual

This manual contains the following sections:

- Section 1, "Introduction," explains the method used by NetIPC to establish connections between processes, and introduces the NetIPC calls involved.
- Section 2, "NetIPC Intrinsics," provides a detailed description of each NetIPC intrinsic, in alphabetical order. This section also explains the structure and function of several parameters that are common to multiple NetIPC calls.
- Section 3, "NetIPC Examples," provides two sample programs that use NetIPC.
- Appendix A, "IPC Interpreter (IPCINT)", describes how to use the **IPCINT** software utility which provides an interactive interface to the NetIPC intrinsics used for programmatic access to X.25 level 3.
- Appendix B, "Cause and Diagnostic Codes lists the possible cause and diagnostic codes generated by NS X.25 packets.
- Appendix C, "Error Messages", includes a list of SOCKERRs and the corresponding protocol module errors returned in the IPCCHECK intrinsic, and a complete table of the SOCKERRs generated by NetIPC.

Related Publications

The following publications may be of additional use to you when writing programs with NetIPC:

MPE-V Programming:

PASCAL/3000 Reference Manual (32106-90001)

COBOL/3000 Reference Manual (32213-90001)

FORTRAN Reference Manual (30000-90040)

MPE V Intrinsics Reference Manual (32033-90007)

Refer to the above manuals for lists of additional language-related publications.

NS3000/V:

NS3000/V User/Programmer Reference Manual (32344-90001)

NS3000/V Network Manager Reference Manual, Volume I (32344-90002)

NS3000/V Network Manager Reference Manual, Volume II (32344-90012)

NS3000/V Error Message and Recovery Manual (32344-90005)

X.25 Protocol:

NS X.25 3000/V Link Guide (24405-90002)

X.25: The PSN Connection (5958-3402)

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.
, (shaded)	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*

CONVENTIONS (continued)

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.

NOTE

NetIPC intrinsics can be coded in either uppercase or lowercase characters. In this manual, intrinsics are sometimes shown in uppercase and sometimes shown in lowercase; however, this is not intended to indicate a requirement for using either uppercase or lowercase.

CONTENTS

Section 1
INTRODUCTION

CONTENTS (continued)

Section 2
NETIPC INTRINSICS

CONTENTS (continued)

FIGURES AND TABLES

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

SECTION

1

Network Interprocess Communication (NetIPC) is a facility that enables processes on the same or different nodes to communicate using a series of programmatic calls.

NetIPC3000/V can be purchased as part of any NS3000/V link product. It provides access to TCP (Transmission Control Protocol), the Transport Layer protocol used in NS3000/V link products. TCP corresponds to layer 4 (or level 4) of the OSI seven layer network model. Over an NS X.25 network, NetIPC provides access to X.25 protocol features at level 3 (of the OSI seven layer model). See the *NS3000/V Network Manager Reference Manual, Volume 1*, for more information about NS3000/V network architecture and the OSI model.

The form of process communication offered by NetIPC is more flexible than that provided by PTOP (Program-to-Program Communication) and more powerful than that provided by MPE V Message Files. (Refer to the *NS3000/V User/Programmer Reference Manual* for more information about these services.) Because the relationship between NetIPC processes is peer-to-peer rather than master-to-slave, NetIPC processes are more independent than PTOP processes where the "master" process is in control of communication.

NOTE

> To communicate by means of NetIPC, processes must be executing concurrently. One or more users (or programs) can run these processes independently, or one process can initiate the execution of another by using the Remote Process Management (RPM) Network Service. In conjunction with NetIPC, RPM can be used to manage distributed applications. Refer to the *NS3000/V User/Programmer Reference Manual* for information about RPM.

Processes that use NetIPC calls gain access to the communication services provided by the network protocols of NS3000/V. NetIPC does not encompass a protocol of its own, but acts as a generic interface to the protocols underlying all of the NS3000/V network services.

NETIPC FUNDAMENTALS

The following explanations are based on access to level 4 (TCP) but most of the principles also apply to direct access to level 3 (X.25). Information specific to X.25 is noted in the discussion.

Sockets

NetIPC processes communicate with each other by means of a data structure called a **socket**. Processes make use of sockets via the NetIPC calls to establish connections and exchange data with other processes. The Transport Layer's Transmission Control Protocol (TCP) regulates the transmission of data to and from these data structures. When direct access to level 3 (X.25) is used, the X.25 protocol regulates the transmission of data between sockets. Although data must pass through the control of lower-level protocols, these details are transparent to NetIPC processes when they send and receive data. For information about NS3000/V network architecture, refer to the *NS3000/V Network Manager Manual, Volume I.*)

Connections

Before a connection can be established between two NetIPC processes, each process must create a **call socket**. A call socket is roughly analogous to a telephone handset with multiple buttons or extensions. NetIPC processes engage in a dialogue, or "conversation," over the connections formed by their respective call sockets in order to create a **virtual circuit (VC) socket** at each process. A call socket can be thought of as one of the steps needed to build a VC socket.

The VC sockets created by this dialogue are the endpoints of a new connection called a **virtual circuit** or **virtual circuit connection**. While a call socket is analogous to a telephone with multiple extensions, a VC socket is analogous to one of the extensions on that telephone. Figure 1-1 is an illustration of this telephone analogy.

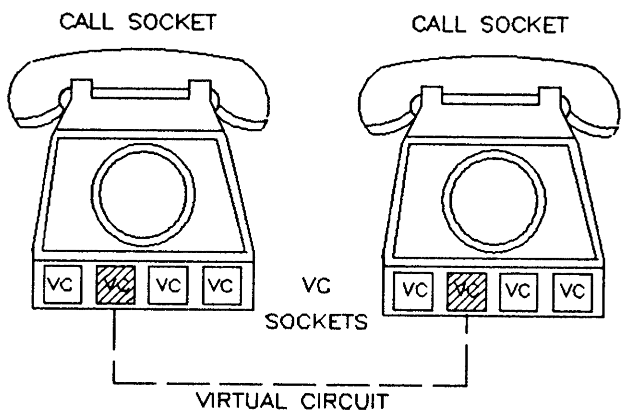


Figure 1-1. Telephone Analogy.

Virtual circuits are the basis for interprocess communications. Once a virtual circuit is established, the two processes that created it may use it to exchange data. *Two processes pass data only through VC sockets, not through call sockets.* For example, a process may use one call socket to establish multiple VC sockets; these VC sockets are then used to communicate with different processes. A call socket may even be shut down once a virtual circuit connection is established without affecting communication between the processes. A virtual circuit has two major properties:

- It is a dedicated link, accessible only to the two processes that established the connection.
- It provides reliable service, guaranteeing that data will not be corrupted, lost, duplicated or received out of order.

Naming, Socket Registry and Destinations

When a NetIPC process initiates a connection with a peer process, it must reference a call socket that was created by the peer process. In order to gain access to another process's call socket, a NetIPC process must reference the socket's **name**.

NetIPC processes associate ASCII-coded names with the call sockets they create and insert this information into their node's **socket registry**. Each NS3000/V node has a socket registry that contains a listing of all the named call sockets that reside at that node. Pursuing the telephone analogy begun earlier, the socket registry could be compared to a telephone directory: a call socket is associated with a name and inserted in the local socket registry in much the same way as a telephone number is associated with a person's name and placed in a local telephone directory.

NetIPC processes reference call sockets created by other processes by passing a socket name and the corresponding node name to the socket registry software. The socket registry determines which socket is associated with the name and translates the address of that socket into a **destination descriptor** which it returns to the inquiring process.

A destination descriptor is a data structure which carries address information. Specifically, when a destination descriptor is returned to a process, it tells the process:

- how to to get to the node where the referenced socket resides, and
- how to get to the referenced socket at that node.

Using the socket registry to gain access to another process's call socket is similar to using directory assistance to find a person's phone number. The end result is also similar: a destination descriptor, like a telephone number, is used to direct a caller to a particular destination.

Descriptors

NetIPC processes reference call sockets, VC sockets and destinations with **descriptors**. Descriptors are returned to processes when certain NetIPC calls are invoked. Below is an explanation of these descriptors, the NetIPC call, or calls, that are used to obtain them, and the terms which refer to them in syntax and parameters.

- **Call Socket Descriptor**. A call socket descriptor describes a call socket. A process obtains a call socket descriptor by invoking IPCCREATE (to create a call socket) or IPCGET (to get a call socket descriptor given away by another process). When a call socket descriptor is obtained with either one of these calls the call socket it describes is said to be *owned* by the calling process. The term *calldesc* refers to a call socket descriptor parameter.

- **Destination Descriptor**. A destination descriptor describes a destination socket. The descriptor points to addressing information that is used by the NS3000/V link product to direct requests to a certain call socket at a certain node. A process obtains a destination descriptor by invoking IPCDEST which creates a destination descriptor for the designated remote node, or by invoking IPCLOOKUP (to look up the name of a call socket in a specific socket registry). The term *destdesc* refers to a destination descriptor parameter.

- **VC Socket Descriptor**. A VC socket descriptor describes a VC socket. A VC socket is the endpoint of a virtual circuit connection between two processes. A VC socket descriptor is returned by IPCRECVCN and IPCCONNECT after an initial dialogue takes place over a connection formed by call sockets. A process can also obtain a VC socket descriptor given away by another process by invoking IPCGET. The term *vcdesc* refers to a VC socket descriptor parameter.

Table 1-1. Descriptor Summary.

TYPE OF DESCRIPTOR	PARAMETER NAME	DESCRIPTION	RETURNED AS OUTPUT FROM
call socket descriptor	*calldesc*	Refers to a call socket. A call socket is used to build a VC socket.	IPCCREATE IPCGET
destination descriptor	*destdesc*	Refers to a destination socket. A destination socket points to addressing information that is used to direct requests to a certain call socket at a certain node.	IPCLOOKUP IPCDEST
VC socket descriptor	*vcdesc*	Refers to a VC socket. A VC socket is the endpoint of a virtual circuit connection between two processes.	IPCCONNECT IPCRECVCN IPCGET

NOTE

The parameter *descriptor* is used in several NetIPC calls to refer to either a call socket descriptor, destination descriptor or a VC socket descriptor.

Establishing a Level 4 Connection

The steps needed to establish a virtual circuit connection are described in the following examples. Although only two processes are shown, this is not meant to imply that communication cannot exist between more than two processes. Either or both of the processes shown can establish virtual circuit connections with other processes. Secondary or auxiliary connections can also be set up between the same two processes.

NOTE

Both of the processes in the following dialogue are assumed to be created and running at their respective nodes. NetIPC does not include a call to schedule remote processes. Refer to "Remote Process Management" (RPM) in the *NS3000/V User/Programmer Reference Manual* for more information about initializing processes with RPM.

The following paragraphs are a call-by-call explanation of the dialogue of how a virtual circuit connection is built. The telephone analogy that was used to explain call sockets, VC sockets, and virtual circuits is continued as each call is compared to a certain aspect of the telephone system.

Creating a Call Socket

Interprocess communication is initiated when Process A and Process B each create a call socket by invoking the NetIPC call IPCCREATE. As explained previously, a call socket is roughly analogous to a telephone with multiple extensions (see Figure 1-1). IPCCREATE returns a **call socket descriptor** to the calling process in its *calldesc* parameter that describes the call socket, or "telephone," that the process has created. This call socket descriptor is used in subsequent NetIPC calls.

Figure 1-2. IPCCREATE. (Processes A and B)

Naming a Call Socket

Process B associates a name with its call socket by calling IPCNAME. When a call socket is named, this information is placed in the socket registry at the local node. The name Process B assigns to its call socket must also be known to Process A because Process A must reference it later in its IPCLOOKUP call. (When a socket name is known to both processes in this way, it is called a **well-known name**.) The socket must be named and be in the socket registry at Process B's node when Process A calls IPCLookUp.

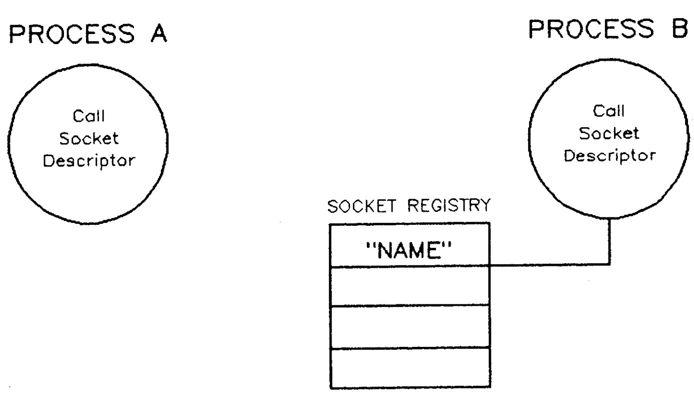


Figure 1-3. IPCNAME. (Process B)

Looking Up a Call Socket Name

Process A must know the name assigned to Process B's call socket. It calls IPCLOOKUP to "look up" the name of the call socket in the socket registry at the node where Process B resides. IPCLOOKUP returns a **destination descriptor** in its *destdesc* parameter. The destination socket described indicates the location of the destination call socket which is owned by Process B. Compared to the telephone system, IPCLOOKUP is similar to directory assistance: Process A calls the "operator" (IPCLOOKUP), and gives him/her a "city" (*location* parameter) and a "name" (*socketname* parameter). Using the "city," that is, the node name or environment id, the operator looks for the name in the proper "telephone directory" (socket registry). Once the name is found, the operator returns a "telephone number" (*destdesc* parameter) to the caller.

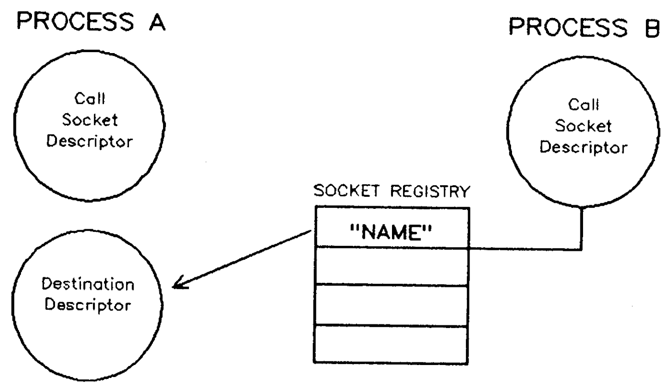


Figure 1-4. IPCLookUp. (Process A)

NOTE

An alternative to using IPCNAME and IPCLOOKUP to name a socket and then obtain its destination descriptor is available through the use of the IPCDEST call. IPCDEST enables you to assign an address to the remote socket. For direct access to level 3 (X.25) the IPCDEST intrinsic must be used to obtain a destination descriptor. Refer to the description of IPCDEST in Section 2 for more information.

Requesting a Connection

Process A specifies the destination descriptor returned by IPCLOOKUP and the call socket descriptor returned by IPCCREATE in its IPCCONNECT call. With these two parameters, IPCCONNECT requests a virtual circuit connection between Process A and Process B. Because of this, IPCCONNECT can be compared to dialing a phone, but not waiting for an answer. IPCCONNECT returns a **VC socket descriptor** in its *vcdesc* parameter that describes the VC socket endpoint of the connection at Process A.

IPCCONNECT is a non-blocking call; it does not suspend the execution of the calling process.

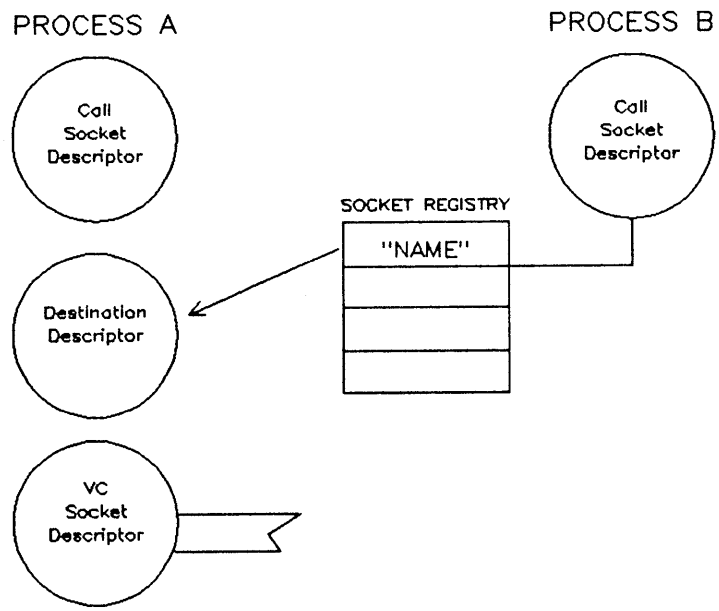


Figure 1-5. IPCCONNECT. (Process A)

Receiving a Connection Request

Using the call socket descriptor returned by its IPCCREATE call, Process B calls IPCRECVCN to receive any connection requests. In this example, Process B will receive a connection request from Process A. (Process A "dialed its telephone" to call Process B when it called IPCCONNECT.) IPCRECVCN returns a **VC socket descriptor** in its *vcdesc* parameter. This VC socket is the endpoint of the virtual circuit at Process B. The connection will not be established, however, until Process A calls IPCRECV. Compared to the telephone system, IPCRECVCN is similar to hearing the telephone ring and answering it.

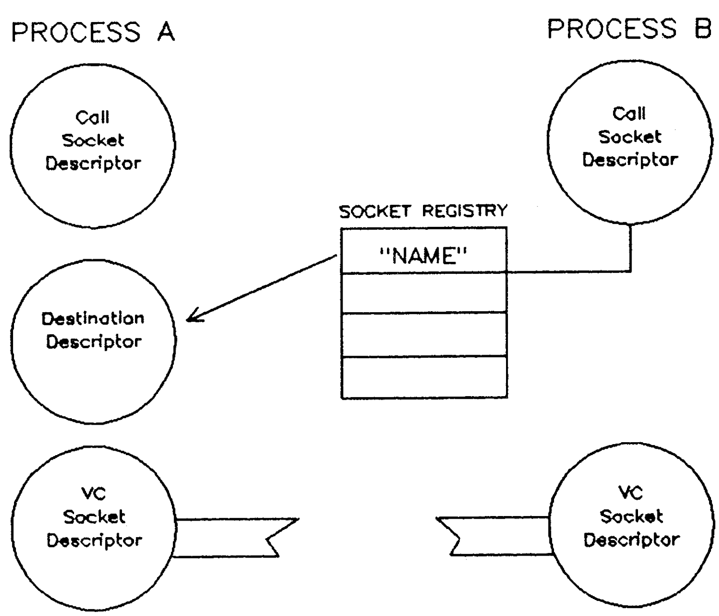


Figure 1-6. IPCRECVCN. (Process B)

Completing a Connection

Process A calls IPCRECV using the VC socket descriptor returned by its IPCCONNECT call. IPCRECV returns the status of the connection (successful/unsuccessful) initiated by IPCCONNECT. If the status is successful, the connection has been established and Process A and Process B can "converse" over the new virtual circuit. Compared to the telephone system, IPCRECV is similar to listening to hear if the phone was answered. IPCRECV can also be used to receive data. This function is described in the IPCRECV call discussion later in this section.

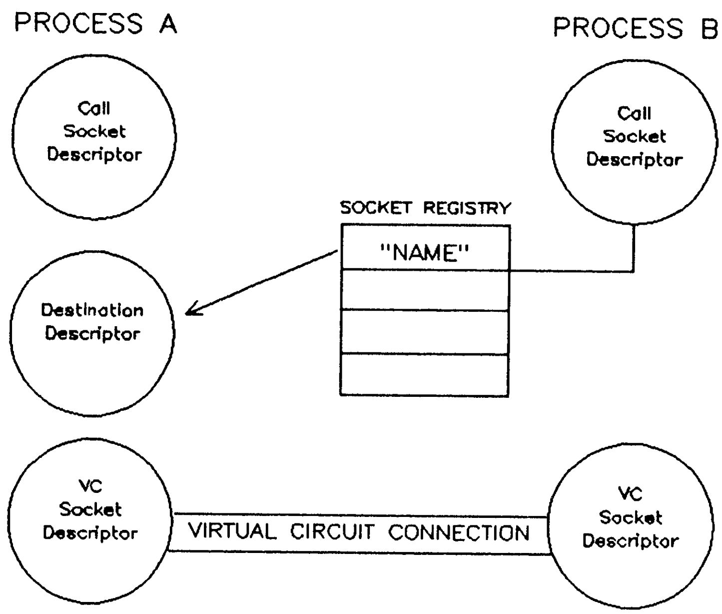


Figure 1-7. IPCRECV. (Process A)

Sending and Receiving Data Over a Connection

Once a virtual circuit connection is established, the two processes can exchange data using the NetIPC calls IPCSEND and IPCRECV. Either process can send or receive data. IPCSEND is used to send data on an established connection. Invoking IPCSEND is analogous to "speaking" over a telephone connection. IPCRECV is used to receive data on an established connection; the use of IPCRECV is similar to "listening" at your telephone handset. (Note that IPCRECV has a dual function: to complete a virtual circuit connection as well as to receive data on a previously established connection.)

X.25 Access

Direct access to level 3 (X.25) provides message mode transfer. Stream mode is not supported for X.25. Each IPCRECV returns a complete message (provided the data length specified is of sufficient size). The X.25 protocol signals the end of message and NetIPC buffers the message until an IPCRECV (or required IPCRECVs) retrieve it.

TCP Access

For TCP access, all data transfers between user processes are in **stream mode**. In stream mode, data is transmitted as a stream of bytes; there are no end-of-message markers. This means that *the amount of data received in an individual IPCRECV request is not necessarily equivalent to a message sent by an IPCSEND call.* In fact, the data received may contain part of a message or multiple messages sent by multiple IPCSEND calls. You specify the maximum number of bytes you are willing to receive through a parameter of IPCRECV. When the call completes, that parameter contains the number of bytes *actually* received. This will never be more than the amount requested by IPCRECV, but it may be less. The data you receive will always be in the correct order (in the order that the messages were sent), but there is no indication of where one message ends and the next one starts. It is up to the receiving process to check and interpret the data it actually receives. An application which does not need the information in the form of individual messages can simply process the data on the receiving side.

If an application *is* concerned about messages, the programmer needs to devise a scheme to allow the receiving side to determine what the messages are. If the messages are of a *known length,* the receiving process can execute a loop which calls IPCRECV with a maximum number of bytes equal to the length of the portion of the message not yet received. Since IPCRECV returns to you the actual number of bytes received, you can continue to execute the loop until all the bytes of the message have been received. The following Pascal program fragment demonstrates this idea:

```
received_len := 0;
while (received_len < msg_length) and (errorcode = 0) do
begin
  data_len := msg_length - received_len;
  ipcrecv( connection, tempbfr, data_len, , , errorcode );
  if errorcode = 0
    then strmove( data_len, tempbfr, 1, databfr, received_len + 1 );
  received_len := received_len + data_len;
end;
```

In the above example, the Pascal function `strmove` takes each piece of the message received in `tempbfr` and concatenates it to the portion of the message already in `databfr`. Upon exiting the loop, the entire message has been stored in `databfr`.

If the length of the messages are *not known,* the sending side could send the length of the message as the first part of each message. In that case, the receiving side must execute two IPCRECV loops for each message: first to receive the length and then to receive the data. An example of this technique is shown at the end of this section.

Shutting Down Sockets and Connections

The NetIPC call IPCSHUTDOWN releases a descriptor and any resources associated with it. IPCSHUTDOWN can be called to release a call socket descriptor, a destination descriptor, or a VC socket descriptor. Since system resources are used up as long as call sockets and destination sockets exist, you may want to release them whenever they are no longer needed.

The call socket is needed as long as a process is expecting to receive a connection request for that socket. A process which receives a connection request can release the call socket any time after the connection request is received via IPCRECVCN, as long as no other connection requests are expected for that call socket.

Similarly, a process which requests a connection can release its call socket any time after the call to IPCCONNECT, as long as it is not expecting to receive a connection request for that socket. In fact, a process which requests a connection need not create a call socket (via IPCCREATE) at all; instead, it can use a temporary call socket by calling IPCCONNECT without specifying a call socket descriptor. (A temporary call socket is automatically destroyed when the IPCCONNECT call completes.) A process which requests a connection can also release the destination socket any time after the call to IPCCONNECT.

For example, referring to Process A discussed in *Establishing a Connection*, Process A no longer needs the destination socket after calling IPCCONNECT (see *Requesting a Connection*). Process A can then call IPCSHUTDOWN to release the destination socket. In addition, if Process A does not expect to receive connection requests, it can call IPCSHUTDOWN a second time to release the call socket.

Process B, as described in *Establishing a Connection*, can call IPCSHUTDOWN to release its call socket any time after the call to IPCRECVCN (see *Receiving a Connection Request*). Process B should release its call socket only if it does not want to establish additional connections.

Before a process terminates, it should terminate its virtual circuit connections by releasing its VC sockets with IPCSHUTDOWN. If a process does not release its VC sockets before terminating, the system releases them when the process terminates. Because IPCSHUTDOWN takes effect very quickly, *all of the data that is in transit on the connection is lost when the connection is shut down.* As a result, if there is a possibility that data would be in transit on the connection, the processes that share a connection must cooperate to ensure that no data is lost.

X.25 Access

X.25 direct access to level 3 does not support the *graceful release* bit. Using IPCSHUTDOWN on a VC socket description causes a clear packet to be sent. As a suggestion, to ensure that no data packets are lost before the clear packet is sent, the D bit option could be set in the last IPCSEND. This would assure end-to-end acknowledgement of this message before issuing the IPCSHUTDOWN to clear the virtual circuit.

TCP Access

To ensure that no data is lost, the IPCSHUTDOWN *graceful release* bit can be set, and the following sequence of steps can be followed:

1) Process A calls IPCSHUTDOWN and sets bit 17, the graceful release flag. Process B receives a message (with an IPCRECV) informing it that Process A has called for graceful release. (This message is sent to B automatically when A sets the graceful release flag.) Process A enters a simplex-in state; that is, it can receive data but not send any. Process B will enter a simplex-out state, in which it can send data but not receive any. As a result, data that is in transit to Process A (which initiated the graceful release shutdown) will reach Process A without being lost.

2) Next, one of two steps must occur to completely shut down the connection. Either (1) Process B initiates its own graceful release or (2) Process A calls IPCSHUTDOWN *without* the graceful release option. This releases Process A's VC socket descriptor and shuts down the connection. In this case, Process B must also release its socket descriptor by calling IPCSHUTDOWN.

If the graceful release option is not used (this may be necessary, for example, if the remote node does not support graceful release) the following steps should be followed when shutting down a connection:

1) Process A sends a "last message" to Process B via an IPCSEND call. This message contains data that will be recognized by Process B as a termination request, and may also contain data to be processed by Process B. Process A then calls IPCRECV.

2) Process B receives Process A's message with a call to IPCRECV and sends a "confirmation message" to Process A via IPCSEND. This message contains data that indicates to Process A that it is okay to terminate the connection, and may also contain data to be processed by Process A. Process B then calls IPCRECV.

3) Process A receives Process B's "confirmation message" via the call to IPCRECV and calls IPCSHUTDOWN to release its VC socket descriptor and shut down the connection.

4) Process B's IPCRECV completes with a *result* parameter value of 64 ("REMOTE ABORTED THE CONNECTION"). It then calls IPCSHUTDOWN to release its VC socket.

Additional NetIPC Calls

Once a virtual circuit is established between processes, call or VC descriptors can be given away, names can be erased, and other functions can be performed. The following NetIPC calls are provided in addition to those described in the previous paragraphs to enable you to perform these functions. A brief introduction to each call and its use follows. (A complete description of these and all of the NetIPC calls is provided in Section 2.)

- **IPCCONTROL**. Performs special operations on sockets such as enabling synchronous mode, and changing asynchronous timeout values.

- **IPCDEST**. Returns a destination descriptor which can be used to send messages to another process. This is an alternative to naming the descriptor with IPCNAME and acquiring it with IPCLOOKUP.

- **IPCGET**. The companion call to IPCGIVE. Receives a descriptor given away by a process that has called IPCGIVE. This call is similar to IPCLOOKUP because it enables your process to acquire a descriptor that can be used in subsequent NetIPC calls.

- **IPCGIVE**. The companion call to IPCGET. Releases ownership of a descriptor to NetIPC so that it can be acquired by another process via a call to IPCGET.

- **IPCNAMERASE**. Does the reverse of IPCNAME: it removes a name associated with a call socket from the socket registry. Only the owner of a call socket descriptor can remove its name.

DIRECT ACCESS TO LEVEL 3 (X.25)

Features

Features of direct access to level 3 (X.25) with NetIPC are:

- Supports switched virtual circuits (SVCs) and permanent virtual circuits (PVCs).
- Provides access to the call user data (CUD) field in data packets.
- Creation of a catch-all socket which can be used to accept data packets with no CUD or unknown CUDs.
- Provides access to X.25 protocol options.

Limitations

Limitations using direct access to level 3 (X.25) are:

- Intranet use only (level 4 provides internet and intranet connections)
- One virtual connection socket accesses one X.25 virtual circuit for data transfers over X.25. Multiplexing of connections over a virtual circuit is not supported.
- Message mode transfer of data only. Stream mode is not supported.
- IPCNAME, IPCNAMERASE and IPCLOOKUP are not supported.

Switched Virtual Circuits (SVCs)

Switched virtual circuits are defined as a logical association that only exists as long as the connection does. Both processes create their own local call sockets using IPCCREATE that can be associated with protocol relative addresses. To establish a connection with a specific server process, a request process can include a server protocol relative address in the IPCDEST intrinsic. Alternatively, an *opt* parameter in IPCCREATE can be used to create a catch-all socket where any incoming request for a connection can be accepted (whether or not the server protocol relative address exists or has been included in IPCDEST). A catch-all socket receives incoming call requests that do not match any other given protocol relative address. One catch-all socket can be defined for each X.25 network.

As an example, two programs communicating over an SVC can be designated as the requester and server. Both programs need to be running in order for communication to occur. Figure 1-8 shows the order of NetIPC calls used for a requestor program and the X.25 packets generated as a result of the calls. Figure 1-9 describes the order of NetIPC calls used for a server program.

The calls outlined in Figure 1-8 perform the following functions:

1. Create a call socket with IPCREATE. The call socket descriptor (*calldesc*) is returned.
2. Create a destination descriptor socket (*destdesc*) with IPCDEST. You can specify a remote protocol relative address (*protoaddr*) to be associated with the destination descriptor.
3. Establish the virtual circuit socket with IPCCONNECT, supplying the *calldesc* and *destdesc* created by the previous two calls.
4. Receive a response to the connection request with IPCRECV, setting the data length parameter (*dlen*) equal to zero.
5. Send a message over the connection with IPCSEND.
6. Receive a message over the connection with IPCRECV.
7. Shutdown the connection with IPCSHUTDOWN. Cause and diagnostic values can be entered that will be included in an X.25 clear packet sent as a result of this call.
8. The IPCSHUTDOWN intrinsic will not complete until X.25 has received a clear confirmation packet.

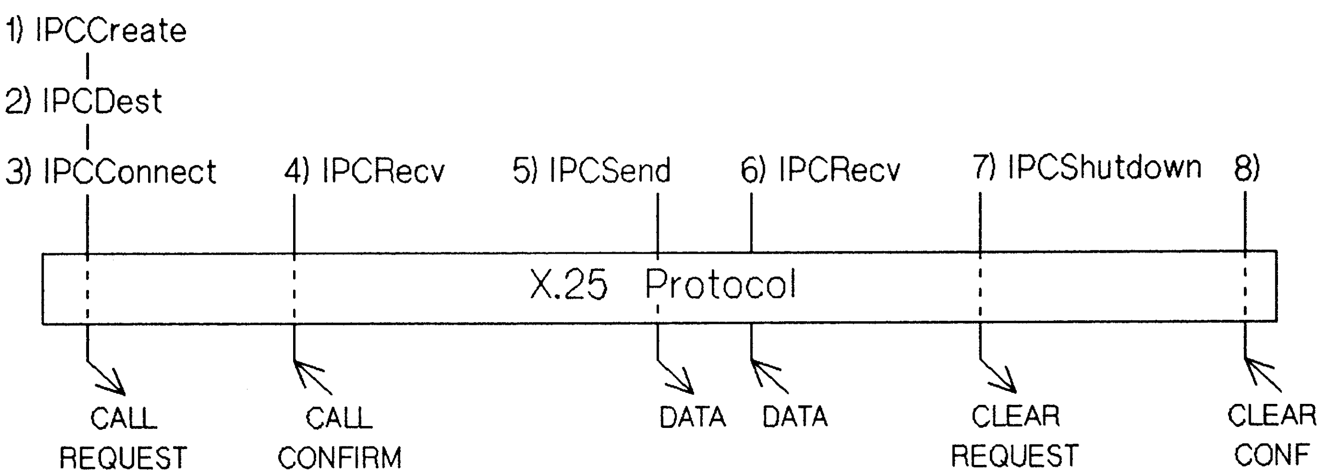


Figure 1-8. SVC Requestor Processing Example

Figure 1-9 shows the order of NetIPC calls used for a server program and the X.25 packets generated as a result of the calls. The calls outlined in Figure 1-9 perform the following functions:

1. Create a call socket with IPCREATE. The call socket descriptor (*calldesc*) is returned. The socket could be created as a catch-all or bound to a protocol relative address.

2. Call IPCRECVCN and wait for an incoming call request packet. IPCRECVCN will return a VC descriptor (*vcdesc*) when it is established that the incoming protocol relative address defined in (1) matches the incoming protocol relative address, or a catch-all socket was created in (1).

3. As IPCRECVCN completes and returns a *vcdesc*, X.25 sends the requestor process a call accepted packet.

4. Receive a message over the connection with IPCRECV.

5. Send a message over the connection with IPCSEND.

6. Since the server (IPCRECV) in this example waits to receive a message, you may decide to set a timer to handle the inactivity.

7. (Optional step.) Shutdown the connection with IPCSHUTDOWN after data has not been received for a period of time. (For example, after a timeout has occurred.) Note that the X.25 protocol implicitly handles the incoming clear request by sending a clear confirmation packet.

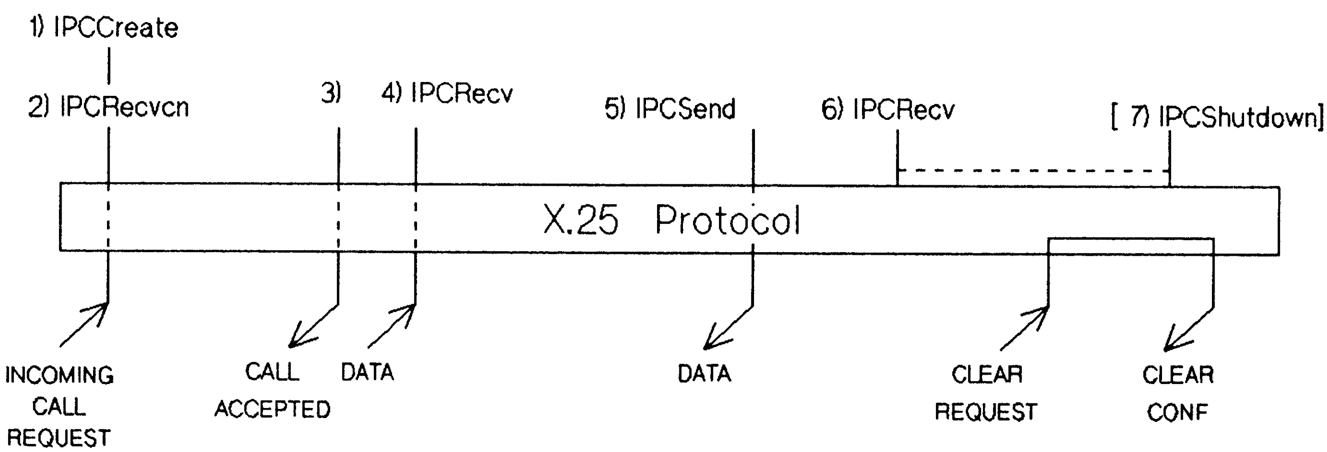


Figure 1-9. SVC Server Processing Example

Note that Figures 1-8 through 1-9 do not show synchronization of data transfer between the two programs, and do not include error checking, or the intrinsic calls required for adding options and special user capabilities. See example 4 in section 3 of this manual for programmatic examples of a server and requestor using access to the X.25 protocol.

Permanent Virtual Circuits (PVCs)

Permanent virtual circuits are defined as two DTEs with a logical association permanently held by the network. Since the connection is permanent, both processes must initiate the connection using the IPCCREATE intrinsic. Both processes must specify the destination of a connection request with the IPCDEST intrinsic which requires a node name corresponding to a configured PVC number.

The possible ordering of intrinsic calls to communicate over a PVC could be as follows:

1. Create a call socket with IPCREATE. The call socket descriptor (*calldesc*) is returned.
2. Create a destination descriptor socket (*destdesc*) with IPCDEST.
3. Establish the virtual circuit socket with IPCCONNECT, supplying the *calldesc* and *destdesc* created by the previous two calls.
4. Send a reset packet (to the DCE) by setting the reset request in IPCCONTROL.
5. Send an interrupt packet to the remote process by setting the interrupt request in IPCCONTROL.
6. Send data over the connection with IPCSEND.
7. Receive data over the connection with IPCRECV.
8. Send a reset packet by setting the reset request in IPCCONTROL when all data has been sent/received.
9. Shutdown the connection with IPCSHUTDOWN. Note that a PVC is a permanent connection, and the shutdown process causes the connection to go to a reset state.

Note that these steps do not show how to synchronize data transfer between the two programs, and do not include error checking, or the intrinsic calls required for adding options and special user capabilities.

Access to the Call User Data (CUD) Field

The NetIPC intrinsics IPCCONNECT, IPCRECVCN and IPCCONTROL provide access to the call user data (CUD) field in data packets as follows:.

- **Specifying a protocol relative address in the CUD.**

  This field may be present in X.25 call request and incoming call packets which you can access with IPCCONNECT and IPCRECVCN. The call user data field can only be accessed over an SVC. The maximum length of the call user data (CUD) field is 16 bytes. In the NS X.25 3000/V implementation of X.25, the first four bytes of the CUD are reserved for protocol relative addressing. Figure 1-10 shows the contents of the first four bytes of the NS X.25 CUD. The first two bytes, as shown in Figure 1-10, indicate that the source of the call request packet is an NS X.25 3000/V node using direct access to level 3. Optionally, the last two bytes contain the protocol relative address that the call request expects to find (if any).

  To access all 16 bytes of the CUD, the *opt* parameter *protocol flags* bit 17 can be set in IPCCONNECT. This option is useful for communication with non-HP nodes.

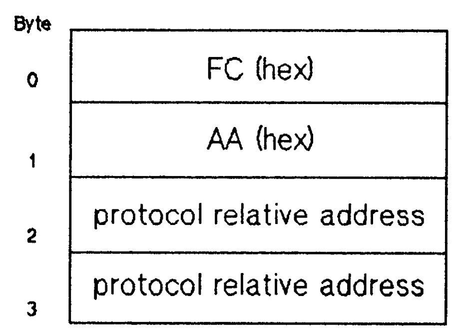


Figure 1-10. NS X.25 Call User Data Field (first four bytes)

- **Connecting to a catch-all socket.**

  Using IPCCREATE, you can identify a socket as a catch-all socket over an SVC. All incoming calls with a protocol relative address specified in the CUD that does not match any given protocol relative address are routed to the catch-all socket. One catch-all socket may be defined for each X.25 network.

  For an incoming call with a protocol relative address specified, NetIPC checks if the address matches one created. If it matches, the call is accepted. If it does not match, NetIPC checks for the existence of a catch-all socket. If no catch-all socket has been created, the call is rejected and a clear packet is sent by X.25. If a catch-all socket has been created, the call is accepted.

  If no protocol address is specified in the incoming call, NetIPC checks for the existence of a catch-all socket. If no catch-all socket has been defined, the call is rejected. If there is a catch-all socket, the call is accepted.

- **Defer connection requests**

  The IPCCONTROL intrinsic provides you with the capability to accept or reject a connection request that is in the deferred state. Using the intrinsic IPCCONTROL, it is possible to inspect the inbound CUD and/or the calling DTE address before accepting the call.

Access to X.25 Protocol Options

The NetIPC intrinsic parameters *flags* and *opt* provide access to the following X.25 protocol functions:

- **Qualifying X.25 data packets**

  The Q bit in the general format identifier field in an X.25 data packet can be set using the IPCSEND intrinsic. The status of the Q bit in incoming data packets is returned in the IPCRECV intrinsic. The Q bit status indicates whether the data is a user message (Q bit=0) or a device control message (Q bit=1) from or to a remote PAD.

- **Set end-to-end acknowledgment.**

  The D bit in the general format identifier field in an X.25 data packet can be set using the IPCSEND intrinsic. The status of the D bit in incoming data packets is returned in the IPCRECV intrinsic.

  Setting the D bit locally specifies end-to-end acknowledgment of data packets. IPCSEND does not complete until it receives acknowledgment that the entire message has been received. For HP 3000 to HP 3000 communication, IPCRECV initiates the acknowledgment when the remote HP 3000 process calls IPCRECV.

- **Identify a facilities set.**

  For an SVC, you can specify a facilities set name in the IPCCONNECT intrinsic. The facility sets are created when you configure the X.25 link with NMMGR. If no facility set is specified, the facilities set defaults to the NMMGR configured facility set. For a PVC, the facility set cannot be specified with IPCCONNECT and the facility set configured in NMMGR is used.

- **Set cause and diagnostic codes.**

  Using IPCSHUTDOWN, you can enter a reason code that will be included in X.25 clear packets as cause and diagnostic values. This option is only used with SVCs. Reasons for events or errors are returned by IPCCONTROL. See Appendix A for a list of diagnostic codes used with X.25 protocol access. Note that when the DTE sends the clear packet, the cause code is always set to zero.

- **Send and receive interrupt and reset packets.**

  You can request the X.25 protocol to send an interrupt or reset packet with IPCCONTROL. When used in this way, the IPCCONTROL intrinsic will not return until the appropriate confirmation packet is received by X.25.

- **Set no activity timeout.**

  You can set a no activity timeout value with the IPCCONTROL intrinsic. This option clears the connection after the specified time if no data packets are exchanged on the virtual circuit.

CROSS-SYSTEM NETIPC FOR TCP ACCESS

A cross-system application refers to NetIPC communication between processes running on computers of different types. Cross-system NetIPC is supported using access to the Transmission Control Protocol (TCP) only. This section explains what NetIPC calls using TCP access need to be considered for a cross-system application between an HP 3000 and HP 1000 and between an HP 3000 and HP 9000 (Series 300 or 800). Cross-system NetIPC is also supported between HP 3000s and personal computers (PCs) in an HP Office Share Network. See the *PC NetIPC/RPM Programmers' Reference Guide* (50924-90000) for programming considerations and the NetIPC calls available on the PC.

NetIPC communication between MPE-V based and MPE/XL based HP 3000s is not considered cross-system. See the *NetIPC 3000/XL Programmer's Reference Manual* for more information about NetIPC on MPE/XL based HP 3000s.

This section does *not* explain details about the NetIPC calls available on the HP 1000 or HP 9000. For this information, refer to the following manuals:

- *NS/1000 User/Programmer Reference Manual* (91790-90020)
- *HP 9000 NetIPC Programmer's Guide (for the Series 300 and 800)* (98194-90002)

For cross-system NetIPC to function properly, the software revision codes must be as follows:

- NS/1000 software revision code 5.0 or greater for the HP 1000
- NS3000/V V-delta-1 MIT (Master Installation Tape) or later (to be used with IEEE 802.3 LAN only) for the HP 3000
- LAN/9000 Series 800 Release 2.1 or later for the HP 9000 Series 800
- NS-ARPA Services Release 6.2 or later for the HP 9000 Series 300

To use this "Cross-System NetIPC" section, you must first have a good understanding of the NetIPC calls. Review the remaining sections on the calls before and while you read this section. For an example of programs for an HP 3000 system that will communicate with similar programs on an HP 1000 system, or on an HP 9000 system, refer to Section 3, example 2.

There are two categories of calls when considering cross-system NetIPC communication -- local and remote. Calls made for the local process do not directly affect the remote process. The local NetIPC calls are used to set up or prepare the local node for interprocess communication with the remote node. That is, the resulting impact on the local calls is only to the local node. There is no information that needs to be passed to the remote node. This is true whether or not the remote node is another HP 3000.

The intrinsics listed in Table 1-2 affect local processes only and will therefore have no adverse effects if used in a program communicating with an unlike system (e.g., an HP 3000 program communicating with an HP 1000 program). However, keep in mind that the calls (even those of the same name) differ from system type to system type. The following are some local call differences to be aware of:

- **Maximum number of sockets.** The maximum number of socket descriptors owned by an HP 3000 process at any given time is 64; on the HP 1000 the maximum is 32; on the HP 9000, the maximum is 60 (including file descriptors). (This number includes both call socket and virtual circuit socket descriptors.)

- **IPCCONTROL parameters.** The IPCCONTROL intrinsic supports different sets of request codes on different system types. Refer to the NetIPC documentation for a particular system for a full description of the request codes available on that system. This manual describes HP 3000 request codes only.

- **Manipulation of descriptors.** On the HP 3000, the IPCGIVE, IPCGET, IPCNAME, and IPCNAMERASE calls can be used to manipulate call socket and VC socket descriptors. You can manipulate call socket and destination descriptors on the HP 9000 with the ipcname() and ipcnamerase() intrinsics, and on the HP 1000 with the IPCName and IPCNamerase intrinsics. In addition, on the HP 1000, you can manipulate call socket and destination descriptors with the IPCGive and IPCGet intrinsics.

- **Asynchronous I/O.** The HP 3000 utilizes the MPE intrinsics IOWAIT and IODONTWAIT to perform asynchronous I/O. On the HP 9000 and HP 1000, the NetIPC intrinsics ipcselect() and IPCSelect are used to perform asynchronous I/O.

TABLE 1-2. NetIPC Calls Affecting The Local Process

HP 3000	HP 1000	HP 9000
ADDOPT	Addopt	addopt()
(Not implemented)	Adrof	(Not implemented)
INITOPT	InitOpt	initopt()
IPCCONTROL	IPCControl	ipccontrol()
IPCCREATE	IPCCreate	ipccreate()
IPCGET	IPCGet	(Not implemented)
IPCGIVE	IPCGive	(Not implemented)
IPCNAME	IPCName	ipcname()
IPCNAMERASE	IPCNamerase	ipcnamerase()
(Not implemented)	IPCSelect	ipcselect()
OPTOVERHEAD	(Not implemented)	optoverhead()
READOPT	ReadOpt	readopt()

NOTE

There are many additional differences between local NetIPC calls for the HP 3000 and those used for other HP systems. Refer to the corresponding system's NetIPC documentation for more information.

Table 1-3 lists the NetIPC calls affecting cross-system communication with the remote process. The table also describes differences between each call on the HP 3000, HP 1000 and HP 9000, if the difference will affect cross-system communication.

TABLE 1-3. NetIPC Calls Affecting The Remote Process

NetIPC Call	Cross-System Considerations
IPCConnect	Checksumming - TCP checksumming will be enabled for both sides of the connection if it is enabled by either side for HP 3000 to HP 1000 or HP 3000 to HP 9000 cross-system communication. Checksumming is always enabled on the HP 9000. On the HP 3000, enabling/disabling checksumming with NetIPC intrinsics allows you to override the checksumming decision made during network transport configuration for this particular process. Send and receive sizes - The HP 3000 send and receive size range is 1 to 30,000 bytes. The HP 1000 send and receive size range is 1 to 8,000 bytes. The HP 9000 send and receive size range is 1 to 32,767 bytes. Although the ranges are different, you must specify a send size within the correct range for the respective receiving system; otherwise, an error will occur. For example, if the HP 3000 node sends 16,000 bytes, the HP 1000 node can call IPCRecv twice, receiving the first 8,000 bytes the first time and the second 8,000 bytes the second time. Note that the default send and receive sizes are different on different HP systems. On the HP 3000, the default send and receive size is less than or equal to 1,024 bytes. On the HP 1000 and HP 9000, the default send and receive size is 100 bytes.
IPCDest	TCP protocol address - The recommended range of TCP addresses for cross-system user applications is from 30767 to 32767 decimal (%74057 to %77777) for the HP 3000, HP 1000 and HP 9000.
IPCLookUp	No differences that affect cross-system operations.
IPCRecv	Receive size (*dlen* parameter) - Range for the HP 3000 is 1 to 30,000 bytes. Range for the HP 1000 is 1 to 8,000 bytes. Range for the HP 9000 is 1 to 32,767 bytes. Refer to the discussion of send and receive sizes for IPCConnect and IPCRecvcn. Data wait flag - The HP 1000 and HP 9000 IPCRecv call supports a "DATA_WAIT" flag. This flag, when set, specifies that the call will not complete until the amount of data specified by the *dlen* parameter has been received. This flag is not available on the HP 3000, meaning that the call may complete before all the data is received. However, the HP 3000 IPCRecv supports other flags such as the "more data" and "destroy data" flags. Refer to the description of IPCRecv in Section 2 for more information.

TABLE 1-3. NetIPC Calls Affecting The Remote Process (cont'd)

NetIPC Call	Cross-System Considerations
IPCRecvCn	Checksumming - TCP checksumming will be enabled for both sides of the connection if it is enabled by either side for HP 3000 to HP 1000 or HP 3000 to HP 9000 connections. Checksumming is always enabled on the HP 9000. On the HP 3000, enabling/disabling checksumming with NetIPC intrinsics allows you to override the checksumming decision made during network transport configuration for this particular process. Send and receive sizes - The HP 3000 send and receive size range is 1 to 30,000 bytes. The HP 1000 send and receive size range is 1 to 8,000 bytes. The HP 9000 send and receive size range is 1 to 32,767 bytes. Although the ranges are different, you must specify a send size within the correct range for the respective receiving system; otherwise, an error will occur. For example, if the HP 3000 node sends 16,000 bytes, the HP 1000 node can call IPCRecv twice, receiving 8,000 bytes the first time and the second 8,000 bytes the second time. Note that the default send and receive sizes are different on different HP systems. On the HP 3000, the default send and receive size is less than or equal to 1,024 bytes. On the HP 1000 and HP 9000, the default send and receive size is 100 bytes.
IPCSend	No differences that affect cross-system operations. Note that the *urgent data* bit is not supported on the HP 1000; however, if this bit is set by the HP 3000 program, it will be ignored by the receiving process on the HP 1000. For differences in send and receive sizes see the discussion for IPCRecvcn.
IPCShutDown	Socket shut down - The HP 3000 provides a graceful release flag that is not available on the HP 1000 or HP 9000. If the graceful release flag (flags 17) is set on the HP 3000, the HP 1000 will respond as though it were a normal shutdown. The HP 3000 and HP 1000 do not support shared sockets; the HP 9000 does. Shared sockets are destroyed only when the descriptor being released is the sole descriptor for that socket. Therefore, the HP 9000 process may take longer to close the connection than expected.

NOTE

There are many additional differences between NetIPC calls for the HP 3000 and those for other HP systems. However, these differences should not affect the cross-system communication capabilities of your program because they affect the local node only. Refer to the corresponding system's NetIPC documentation for more information.

NetIPC Error Codes

NetIPC calls with the same names on different systems may return different error codes. Refer to the system's NetIPC documentation for a complete list of the NetIPC error codes that are applicable to your implementation.

Program Startup

NetIPC itself does not include a call to schedule a peer process. In programs communicating between multiple HP 3000s, you can use the **Remote Process Management (RPM)** call `RPMCreate` to programmatically schedule program execution. However, RPM between HP 3000s and HP 1000s, and HP 3000s and HP 9000s is not currently supported by Hewlett-Packard. Instead, you must manually start up each NetIPC program on its respective system.

HP 3000 Program Startup

To manually start up an HP 3000 NetIPC program, log on to the HP 3000 and run the NetIPC program (with the `RUN` command).

You can schedule the program to start at a particular time by writing a job file to execute the program, and then including time and date parameters in the `STREAM` command that executes the job file.

HP 1000 Program Startup

To manually startup an HP 1000 NetIPC program, logon to the HP 1000 system and run the NetIPC program with the RTE `XQ` (run program without wait) command.

To have the NetIPC program execute at system start up, put the RTE `XQ` command in the WELCOME file.

HP 9000 Program Startup

Remote HP 9000 processes can be manually started or can be scheduled by daemons that are started at system start up. In HP-UX a daemon is a process that runs continously and usually performs system administrative tasks. Although a daemon runs continuously, it performs actions either when an event occurs, or at designated times.

To manually start up a NetIPC program, logon to the HP 9000 system and run the NetIPC program. HP recommends that you write a NetIPC daemon to schedule your NetIPC programs. You can start the daemon at start up by invoking it from the `/etc/netlinkrc` file.

NETIPC INTRINSICS

SECTION 2

COMMON PARAMETERS

The *flags*, *opt*, *data*, and *result* parameters are common to many NetIPC intrinsics. Remote Process Management intrinsics also use these parameters, with the exception of the *data* parameter. The following discussion of these parameters may help to clarify the more condensed information given under each intrinsic.

Flags Parameter

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

Opt Parameter

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

The *opt* parameter consists of these fields, as shown in Figure 2-1:

- length, in bytes, of option entries and data (2-byte integer), where

  `length=(8 * number of entries) + length of data;`

- number of entries (2-byte integer);
- option entries (eight bytes per entry);
- data associated with the option entries (variable length).

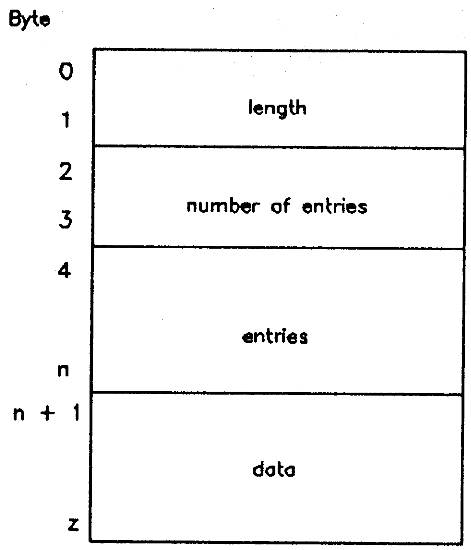


Figure 2-1. Opt Parameter Structure

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

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

Figure 2-2 shows the structure of each option entry.

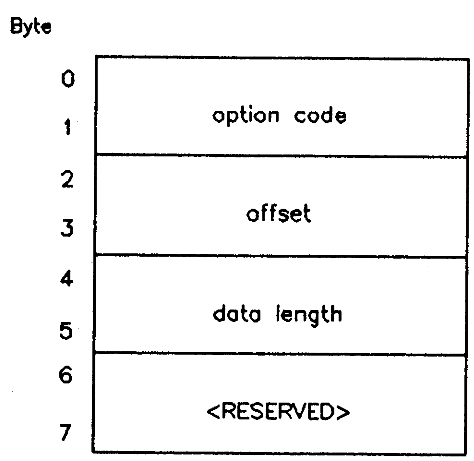


Figure 2-2. Option Entry Structure

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

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

NOTE

Use of certain *opt* parameter options may result in the loss of portability between heterogeneous HP machines.

Data Parameter

The data transmitted by NetIPC intrinsics can in most cases be vectored. In the case of vectored data, the *data* parameter does not contain actual data but rather the addresses from or to which the data will be gathered or distributed. The *data* parameter may always be defined as a byte array. If the data are vectored, the parameter may also be a record explicitly structured in the manner described below.

The addresses of the data are represented by data location descriptors. For all intrinsics supporting vectored data, a maximum of two data location descriptors is permitted. Each data location descriptor is eight bytes long and consists of four 2-byte fields as shown in Figure 2-3:

- the descriptor type (represented by a 2-byte integer);
- a DST (data segment) number or index;
- a byte offset (from DB on the calling process's stack or on an extra data segment) indicating the location of the data;
- the length in bytes of the data.

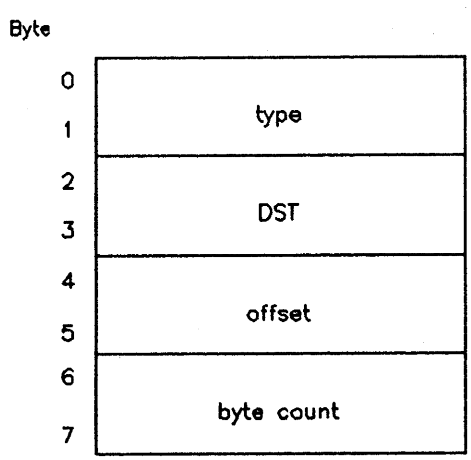


Figure 2-3. Data Location Descriptor Structure

The descriptor type field can have one of the following values:

- 0--the offset is a DB-relative byte address on the calling process's data stack (the DST is ignored);
- 1--DST is the logical index number returned by the MPE V intrinsic;
- 2--DST is an actual data segment number.

All data segment references require privileged mode.

The *dlen* parameter indicates the length of the *data* parameter. If the data are vectored, *dlen* must give the total length of the data location descriptors (i.e. 8 or 16 bytes), not the length of the actual data. Actual data can be from 1 to 30,000 bytes long for both vectors combined.

Result Parameter

If a NetIPC (or Remote Process Management) intrinsic call that uses waited I/O is successful, the *result* parameter will return a value of zero. Otherwise the value returned represents a NetIPC error code. NetIPC error messages are listed in the *NS3000/V Error Message and Recovery Manual.* You can also obtain the appropriate error message by calling IPCERRMSG.

NOTE

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

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

Summary of NetIPC Intrinsics

Table 2-1. NetIPC Intrinsics

Intrinsic	Function
ADDOPT	Adds an option entry to the *opt* parameter.
INITOPT	Initializes the *opt* parameter so that entries may be added.
IPCCHECK	Returns the number of the last recorded error for a call or VC socket.
IPCCONNECT	Requests a connection (a virtual circuit) to another process, returning a VC socket descriptor for a VC socket belonging to the calling process.
IPCCONTROL	Performs special operations such as enabling nowait I/O, enabling user-level tracing, and enabling software interrupts.
IPCCREATE	Creates a call socket for the calling and called process..
IPCDEST	Returns a destination descriptor which the calling process can use to establish a connection to another process.
IPCERRMSG	Returns the IPC error message corresponding to a given error code.
IPCGET	Enables the calling process to obtain a call or VC socket that has been given away by another process.
IPCGIVE	Gives away a call or VC socket, thereby allowing another process to obtain it.
IPCLOOKUP	Returns a destination descriptor associated with a given socket name. Used with TCP access only.
IPCNAME	Specifies a name for a call socket, thereby enabling other processes to obtain access to that socket. Used with TCP access only.
IPCNAMERASE	Deletes a call socket name from the socket registry. Used with TCP access only.

Table 2-1. NetIPC Intrinsics (cont.)

Intrinsic	Function
IPCRECV	Receives the reply to a connection request, thereby establishing the connection, or receives data on an already-established connection.
IPCRECVCN	Receives a connection request from another process, returning a VC socket descriptor.
IPCSEND	Sends data on a connection.
IPCSHUTDOWN	Releases a socket descriptor and any resources associated with it.
OPTOVERHEAD	Returns the amount of space needed for the *opt* (option) parameter, a parameter common to many IPC intrinsics.
READOPT	Allows the user to read an entry from the *opt* array. Useful for looking at an entry when it is received as output by an intrinsic.

Capabilities

Some NetIPC intrinsics require special capabilities if you use the functions described below.

User-specified Protocol Addressing

NetIPC intrinsics IPCCONNECT, IPCREATE, and IPCDEST allow you to specify protocol relative addresses. Addresses in the range %74057 to %77777 can be used without special capabilities. In privileged programs you can specify protocol relative addresses between %1 and %74056.

NOTE

> The protocol relative address range %1 to %74056 is administered by HP. Contact your HP representative before using an address within this reserved range.

X.25 Catch-all Socket

Using access to X.25 (level 3), network administrator (NA) capability is required to create a catch-all socket for an X.25 network. NA capability is required to run a program that creates a catch-all socket.

Declaring NetIPC Intrinsics in Programs

All NetIPC intrinsics must be declared in your program. See the examples in section 3 of this manual for Pascal declarations. Refer to the appropriate language reference manuals for declarations in other languages.

ADDOPT

Adds an option entry to the *opt* parameter.

Syntax

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

Parameters

opt (input/output)	**Record or byte array, by reference.** The *opt* parameter to which you want to add an entry. Refer to "NetIPC Intrinsics/Common Parameters" for more information on the structure of this parameter.
entrynum (input)	**16-bit integer, by value.** Indicates which entry is to be initialized. The first entry is entry zero.
optioncode (input)	**16-bit integer, by value.** The entry's option code, identifying the option.
datalength (input)	**16-bit integer, by value.** The length (in bytes) of the data associated with the option.
data (input)	**Byte array, by reference.** The data associated with the option.
result (output)	**16-bit integer, by reference.** The error code returned; zero if no error.

Discussion

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

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

`INITOPT (opt, 1);`	{one option entry}
`ADDOPT (opt, 0, 8, 2, data_offset);`	{first entry is entry zero, option code 8; entry's data area contains a 2-byte integer specifying an offset from *data* parameter address}
`IPCSEND (cd, data, dlen, , opt, result);`	{sends data located at offset from *data* address specified in *opt*}

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

Condition codes returned by ADDOPT are:

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

This intrinsic may be called in split stack mode.

INITOPT

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

Syntax

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

Parameters

opt (output)
: **Record or byte array, by reference.** The *opt* parameter which is to be initialized. Refer to "NetIPC Intrinsics/Common Parameters" for more information on the structure of this parameter.

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

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

Discussion

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

Condition codes returned by this intrinsic are:

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

This intrinsic may be called in split stack mode.

Returns the number of the last applicable error.

Syntax

```
IPCCHECK (descriptor[,ipcerr][,pmerr][,result])
```

Parameters

descriptor (input)
: **32-bit integer, by value.** The call socket or VC socket descriptor for which the error is to be reported. A zero value indicates the last call socket or VC socket descriptor referenced.

ipcerr (output)
: **32-bit integer, by reference.** The error code of the last recorded NetIPC error.

pmerr (output)
: **32-bit integer, by reference.** The error code of the last recorded Transmission Control Protocol (TCP) or X.25 protocol error.

result (output)
: **32-bit integer, by reference.** The error code returned for this intrinsic call (not the previously recorded error). A zero value indicates no error.

Discussion

The IPCCHECK intrinsic returns the last recorded NetIPC and/or protocol module error for a given call socket or VC socket (i.e. the VC socket at the calling process's end). If the *descriptor* value is zero, the most recent error applicable to the last call or VC socket referenced is returned. The *descriptor* is the only required parameter (option variable).

Condition codes returned by this intrinsic are:

- CCE--The intrinsic call was successful.
- CCL--Unsuccessful.
- CCG--Unsuccessful. The intrinsic could not return the error code because the data structure which retains error codes has been released.

Split stack calls are permitted.

IPCCONNECT

Requests a connection to another process.

Syntax

```
IPCCONNECT ([calldesc],destdesc[,flags][,opt],vcdesc[,result])
```

Parameters

calldesc (input)

32-bit integer, by value. A call socket descriptor for a call socket belonging to this process. For TCP access, if -1, or if omitted, a call socket is created temporarily to establish the connection.

destdesc (input)

32-bit integer, by value. Destination descriptor. Describes the location of the named call socket. (this is the call socket to which the connection request will be sent). A destination descriptor can be obtained by calling IPCDEST. For TCP access, you can also obtain a destination descriptor by calling IPCLOOKUP.

flags (input)

32 bits, by reference. A bit representation of various options. No flags are defined for access to the X.25 protocol. The following flags are defined for access to TCP:

- flags [0] (input). (TCP only.) Makes the connection a "protected" one. A protected connection is one which only privileged users may establish or use.
- flags [21] (input). (TCP only.) Enables checksum on the Transmission Control Protocol (TCP) connection for error checking. Checksum may also be set by the corresponding IPCRECVCN call. If either side specifies "checksum enabled" then the connection will be checksummed. TCP checksum may be enabled globally, over all connections, when configuring the Network Transport. See the *NS3000/V Network Manager Reference Manual, Volume I* for details on Network Transport configuration. Checksum enabled by either IPCRECVCN or TCP (remote or local) configuration overrides a 0 setting (checksum disenabled) for this flag. Checksum error checking is handled at the link level and is not normally required at the user level. Enabling checksum may reduce network performance. Recommended value: 0.

opt (input)

Record or byte array, by reference. A list of options, with associated information. Possible options are:

- call user data (code=2, length=n, n bytes) (input). For access to the X.25 protocol only. This option contains data to be inserted as the call user data (CUD) field in an X.25 packet. The maximum length for the CUD is 16 bytes. HP has reserved the first four bytes of the CUD for protocol addressing. The user can supply data up to 12 bytes. By setting the no address flag (protocol flags option), the user can access all 16 bytes of the CUD. See section 1, Access to the Call User Data (CUD) Field for more information.

- maximum send size (code=3, length=2; 2-byte integer) (input). (TCP only.) This option, which must be in the range 1 to 30,000, specifies the length of the longest message the user expects to send on this connection. The information is passed to TCP. If this option is not used, TCP will be able to handle messages at least 1024 bytes long. If the value specified is smaller than a previously specified maximum send size, the new value will be ignored.

- maximum receive size (code=4, length=2; 2-byte integer) (input). (TCP only.) This option, which must be in the range 1 to 30,000, specifies the length of the longest message the user expects to receive on this connection. The information is passed to TCP. If this option is not used, TCP will be able to handle messages at least 1024 bytes long. If the value specified is smaller than a previously specified maximum receive size, the new value will be ignored.

- address option (code=128, length=2; 2-byte integer) (input). (TCP only.) This option specifies the source port address of the connection request. Address values in the range %74057 to %77777 can be used without special capabilities. In privileged programs, values in the range %1 and %74056 can be used. See the paragraph "User-specified Protocol Addressing" at the beginning of this section for more information.

- facilities set name (code=142, length=8, packed array of 8 characters) (input). For access to the X.25 protocol only. This option field is used to associate a facilities set with the virtual circuit to be created over an SVC. This option does not apply to a PVC. This is an optional parameter and defaults to the facilities set name entered while configuring the X.25 network (see *NS3000/V Network Manager Reference Manual, Volume 1*).

- protocol flags (code=144, length=4, 4-byte buffer) (input). This option contains 32 bits of protocol-specific flags. The following flags are currently defined:

  - no address (bit 17, input). (X.25 only.) This flag provides the user with access to the entire X.25 call user data field (16 bytes). This option can be useful for communication with non-HP nodes.

vcdesc (output) — **32-bit integer, by reference.** The returned VC socket descriptor, a number identifying a VC socket belonging to this process through which data can be sent or received. This descriptor can be used in other intrinsics.

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

Discussion

The IPCCONNECT intrinsic is used to establish a VC socket (a virtual circuit) to another process. The calling process must first create a call socket for itself and obtain the destination descriptor of a call socket belonging to the other process.

A successful result means that the connection request has been initiated. The process which requested the connection (via IPCCONNECT) must then call IPCRECV with the VC socket descriptor value in order to complete the connection. (IPCCONNECT is a non-blocking call: the calling process is not blocked pending completion of its request.)

Only the destination descriptor and VC socket descriptor parameters are required (option variable). If a call socket descriptor is not supplied, or if the specified value is -1, a call socket will be created for the purpose of setting up the connection. This socket will be destroyed before completion of the IPCCONNECT call.

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called from split stack mode.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

Table 2-2. IPCCONNECT Protocol Specific Parameters

Parameters	TCP	X.25
flags		
0	Protected connection	n/a
21	Enable/disable checksum	n/a
opt		
2	n/a	Call user data (CUD)
3	Maximum send size	n/a
4	Maximum receive size	n/a
128	TCP source port address	n/a
142	n/a	Facilities set name
144	None defined	Bit 17: access to CUD

X.25 Considerations

IPCCONNECT used over a switched virtual circuit causes the X.25 protocol to send a call request packet to the node and process described by the destination socket. Over a permanent virtual circuit (PVC), a reset packet is sent.

The *opt* parameter CUD field is sent as the CUD field in the call request packet. Based on the setting of the *opt protocol flags* "no address" flag, the user has access to either 12 or 16 bytes in the CUD field.

For communication between HP nodes, the first four bytes of the CUD field are interpreted as an address for incoming call packets (the third and fourth bytes contain the protocol relative address). The X.25 protocol uses this data to find the proper source socket to route the incoming call. This corresponds to the relative address parameter passed when the source socket was created.

Common errors returned by IPCCONNECT in *result* are:

SOCKERR	0	Request completed successfully.
SOCKERR	46	Unable to interpret received path report.
SOCKERR	55	Exceeded protocol module's limit.
SOCKERR	116	Destination unreachable.
SOCKERR	143	Invalid facilities set.
SOCKERR	157	All outgoing switched virtual circuits are busy.
SOCKERR	160	Incompatible with protocol state.
SOCKERR	162	X.25 permanent virtual circuit does not exist.
SOCKERR	163	Permanent virtual circuit already established.

A complete table of SOCKERRs is included in Appendix C.

Cross-System Considerations for TCP

The following are HP 3000 to HP 1000, and HP 3000 to HP 9000 programming considerations for this intrinsic:

Checksumming - TCP checksumming will be enabled for both sides of the connection if it is enabled by either side for HP 3000 to HP 1000 or HP 3000 to HP 9000 connections. Checksumming is always enabled on the HP 9000. On the HP 3000, checksumming can be enabled by setting bit 21. On the HP 3000, this bit can be used to override the checksumming decision made during network transport configuration for this particular process.

Send and receive sizes - The HP 3000 send and receive size range is 1 to 30,000 bytes. The HP 1000 send and receive size range is 1 to 8,000 bytes. The HP 9000 send and receive size range is 1 to 32,767 bytes. Although the ranges are different, you must specify a send size within the correct range for the respective receiving system; otherwise, an error will occur.

Note that the default send and receive sizes are different on different HP systems. On the HP 3000, the default send and receive size is less than or equal to 1024 bytes. On the HP 1000 and HP 9000, the default send and receive size is 100 bytes.

Performs special operations.

Syntax

```
IPCCONTROL (descriptor,request[,wrtdata][,wlen]
           [,readdata][,rlen][,flags][,result])
```

Parameters

descriptor (input)
: **32-bit integer, by value.** Either a call socket descriptor or a VC socket descriptor.

request (input)
: **32-bit integer, by value.** The value supplied indicates what control operation is to be performed.

NOTE

500-level requests are available only to processes running in Privileged Mode.

- 1 = Enable nowait (asynchronous) I/O for the specified call socket or VC socket descriptor. See the paragraph, Asynchronous I/O in this chapter for more information on asynchronous processing.
- 2 = Disable nowait (asynchronous) I/O for the specified call socket or VC socket descriptor; perform waited (blocking) calls only.
- 3 = Change the default timeout (initially 60 seconds) for waited and nowait I/O (receive operations only). The *wrtdata* parameter contains the timeout value in tenths of seconds (16-bit signed integer).
- 9 = Accept a connection request that is in the deferred state. This request is valid only over connection sockets in the connection pending state. If the user wishes to clear, rather than accept the call, then use the reject request (15). The call must be accepted before attempting to send or receive data on the connection. No *readdata* or *wrtdata* parameters are associated with this request.
- 10 = Send a reset packet (X.25 only). This request is valid only over connection sockets. The *wrtdata* parameter (2 bytes) can contain the cause (byte 1) and diagnostic (byte 2) fields to be included in the reset packet sent by the X.25 protocol. The cause field may be overridden by the PDN. If configured as a DTE, the cause will always be 0, irrespective of the value entered. Suggested value for the cause field is 0 (zero), DTE originated. No *readdata* is associated with this request.

- 11 = Send an interrupt packet (X.25 only). This request is valid only over connection sockets. The *wrtdata* parameter can contain 1 byte of user data that will be inserted in the interrupt packet sent by the X.25 protocol.

- 12 = Reason for error or event (X.25 only). This request returns the reason for the NetIPC error or event on an X.25 connection in the *readdata* parameter. The first byte of *readdata* contains the type of packet, the second byte contains the interrupt user data field, and the third and fourth bytes contain the cause and diagnostic fields. This request is valid only over an X.25 connection socket after a communications line error has occurred. Possible cause and diagnostic codes generated by NS X.25 are listed in Appendix B.

  The types of packets returned are:

  - 10 = Clear packet received
  - 11 = Reset packet received
  - 12 = Interrupt packet received
  - 14 = Network shutdown
  - 15 = Restart sent by local network operator
  - 16 = Level 2 failure detected
  - 17 = Restart sent by local protocol module
  - 18 = Restart packet received

  If no event is reported, *readdata* contains zeros. If the error was caused by a clear or restart packet, the connection is lost, and the user must use IPCSHUTDOWN to clear the connection. There is no *wrtdata* associated with this request.

- 13 = Set no activity timeout (X.25 only). This request is only valid on connection sockets. The *wrtdata* parameter contains the timeout value in minutes (16-bit positive integer). If not specified, the default value of zero will be passed to *wrtdata* disabling the timer. After a timeout, IPCSHUTDOWN must be used to remove the connection socket. There is no *readdata* associated with this request.

- 15 = Reject a connection request that is in the deferred state. The socket is automatically deleted after this request.

  For X.25, this request causes the protocol to send a clear packet with the cause field set to zero (DTE originated) and the diagnostic field set to 64. This request is valid only over connection sockets in the connection pending state.

- 256 = Enable nowait receives; disable nowait sends.

- 257 = Enable nowait sends; disable nowait receives.

- 258 = Abort outstanding nowait receives.

- 259 = Enable user-level NetIPC tracing. This request causes NetIPC intrinsic calls (both initiation and completion of I/O requests) to be traced. If tracing is enabled, the *wrtdata* parameter has three tracing related options, described under *wrtdata*.

- 260 = Disable user-level NetIPC tracing.
- 261 = Enable immediate acknowledgment. (TCP only.) Instructs the TCP protocol module to acknowledge received frames immediately. **Note that use of option 261 can degrade performance of the user's process.**
- 262 = Change the timeout for waited and no-wait sends. (Default= timeout disabled.)
- 514 = Return the socket's address in the *readdata* buffer (privileged users only). The *rlen* parameter returns the length of *readdata*. See the table in "Discussion" of this intrinsic for explanations of the values returned in *readdata*.

wrtdata (input)

Record or byte array, by reference. If the request is to change the default timeout, (*request* code 3 or 262) the value in the first two bytes of the *wrtdata* buffer will become the new timeout, in tenths of a second. A zero value indicates an indefinite timeout: a call to IOWAIT will return only when the next I/O request completes. If the request is to enable tracing, (*request* code 259) this parameter may (optionally) contain information in the same format as the *opt* parameter in other intrinsics. Permitted options are:

- code 131--Indicates that the data portion of this parameter contains the trace file name. If omitted, the trace file will be named SOCK####, where #### are four randomly chosen digits, and placed in the caller's group and account.
- code 132--Indicates that the data portion of this parameter contains a 2-byte value representing the number of records allotted to the trace file. If omitted, or if this value is zero, the DEFAULT is 1024 records.
- code 133--Indicates that the data portion of this parameter contains a 2-byte value representing the maximum number of bytes of user data which you wish to trace. If omitted, or if the value is -1, the DEFAULT is 2000 bytes (a zero value means zero bytes). The largest amount of user data which may be traced is 8,192 bytes.

wlen (input)

32-bit integer, by value. Length in bytes of the *wrtdata* parameter.

readdata (output)

Record or byte array, by reference. If *request* enables tracing, the trace file's name is returned in this parameter. If *request* asks for the socket's address, that address is returned here.

rlen (input/output)

32-bit integer, by reference. The maximum number of bytes that you expect to receive in the *readdata* parameter. If *readdata* returns the trace file name, *rlen* will return the length in bytes of this name. If *readdata* returns the socket's address, *rlen* will return the byte length of the address.

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

- flags [31] (input)-- (TCP only.) If NetIPC tracing is enabled in this intrinsic, this flag indicates that Transport Layer protocol activity (headers and internal messages) should also be traced.

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

Discussion

The IPCCONTROL intrinsic is used to perform various special operations on sockets. The intrinsic is "option variable." All requests require the *descriptor* and *request* parameters. The timeout and software interrupt requests also require the *wrtdata* parameter. For tracing and socket address requests, information may be returned in the *readdata* buffer.

Request code 3 is used to set a receive timeout value as specified in *wrtdata* (two bytes). Zero (0) may be used to indicate no timeout. The timeout value should be in tenths of a second. The default value is *60 seconds* with the timeout enabled.

Request code 262 is used to set a send timeout value as specified in *wrtdata* (two bytes). Zero (0) may be used to indicate no timeout. If timeouts are enabled, the timer will expire the number of timeout seconds (as specified in *wrtdata*) after completion of the last send. The default value is timeout NOT enabled. There is only one send timer per connection. It will be running any time there is an outstanding send. That is, if nowait I/O is used, it will run until IOWAIT completes for all sends. For a waited send, the timer will run until the intrinsic completes. If multiple nowait sends are issued, the timer will be restarted for each send initiated and for each IOWAIT completed with sends still outstanding. If a send timer expires before a send completes, the connection must be shutdown.

Request codes 9 and 15 allow the user to accept or reject a connection that is in the deferred-connection state (see IPCRECVCN). If the connection request is accepted, the connection can receive and send data upon the completion of IPCCONTROL. If the connection is rejected, all resources allocated for the connection are returned and the requestor is notified of the rejection.

When requesting the descriptor's address (request code 514), *readdata* has the meanings shown in Table 2-3:

Table 2-3. *readdata* **Meanings**

Descriptor Type	Address Meaning
call socket	port address of socket (for TCP, length=2 bytes)
connection from IPCCONNECT	local port address of connection socket (for TCP, length=2 bytes)
connection from IPCRECVN	remote port address of connection socket in bytes 0 and 1, remote internet address of node in bytes 2 through 5. (6 bytes total length)

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

Table 2-4. IPCCONTROL Protocol Specific Parameters

Parameters	TCP	X.25
request		
10	n/a	Send reset
11	n/a	Send interrupt
12	n/a	Reason for error or event
13	n/a	Set inactivity timeout
261	Enable immediate ack	n/a
flags		
31	Trace transport layer protocol activity	n/a

X.25 Considerations

Common errors returned by IPCCONTROL in *result* are:

```
SOCKERR   0  Request completed successfully.
SOCKERR  59  Socket timeout.
SOCKERR  65  Connection aborted by local protocol module.
SOCKERR  67  Connection failure detected.
SOCKERR 107  Transport is going down.
SOCKERR 160  Incompatible with protocol state.
SOCKERR 168  Restart event occurred on X.25 connection.
```

A complete table of SOCKERRs is included in Appendix C.

Creates a call socket for the calling process.

Syntax

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

Parameters

socketkind (input)
: **32-bit integer, by value.** Indicates the type of socket to be created. The only type which a user process may create is:

- 3 = call socket. Used for sending and receiving connection requests.

protocol (input)
: **32-bit integer, by value.** Indicates the protocol module which the calling process wishes to access. If the value is zero or if this parameter is not specified, the TCP module is chosen by default. The protocols currently available to user processes are:

- 0 = Default protocol. The current default is TCP. The recommended value for programs using IPCNAME and IPCLOOKUP is 0 rather than 4 for TCP.
- 2 = X.25 protocol
- 4 = TCP (Transmission Control Protocol)

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

- flags [0] (input). TCP only. Makes the newly created socket a "protected" socket. A protected socket is one which only a privileged user may create or use.

opt (input)
: **Record or byte array, by reference.** A list of options, with associated information. Refer to "NetIPC Intrinsics/Common Parameters" for more information on the structure of this parameter. The following options are available:

- maximum connection requests queued (option code=6, length=2, 2-byte integer) (input). Used to specify the maximum number of unreceived connections that can be queued to a call socket. The default value is 7.
- address option (option code=128, length=*n*; *n*-byte array) (input). Allows users to specify the socket's protocol relative address rather than having NetIPC allocate an address. The format of this address is defined by the

protocol. For TCP and X.25 protocol access, the address is a 2-byte array. For X.25, you must either specify a protocol relative address, or identify the socket as catch-all. (See the *opt* protocol flags "catch-all socket flag" (bit 2) description.) Address values in the range % 74057 to % 77777 can be used without special capabilities. In privileged programs, values in the range % 1 and % 74056 can be used. See the paragraph "User-specified Protocol Addressing" at the beginning of this section for more information.

- network name (code=140, length=8, packed array of characters) (input) The X.25 network name is the network interface (NI) name defined when the network is configured with NMMGR (see the *NS3000/V Network Manager Reference Manual, Volume 1*). This option is required for X.25 protocol access. This field is left-justified. For unused bytes pad the field with nulls (ASCII zero).

- protocol flags (code=144, length=4, 4-byte buffer).

  - catch-all socket flag (bit 2, input). X.25 protocol access only. This flag identifies the socket as a catch-all socket. Network administrator (NA) capability is required to set this flag. User capability is required to run a program that creates a catch-all socket. The address option (protocol relative address) does not apply to a catch-all socket.

calldesc (output) — **32-bit integer, by reference.** Call socket descriptor. The socket descriptor which identifies the created socket.

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

Discussion

The IPCCREATE intrinsic creates a call socket, returning a call socket descriptor. A call socket descriptor is an identifying number which may be used in other NetIPC intrinsic calls. (Internally, a call socket descriptor is an AFT, an Available File Table entry number; the descriptor is stored in the Available File Table.) A process may own a maximum of 64 (call and VC) sockets. If a socket has been given away (via the IPCGIVE intrinsic), it is included in this total until another process takes it (via IPCGET).

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

IPCCREATE may not be called in split stack mode. IPCCREATE runs in waited mode. It does not return until the request is completed.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

Table 2-5. IPCCREATE Protocol Specific Parameters

Parameters	TCP	X.25
flags		
0	Protected socket	n/a
opt		
140	n/a	NI name required
144	None defined	Bit 2: catch-all socket flag

X.25 Considerations

For direct access to X.25, the *protocol* parameter must be 2 (X.25). The *opt* parameter network name must include the X.25 network NI name.

The *opt* parameter address option (code 128) is used to contain the protocol relative address of the source socket.

X.25 compares the protocol relative address contained in an incoming call (in the CUD field) to the protocol relative addresses assigned to all X.25 call sockets at the source sockets' destination. If the protocol relative address of the source socket matches the incoming call's address (CUD) the call is routed to that socket. If no match is found, the incoming call is routed to the catch-all socket if one has been defined. If the CUD address does not match any of the call sockets and no catch-all socket has been defined, the incoming call is cleared. The cause field of the clear packet is set to 0 and the diagnostic is 64.

The catch-all socket can be defined by setting the *opt protocol flags* catch-all socket flag. Only one catch-all socket can be defined per directly-connected network.

The catch-all socket and address option (protocol relative address) only apply to switched virtual circuits (SVCs).

Common errors returned by IPCCREATE in *result* are:

```
SOCKERR   0  Successful completion.
SOCKERR   4  Transport has not been initialized.
SOCKERR   9  Protocol is not active.
SOCKERR  55  Exceeded protocol module's limit.
SOCKERR 106  Address currently in use by another socket.
SOCKERR 107  Transport is going down.
SOCKERR 153  Socket is already in use.
```

A complete table of SOCKERRs is included in Appendix C.

TCP

For TCP access, only the *socketkind* and *calldesc* parameters are required. (In SPL terms, the intrinsic is option-variable.)

Creates a destination descriptor.

Syntax

```
IPCDEST (socketkind[,location][,locationlen],protocol,
         protoaddr,protolen[,flags][,opt],destdesc[,result])
```

Parameters

socketkind (input)
: **32-bit integer, by value.** Defines the type of socket. The only type which a user process may create is:

- 3 = call socket.

location (input)
: **Character array, by reference.** The name of the node (optionally *node.domain.organization*) on which the destination socket is to be created. If this parameter is omitted, the local node is assumed.

locationlen (input)
: **32-bit integer, by value.** The length in bytes of the destination node name. Zero indicates that no location was given (that is, the node is local). Maximum (for a fully qualified name) is 50.

protocol (input)
: **32-bit integer, by value.** Defines the protocol access to be used by the user processes. The protocols currently available to user processes are:

- 2 = X.25 protocol
- 4 = TCP

protoaddr (input)
: **Byte array, by reference.** Protocol relative address (remote address) with which the socket will be associated. The format of this address, defined by the protocol, is a 2-byte array (16 bits). Nonprivileged programs must use addresses in the range %74057 to %77777. For X.25 access to level 3, this address is not included in the CUD field of an X.25 call packet. (See the discussion of IPCCONNECT for the parameters providing access to the CUD.)

protolen (input)
: **32-bit integer, by value.** The length in bytes of the protocol address.

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

opt
: **Record or byte array, by reference.** A list of options, with associated information. No options are currently defined.

destdesc (output)	**32-bit integer, by reference.** Destination descriptor. Describes the location of the named call socket. May be used in subsequent NetIPC calls (IPCCONNECT, etc.).
result (output)	**32-bit integer, by reference.** The error code returned; zero if no error.

Discussion

The IPCDEST intrinsic creates a destination descriptor for the purpose of sending messages to another process. For TCP access, you can use this intrinsic as alternative to using IPCNAME and IPCLOOKUP to create a destination descriptor. IPCDEST must be used for X.25 protocol access.

Using IPCDEST enables you to specify a particular protocol relative address to be associated with the destination descriptor. See Example 2 in Section 3 of this manual for an example program that uses IPCDEST.

Nonprivileged user processes must use addresses in the range % 74057 to % 77777.

This intrinsic is option variable. The required parameters are: *socketkind*, *protocol*, *protoaddr*, *protolen*, and *destdesc*. Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode. IPCDEST runs in waited mode. It does not return until the request is completed.

Protocol-Specific Considerations

X.25 Considerations

IPCDEST is used to create a destination descriptor for X.25 direct access. The *protoaddr* parameter is only used with switched virtual circuits (SVCs).

Cross-System Considerations for TCP

The following are HP 3000 to HP 1000 and HP 3000 to HP 9000 programming considerations for this intrinsic:

TCP protocol address - The recommended range of TCP addresses for user applications is from 30767 to 32767 decimal (%74057 to %77777) for the HP 3000, HP 1000, and HP 9000.

Returns the NetIPC error message corresponding to a given error code.

Syntax

```
IPCERRMSG (ipcerr,msg,len,result)
```

Parameters

ipcerr (input)	**32-bit integer, by value.** A valid NetIPC error code.
msg (output)	**Character array, by reference.** The NetIPC error message corresponding to the given error code. This array must be at least 80 bytes in length.
len (output)	**32-bit integer, by reference.** The length (in bytes) of the error message. The maximum is 80 bytes.
result (output)	**32-bit integer, by reference.** The error code returned for this intrinsic call; zero if no error.

Discussion

The IPCERRMSG intrinsic returns the NetIPC error message corresponding to a given error code. It also gives the length of the message. All parameters are required.

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed because of a user error.
- CCG--Failed because of an internal error (e.g. unable to open the message catalog, GenMessage failure, etc.).

This intrinsic may not be called in split stack mode.

IPCGET

Receives a (call socket or VC socket) descriptor which has been given away by another process.

Syntax

```
IPCGET (givename,nlen,flags,descriptor,result)
```

Parameters

givename (input)	**Character array, by reference.** The temporary name assigned to the socket when it was given away. It is up to 16 characters long.
nlen (input)	**32-bit integer, by value.** The length in bytes of the specified name.
flags	**32 bits, by reference.** A bit representation of various options. No flags are currently defined for this intrinsic.
descriptor (output)	**32-bit integer, by reference.** The descriptor that was given away via the IPCGIVE command. May be a call socket descriptor or a VC socket descriptor.
result (output)	**32-bit integer, by reference.** The error code returned; zero if no error.

Discussion

The IPCGET intrinsic allows a process to obtain a call or VC socket descriptor which has been relinquished by another process through the IPCGIVE intrinsic. A temporary name identifies the socket for the process which wishes to acquire it. All the parameters are required.

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

Gives away a (call socket or VC socket) descriptor, making it available to other processes.

Syntax

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

Parameters

descriptor (input)
: **32-bit integer, by value.** The descriptor to be given away. May be a call socket or VC socket descriptor.

givename (input/output)
: **Character array, by reference.** A name which will be temporarily assigned to the specified socket. The process which obtains the socket must request it by this name. If the *nlen* (name length) parameter is zero, an 8-character name is randomly assigned and returned in the *givename* parameter. If the name is supplied by the user, it must be no longer than 16 characters.

nlen (input)
: **32-bit integer, by value.** Length in bytes of the specified name. If the value is zero, the NetIPC facility will assign the name.

flags
: **32 bits, by reference.** A bit representation of various options. No flags are currently defined for this intrinsic.

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

Discussion

A process can invoke IPCGIVE to "give" a call or VC socket descriptor that it owns to another process at the same node. For example, Process A at node X can give a VC socket descriptor to Process B, also at node X, so that Process B may use a connection Process A has previously established with process C at node Z. Because Process B was "given" the endpoint of a previously established connection, it does not need to create its own call socket and engage in the NetIPC connection dialogue in order to exchange with Process C.

All the parameters are required.

When it is given away, a socket is assigned a new, temporary name. This name is either specified by the user or assigned by the NetIPC facility. It continues to exist only until the socket is obtained by another process or destroyed. The other process uses this name in a call to IPCGET, not IPCLOOKUP. However, the syntax of the name is the same as it is for other intrinsics permitting socket name parameters. Therefore it is possible to use a socket's "well-known" name - a name bound to the socket and known to other processes - in the IPCGIVE and IPCGET intrinsics.

Once a process has given away a socket, it no longer has access to the call socket/VC socket descriptor specified. If a process expires after giving away a socket, before another process has obtained it, the socket or VC socket will be destroyed.

Other processes may continue to send data to a socket after it has been given away. It is the responsibility of this process to notify other processes that a socket has been given away and to tell them the name by which they can acquire it.

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

IPCLOOKUP

Obtains a destination descriptor for a named call socket. Used with TCP access only.

Syntax

```
IPCLOOKUP (socketname,nlen[,location][,loclen][,flags],
           destdesc[,protocol][,socketkind][,result])
```

Parameters

socketname (input)
: **Character array, by reference.** The name of the socket.

nlen (input)
: **32-bit integer, by value.** The length in bytes of the specified socket name. Maximum is 16.

location (input)
: **Character array, by reference.** An environment id or node name indicating where the socket registry search is to take place. The domain and organization names which fully qualify the node/environment designation are optional. If no location is specified, the local socket registry is searched.

loclen (input)
: **32-bit integer, by value.** The length in bytes of the *location* parameter. A zero value indicates that the socket registry search is to take place on the local node.

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

- flags [0] (input). Causes the destination descriptor to be "protected." A protected destination descriptor is one which only privileged users may create or use.

destdesc (output)
: **32-bit integer, by reference.** The returned destination descriptor, which the calling process may use to access the named socket as a destination. This descriptor is required by the IPCCONNECT intrinsic.

protocol (output)
: **32-bit integer, by reference.** A number identifying the protocol module with which the socket is associated: 4 = TCP.

socketkind (output)
: **32-bit integer, by reference.** A number which identifies the socket's type: 3 = call.

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

Discussion

The IPCLOOKUP intrinsic is used to gain access to a named socket. When supplied with the socket's name, it returns a destination descriptor which the calling process can use in order to send messages to that socket. It is important to synchronize the naming and lookup of sockets so that the naming occurs before the lookup. If these two events are occurring concurrently, you can repeat the IPCLOOKUP call, checking the *result* parameter after each call, until the call is successful. If the *result* value is 37 ("NAME NOT FOUND"), the socket has not yet been given the name. The following Pascal program fragment illustrates this idea:

```
socketname := 'RAINBOW';
location := 'SOMEWHERE';
result := 0;
count:=0;
repeat
 IPCLOOKUP (socketname,7,location, 9, ,destdesc,protocol,socketkind,result)
count:=count+1;
until (result <> 37) or (count >= maxcount)
if result <> 0 then ERRORPROCEDURE;
```

The only required parameters in the IPCLOOKUP intrinsic are *socketname*, *nlen*, and *destdesc* (option variable). Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

IPCNAME

Associates a name with a call socket descriptor. Used with TCP access only.

Syntax

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

Parameters

calldesc (input)
: **32-bit integer, by value.** The call socket descriptor to be named.

socketname (input/output)
: **Character array, by reference.** The name (maximum 16 characters) to be assigned to the socket. If the *nlen* (name length) parameter is zero, an 8-character name is randomly assigned and returned in the *givename* parameter. If the name is supplied by the user, it must be no longer than 16 characters.

nlen (input)
: **32-bit integer, by value.** The length in bytes of the specified socket name. Maximum is 16.

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

Discussion

The IPCNAME intrinsic allows a user to bind a name to a call socket. Using the IPCLOOKUP intrinsic, another process can obtain access to the socket by means of its name. A single call socket on an HP 3000 can have a maximum of 4 names. (VC sockets cannot be named.) If the specified name length is zero, an 8-character name will be randomly generated and returned in the *socketname* parameter. When the socket is destroyed, the name will be removed from the socket registry.

All parameters are required. Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

IPCNAMERASE

Deletes a name associated with a call socket descriptor. Used with TCP access only.

Syntax

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

Parameters

socketname (input)	**Character array, by reference.** The socket name, bound to a socket, which is to be removed.
nlen (input)	**32-bit integer, by value.** The length in bytes of the specified socket name. Maximum is 16.
result (output)	**32-bit integer, by reference.** The error code returned; zero if no error.

Discussion

If a socket has been named with the IPCNAME intrinsic, the owner of the socket may remove the name by means of the IPCNAMERASE intrinsic. The owner is the process which created the socket or, if the socket has been given away, the process which has acquired it.

All the parameters are required. Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

Receives a response to a connection request, thereby establishing a connection, or receives data on an already-established connection.

Syntax

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

Parameters

vcdesc (input)
: **32-bit integer, by value.** The VC socket descriptor, a number identifying the VC socket belonging to this process through which the data will be received.

data (output)
: **Record or byte array, by reference.** A buffer to hold the received data or a list of data descriptors (maximum two) indicating where the data are to be distributed.

dlen (input/output)
: **32-bit integer, by reference.** Gives the maximum number of bytes you are willing to receive. For a response to a connection request, this value may be 0 (or the parameter may be omitted). For actual data on an established connection, the value must be between 1 and 30,000. The returned value indicates how many bytes were actually received.

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

- flags [16]--no output (input). (TCP only.) If nowait I/O is used and this bit is set, the *flags* parameter will not be updated upon completion of this IPCRECV. This allows a calling procedure to have a local *flags* parameter and still complete before the IPCRECV completes. This flag has no effect if waited I/O is being used.

- flags [26]--more data (output). Indicates that there may be more data to be received after completion of this IPCRECV.

  For TCP, this bit will always be set when normal, non-urgent data has been received because TCP sends data in stream mode, with no end-of-data indication. However, if urgent data has been received, and no more is pending, this bit will be set to 0.

  For X.25, the "more data" flag indicates that the data returned is not the complete message. The amount of data specified in *dlen* has been moved into *data*. The following part of the message will be returned in the next call to IPCRECV, unless the *destroy flag* (29) was set.

- flags [29]--destroy data (input). If set, this flag causes delivered data that exceeds the amount allowed by the specified *dlen* or byte count (for vectored data) to be discarded. Use this flag to remove data that may have arrived at your node (and queued in the NetIPC buffer) that you do not want the process to receive. For example, you may want to reuse a previously established connection, but would not want to receive data left over from a previous transmission.

Note that in TCP stream mode, there is no mechanism to verify that data has been discarded.

- flags [30]--preview (input). This flag allows the calling process to preview the data - that is, to read the data without removing them from the queue of data to the receiving socket.
- flags [31]--vectored (input). This flag indicates that the received data are to be distributed to the addresses given in the *data* parameter.

opt (input/output)

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

- data offset (code=8, length=2; 2-byte integer) (input/output). This option specifies an offset in bytes from the *data* parameter's address. The received data are to be written into memory beginning at this location. Do not use this option with vectored data.
- protocol flags (code=144, length=4; 4-byte buffer) (output). This option contains 32 bits of protocol-specific flags. The following flags are currently defined:
  - end-to-end acknowledgment (bit 18, output). (X.25 only.) This flag indicates that the D bit is set in the X.25 packet associated with this call.
  - qualifier bit (bit 19, output). (X.25 only.) This flag indicates that the Q bit is set in the X.25 packet associated with this call.
  - urgent data (bit 27, output). (TCP only.) This flag indicates that urgent data has been received on an established connection. This flag is not output if flags[16] (no output flag) is set when IPCRECV is called in nowait mode.

NOTE

If using nowait I/O and *opt* array options that generate output, the array must remain intact until after IOWAIT completes. Otherwise, the array area will be overwritten or (if the area has been deleted from the stack) an error will occur.

result (output)

32-bit integer, by reference. The error code returned; zero if no error.

NOTE

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

Discussion

The IPCRECV intrinsic serves two purposes: (1) to receive a response to a connection request, thereby establishing a connection, and (2) to receive user data on an established connection.

NOTE

In the first case the VC socket descriptor is the only required parameter; in the second, the VC socket descriptor, data, and data length parameters are required. The brackets in the syntax diagram represent the first case.

In receiving a response to a connection request, the IPCRECV intrinsic returns nothing in the *data* buffer. A *result* value of zero indicates a successful connection establishment. The *result* parameter will indicate an error if the destination rejected the request. In that case you must still call IPCSHUTDOWN with the returned VC socket descriptor value to shutdown the connection.

Successful completion of an IPCRECV request on an established connection (*result* code zero) means that some amount of data was received: the amount requested or the amount transmitted, whichever is smaller. It does not mean that you received all the data you asked for.

Completion of IPCRECV (while using waited I/O) with a non-zero *result* can mean that a fatal error occurred or for TCP access, a gracful release request has been received (indicated by SOCKERR 102). If nowait I/O is being used, IPCCHECK must be called to indicate a fatal error or graceful release.

Unless the intrinsic is called in nowait mode, the process is blocked until some data arrive or a timeout occurs. In nowait mode, the addresses of the *data* and *flags* parameters are retained by NetIPC until needed. The input value of *flags* is retained and updated (with the "more data" flag off or on) when IOWAIT completes. The *data* parameter (or the vectored location) will then contain the data received. Only one nowait receive may be outstanding on a single connection.

The returned *dlen* parameter (or the IOWAIT *tcount* parameter in the case of a nowait request) shows how many bytes of data were actually received in the *data* parameter. This amount may be different from what you requested. If you did not receive all the data you want, you can obtain the additional data in a subsequent IPCRECV call. For more information, see the discussion of "Sending and Receiving Data Over a Connection" earlier in this manual and the programmatic examples in Section 3.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

TABLE 2-6. IPCRECV Protocol Specific Parameters

Parameters	TCP	X.25
flags		
16	No output flag	n/a
26	More data	More data
opt		
144	Bit 27: urgent data	Bit 18: state of D bit in X.25 packets Bit 19: state of Q bit in X.25 packets

X.25 Considerations

A single IPCRECV call returns data for one message only. If the "more data" flag is set, the complete message has not been received. The remaining part of the message can be received by subsequent calls to IPCRECV, unless the destroy flag (29) is set. If the destroy flag is set, the remaining part of the message is destroyed. The end of message is indicated by the reset of the "more data" flag.

If an interrupt packet is received in the middle of a data packet stream, IPCRECV returns no data. The *result* parameter indicates that an event has occurred. The interrupt user data field can be retrieved by calling IPCCONTROL, request 12 (reason for error or event). The next call to IPCRECV returns the whole data message.

If a reset packet is received in the middle of a data packet stream, all previously received packets are discarded. IPCRECV returns no data. The *result* parameter indicates a reset has occurred. Use the IPCCONTROL request 12 (reason for error or event) to retrieve the cause and diagnostic fields for the reset.

Common errors returned by IPCRECV in *result* are:

```
SOCKERR   0  Request completed successfully.
SOCKERR  59  Socket timeout.
SOCKERR  65  Connection aborted by local protocol module.
SOCKERR  67  Connection failure detected.
SOCKERR 107  Transport is going down.
SOCKERR 117  Attempt to establish connection failed.
SOCKERR 146  Event reset.
SOCKERR 156  Event interrupt.
SOCKERR 158  Connection request rejected by remote.
SOCKERR 168  Restart event occurred on X.25 connection.
```

A complete table of SOCKERRs is included in Appendix C.

TCP

The urgent data bit indicates that urgent data has been received. Table 2-3 demonstrates the meaning of urgent data and more data. Use these bits in combination to determine the status of data received.

Table 2-7. TCP Urgent and More Data Bit Combinations

Urgent	More Data	Meaning
0	0	Should never happen. (The receipt of normal data in stream mode causes "more data" to be set.)
0	1	Normal receive, no urgent data.
1	0	Urgent data received, no more urgent data.
1	1	Urgent data received and more is pending.

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may be called in split stack mode.

Cross-System Considerations for TCP

The following are HP 3000 to HP 1000, and HP 3000 to HP 9000 programming considerations for this intrinsic:

Receive size (*dlen* parameter) - Range for the HP 3000 is 1 to 30,000 bytes. Range for the HP 1000 is 1 to 8000 bytes. Range for the HP 9000 is 1 to 32,767 bytes. Although the ranges are different, you must specify a send size within the correct range for the respective receiving system; otherwise, an error will occur.

Note that the default send and receive sizes are different on different HP systems. On the HP 3000, the default send and receive size is less than or equal to 1024 bytes. On the HP 1000 and HP 9000, the default send and receive size is 100 bytes.

Data wait flag - The HP 1000 and HP 9000 IPCRecv call supports a "DATA_WAIT" flag. This flag, when set, specifies that the call will not complete until the amount of data specified by the *dlen* parameter has been received. This flag is not available on the HP 3000, meaning that the call may complete before all the data is received. However, the HP 3000 IPCRECV supports other flags such as the "more data" and "destroy data" flags.

Receives a connection request on a call socket.

Syntax

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

Parameters

calldesc (input)
: **32-bit integer, by value.** Call socket descriptor. The socket descriptor for a call socket belonging to this process.

vcdesc (output)
: **32-bit integer, by reference.** The returned VC socket descriptor, a number identifying a VC socket belonging to this process through which data can be sent or received. This descriptor can be used in other intrinsics.

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

- flags [0]--protected (input). (TCP only.) Ensures that the connection will be "protected" (privileged users only).
- flags [18]--defer (input). Causes the reply to the connection request to be deferred. The intrinsic will complete when a connection request is received, but the virtual circuit will not be established. The IPCCONTROL intrinsic can be used later to accept or reject the connection.
- flags [21]--checksum (input). (TCP only.) Enables checksum on the Transmission Control Protocol (TCP) connection for error checking. Checksum may also be set by the corresponding IPCCONNECT call. If either side specifies "checksum enabled" then the connection will be checksummed. TCP checksum may be enabled globally, over all connections, when configuring the Network Transport. See the *NS3000/V Network Manager Reference Manual, Volume I* for details on Network Transport configuration. Checksum enabled by either IPCCONNECT or TCP (remote or local) configuration overrides a 0 setting (checksum disenabled) for this flag. Checksum error checking is handled at the link level and is not normally required at the user level. Enabling checksum may reduce network performance. Recommended value: 0.
- flags [25]--discarded CUD (output). For X.25 protocol access. Indicates that call user data (CUD) was present, but that the data had to be discarded or truncated. If the call user data option (code=5) is not specified the call user data is discarded. If the CUD buffer is not long enough to contain the data, this flag is set and the data is truncated.

opt
(input/output)

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

- maximum send size (code=3, length=2; 2-byte integer) (input). (TCP only.) This option, which must be in the range 1 to 30,000, specifies the length of the longest message the user expects to send on this connection. The information is passed to TCP. If this option is not used, TCP will be able to handle messages at least 1024 bytes long. If the value specified is smaller than a previously specified maximum send size, the new value will be ignored.

- maximum receive size (code=4, length=2; 2-byte integer) (input). (TCP only.) This option, which must be in the range 1 to 30,000, specifies the length of the longest message the user expects to receive on this connection. The information is passed to TCP. If this option is not used, TCP will be able to handle messages at least 1024 bytes long. If the value specified is smaller than a previously specified maximum receive size, the new value will be ignored.

- call user data (code=5, length=n, n bytes) (output). (X.25 only.) This option provides a buffer for the return of the call user data (CUD) field from an X.25 packet. If call user data is present, but this option is not supplied, the discarded flag [25] is set. If the buffer is not long enough to contain the data, the data is truncated and the discarded flag is set.

- calling node address (code=141, length=8; 8-byte array) (output). An output parameter that is used to contain the address of the requestor.

  For TCP, the first two bytes of the array contain the remote socket's port address and the next four bytes contain the remote node's internet protocol address. The remaining bytes are unused.

  For X.25 protocol access, the X.25 address of the calling node is returned in this field. The format of the record is equivalent to 16 nibbles (or BCD digits) in which the first nibble is the address length (ranging from 0 to 15), and the following 15 nibbles contains the calling address. The calling node address is not available if the call originated from a PAD.

  You can use READOPT to obtain the output of this parameter.

- protocol flags (code=144, length=4; 4-byte buffer) (output). This option contains 32 bits of protocol-specific flags. The following flags are currently defined:

  - request from PAD (bit 14, output). (X.25 only.) This flag indicates that connection request is coming from a PAD as opposed to a connection coming from a host.

  - calling node address available (bit 16, output). (X.25 only.) This flag indicates that the calling node X.25 address was present.

NOTE

If using nowait I/O and *opt* array options that generate output, the array must remain intact until after IOWAIT completes. Otherwise, the array area will be overwritten or (if the area has been deleted from the stack) an error will occur.

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

NOTE

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

Discussion

The IPCRECVCN intrinsic allows a process to receive a connection request and establish a connection (virtual circuit). The connection is identified by the returned VC socket descriptor. The calling process can then employ the IPCSEND and IPCRECV intrinsics to send and receive data on the connection. A maximum of 7 unreceived connection requests may be queued to a call socket.

If the calling process sets the defer reply to connection request flag (*flags* [18]), this intrinsic will complete when a connection request is received, but the virtual circuit will not be established. The calling process must use IPCCONTROL to either accept or reject the request. This feature is useful if an application must defer replying to the connection request and then, depending upon the identity of the requestor, decide to reject or accept the request. The identity of the requestor can be determined by using either the call node address option or IPCCONTROL to return the remote port and internet protocol addresses.

If this intrinsic is called in nowait mode, the data structures for the connection are created when the call to IOWAIT completes. They are not created with the initial call to IPCRECVCN. Therefore the address of the VC socket descriptor parameter is retained by NetIPC, and the descriptor's value is returned to that location when IOWAIT completes. The VC socket descriptor, any flags, and the *opt* parameter must all be global to both the IPCRECVCN and the IOWAIT intrinsic calls. NetIPC also retains the *flags* parameter.

The only required parameters are the call socket descriptor and VC socket descriptor (option variable).

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

Condition codes returned by the call to IOWAIT are:

- CCE--Succeeded.
- CCL--Failure in NetIPC (e.g. resource problems, VC socket descriptor out of bounds) or protocol module. In the event of a NetIPC failure the connection request will still be pending, allowing the user to correct the problem and issue another call to IPCRECVCN.
- CCG--Connection established but a noncritical error (e.g. flags parameter out of bounds) occurred.

The IPCRECVCN intrinsic may not be called in split stack mode.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

Table 2-8. IPCRECVCN Protocol Specific Parameters

Parameters	TCP	X.25
flags		
0	Protected connection	n/a
21	Enable checksum	n/a
25	n/a	Discarded CUD flag
opt		
3	Maximum send size	n/a
4	Maximum receive size	n/a
5	n/a	Received CUD
141	Calling node's IP address	Calling node's X.25 address
144	n/a	Bit 14: PAD Bit 16: calling node address available flag

X.25 Considerations

IPCRECVCN is used with switched virtual circuits (SVCs) only.

The call user data field returned in the *opt* parameter (code=5) is used by X.25 as follows. The first four bytes of the call user data field is used to determine the destination call (source) socket. The incoming call is sent to the call socket whose relative protocol address matches the first four bytes of the call user data. See the discussion for IPCCREATE for more information on protocol relative addresses.

Call acceptance can be affected by the X.25 configuration of the local user group (LUG) facility which can limit access to a node by specifying which remote X.25 addresses are allowed to communicate with the node. See the *NS3000/V Network Manager Reference Manual, Volume I* for more information about the LUG facility.

Common errors returned by IPCRECVCN in *result* are:

```
SOCKERR   0  Request completed successfully.
SOCKERR  59  Socket timeout.
SOCKERR 107  Transport is going down.
```

A complete table of SOCKERRs is included in Appendix C.

TCP

The calling process may also specify whether checksumming is to be employed by the protocol modules (i.e. TCP) that support it. For TCP, checksumming is usually disabled unless it is included by the remote protocol module or if the TCP checksumming flag (*flags* [21]) is set. When checksumming is enabled, performance is usually degraded because of increased overhead.

Cross-System Considerations for TCP

The following are HP 3000 to HP 1000 and HP 3000 to HP 9000 programming considerations for this intrinsic:

Checksumming - TCP checksumming will be enabled for both sides of the connection if it is enabled by either side for HP 3000 to HP 1000 or HP 3000 to HP 9000 connections. Checksumming is always enabled on the HP 9000. On the HP 3000, checksumming can be enabled by setting bit 21. On the HP 3000, enabling checksumming can be used to override the checksumming decision made during network transport configuration for this particular process.

Receive size (*dlen* parameter) - Range for the HP 3000 is 1 to 30,000 bytes. Range for the HP 1000 is 1 to 8000 bytes. Range for the HP 9000 is 1 to 32,767 bytes. Although the ranges are different, you must specify a send size within the correct range for the respective receiving system; otherwise, an error will occur.

Note that the default send and receive sizes are different on different HP systems. On the HP 3000, the default send and receive size is less than or equal to 1024 bytes. On the HP 1000 and HP 9000, the default send and receive size is 100 bytes.

Sends data on a connection.

Syntax

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

Parameters

vcdesc (input)
: **32-bit integer, by value.** The VC socket descriptor, a number identifying the VC socket belonging to this process through which the data will be sent.

data (input)
: **Record or byte array, by reference.** Contains the data to be sent or a list of data descriptors (maximum two) indicating the locations from which the data will be gathered. Flags [31] is set if data descriptors are used.

dlen (input)
: **32-bit integer, by value.** The byte length of the *data* parameter: that is, the amount of actual data (range is 1 to 30,000) or the length of the data descriptors (8 or 16 bytes).

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

- flags [31]--vectored (input). Indicates that the data to be sent are to be gathered from the addresses given in the *data* parameter. (The parameter will not contain actual data.)

opt (input)
: **Record or byte array, by reference.** A list of options, with associated information. Refer to "NetIPC Intrinsics/Common Parameters" for more information on the structure of this parameter. The following options are defined:

- data offset (code=8, length=2; 2-byte integer) (input). An offset in bytes from the *data* parameter's address indicating the actual beginning of the data. HP recommends that you do not use data offset if data descriptors are used to point to another location from which data should be obtained.
- protocol flags (code=144, length=4; 4-byte buffer) (input). This option contains 32 bits of protocol-specific flags. The following flags are currently defined:
  - end-to-end acknowledgment (bit 18, input). (X.25 only.) D bit will be set in the last X.25 data packet corresponding to this message. When this flag is set, IPCSEND waits to complete until acknowlegement from the remote that the complete message has

been received. When the connection is between two HP3000's running NS X.25 3000/V, the acknowlesgement is made when IPCRECV is called on the remote side.

- qualifier bit (bit 19, input). (X.25 only.) This flag indicates to X.25 to set the Q bit in the packets that contain this message.
- urgent data (bit 27, input). (TCP only.) If set, this bit will cause the data sent to be marked *urgent*.

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

NOTE

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

Discussion

The IPCSEND intrinsic is used to send data on a connection. The only required parameters are *vcdesc*, *data*, and *dlen* (option variable).

A set of addresses in the *data* parameter allows vectored data to be gathered from multiple locations.

The value specified by the data offset option is relative to the data array. If data descriptors are used, specifying this option will cause a portion of the descriptor to be passed over (the offset is NOT applied to the pointer in the descriptor). This may lead to unexpected results.

If this intrinsic is called in nowait mode, the address of the data is passed to the protocol module being accessed. The contents of the data buffer will have been read when IOWAIT completes. As many as 7 nowait sends may be outstanding on a connection.

Condition codes returned by IPCSEND and IOWAIT are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned.

Split stack calls are permitted.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

Table 2-9. IPCSEND Protocol Specific Parameters

Parameters	TCP	X.25
opt		
144	Bit 27: urgent data	Bit 18: state of D bit in X.25 packets Bit 19: state of Q bit in X.25 packets

X.25 Considerations

Setting the Q bit flag causes X.25 to set the Q bit (qualifier bit) in X.25 data packets.

Setting the D bit flag causes X.25 to specify end-to-end acknowledgment of data packets. IPCSEND does not complete until it receives acknowledgment that the entire message has been received.

Common errors returned by IPCSHUTDOWN in *result* are:

```
SOCKERR   0  Request completed successfully.
SOCKERR  50  Invalid data length.
SOCKERR  65  Connection aborted by local protocol module.
SOCKERR  67  Connection failure detected.
SOCKERR 107  Transport is going down.
SOCKERR 131  Protocol module does not have sufficient resources.
SOCKERR 146  Event reset.
SOCKERR 156  Event interrupt.
SOCKERR 159  Invalid X.25 D-bit setting.
SOCKERR 160  Incompatible with protocol state.
SOCKERR 168  Restart event occurred on X.25 connection.
```

A complete table of SOCKERRs is included in Appendix C.

TCP

The urgent data bit of the protocol flags option (*opt* parameter) is used to inform TCP that the data to be sent should be marked *urgent*. This will not cause the data to be delivered out of band, and the receiver of this data will not know of urgent data that is pending until a receive is posted. See IPCRECV for more information.

Cross-System Considerations for TCP

There are no differences that affect cross-system operations. Note that the *urgent data* bit is not supported on the HP 1000; however, if this bit is set by the HP 3000 program, it will be ignored by the receiving process on the HP 1000.

Send size (*dlen* parameter) - Range for the HP 3000 is 1 to 30,000 bytes. Range for the HP 1000 is 1 to 8,000 bytes. Range for the HP 9000 is 1 to 32,767 bytes. Although the ranges are different, you must specify a send size within the correct range for the respective receiving system; otherwise, an error will occur.

Note that the default send and receive sizes are different on different HP systems. On the HP 3000, the default send and receive size is less than or equal to 1,024 bytes. On the HP 1000 and HP 9000, the default send and receive size is 100 bytes.

Releases a descriptor and any resources associated with it.

Syntax

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

Parameters

descriptor (input)
: **32-bit integer, by value.** The socket to be released. May be a call socket, destination, or VC socket descriptor. Privileged Mode capability is required to release destination descriptors created in privileged mode.

flags
: **32 bits, by reference.** A bit representation of various options. The following flag is defined:

  flags[17]--graceful release of connection. (TCP only.) This option is not supported for access to the X.25 protocol.

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

  - reason code (code=143, length=2) (input). (X.25 only.) This option allows you to include cause and diagnostic values in the X.25 clear packets when a connection is closed down. The first byte contains the cause and the second byte contains the diagnostic code. A list of cause and diagnostic codes used with NS X.25 protocol access is contained in Appendix B. If DTE originated, the cause code will always be zero.

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

Discussion

The IPCSHUTDOWN intrinsic permits you to close a call socket or release a connection. The *descriptor* is the only required parameter (option variable).

IPCSHUTDOWN can be called to release a call socket descriptor, a destination descriptor, or a VC socket descriptor. Since system resources are used up as long as call sockets and destination sockets exist, you may want to release them whenever they are no longer needed.

The call socket is needed as long as a process is expecting to receive a connection request for that socket. A process which receives a connection request can release the call sockets any time after the connection request is received via IPCRECVCN, as long as no other connection requests are expected for that call socket. For more information on IPCSHUTDOWN, refer to "Shutting Down Sockets and Connections" in Section 1.

Condition codes returned by this intrinsic are:

- CCE--Succeeded.
- CCL--Failed.
- CCG--Not returned by this intrinsic.

This intrinsic may not be called in split stack mode.

Protocol-Specific Considerations

The following table outlines parameters that are specific to the particular protocol you are accessing.

Table 2-10. IPCSHUTDOWN Protocol Specific Parameters

Parameters	TCP	X.25
flags		
17	Graceful release of connection	n/a
opt		
143	n/a	Reason code (SVCs only)

X.25 Considerations

Shutting down an X.25 connection causes a clear packet to be sent by X.25 over an SVC, or a reset packet over a PVC, unless the virtual circuit is already cleared. You can specify the cause and diagnostic fields in the *opt* parameter (code=143) that will be included in the clear packet over an SVC. Over a public data network (PDN), the cause may not be transmitted to the remote node.

When used with direct access to level 3, the intrinsic IPCSHUTDOWN can only be called in waited mode. The intrinsic will not return until the request is completed. Thus, if a *vcdesc* is specified in IPCSHUTDOWN the reception of the X.25 clear confirmation packet will signal the successful completion of the call.

X.25 direct access to level 3 does not support the *graceful release* bit. As a suggestion, to ensure that no data packets are lost before the clear packet is sent, the D bit option could be set in the last IPCSEND. This would assure end-to-end acknowledgment of this message before issuing the IPCSHUTDOWN to clear the virtual circuit.

Common errors returned by IPCSHUTDOWN in *result* are:

```
SOCKERR   0  Request completed successfully.
SOCKERR  54  Invalid call socket descriptor.
SOCKERR  66  Invalid connection descriptor.
```

A complete table of SOCKERRs is included in Appendix C.

TCP

If graceful release is specified and supported by the remote process, the requestor of a graceful release will go to a simplex-in state (i.e. able only to receive, unable to send) and the remote process will go to a simplex-out state. The VC remains in this state until the remote process shuts down its socket, at which time all resources are released. See "Shutting Down a Connection" in Section 1 for a list of steps to take in implementing a graceful release shutdown.

If graceful release is selected, a SOCKERR 102 *result* will be returned if any of the following conditions exist:

- A connection request has been received, but the connection has not been accepted.
- The connection has already been gracefully released, and the process is therefore in a simplex-in state.
- A connection request has been issued, but the connection has not yet been established.
- The connection has been aborted.
- The protocol module (part of the NS transport) does not support graceful release.
- Data is being sent from the connection. This could occur, for example, if IPCSEND was called in nowait mode and has not yet completed.

Cross-System Considerations for TCP

The following are HP 3000 to HP 1000 and HP 3000 to HP 9000 programming considerations for this intrinsic:

Socket shut down - The shutdown procedure for the NS/1000 and NS3000/V processes is the same, except that the *graceful release* flag is not available on the HP 1000. If the graceful release flag (flags 17) is set on the HP 3000, the HP 1000 will respond as though it were a normal shutdown. The shutdown procedure for both NS/1000 and NS/9000 Series 800 processes is identical except for shared sockets on NS/9000 Series 800. Shared sockets are not destroyed until only one socket descriptor exists (the last socket descriptor). Therefore, the NS/9000 Series 800 process may take longer to close the connection than expected.

OPTOVERHEAD

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

Syntax

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

Parameters

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

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

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

Discussion

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

Condition codes returned by this intrinsic are:

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

This intrinsic may be called from split stack mode.

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

Syntax

```
READOPT(opt,entrynum,optioncode,datalength,data,result)
```

Parameters

opt (input)	**Record or byte array, by reference.** The *opt* parameter to be read. Refer to "NetIPC Intrinsics/Common Parameters" for information on the structure and use of this parameter.
entrynum (input)	**16-bit integer, by value.** The number of the option entry to be obtained. The first entry is number zero.
optioncode (output)	**16-bit integer, by reference.** The option code associated with the entry. These codes are described in each NetIPC call *opt* parameter description.
datalength (input/output)	**16-bit integer, by reference.** The length of the data buffer into which the entry should be read. If the data buffer is not large enough to accommodate the entry data, an error will be returned. On output, this parameter contains the length of the data actually read. (The length of the data associated with a particular option code is provided in each NetIPC call *opt* parameter description.)
data (output)	**Array, by reference.** An array which will contain the data read from the option entry. If the array is not large enough to hold the data read, nothing will be returned.
result (output)	**16-bit integer, by reference.** The error code returned; zero if no error.

ASYNCHRONOUS I/O

In order to perform nowait (asynchronous) socket I/O on an HP 3000, a process must use the MPE-V IOWAIT and IODONTWAIT intrinsics. IOWAIT and IODONTWAIT behave in the same way except that, in the first case, the calling process must wait until the I/O operation completes; in the second case, control is immediately returned to the calling process. One of these intrinsics must be called at some point after a nowait I/O request. The calling process is not blocked after the initial nowait I/O request.

IPCSEND, IPCRECV, and IPCRECVCN are normally blocking calls. The calling process must wait until the send/receive request is completed. A process can use IPCCONTROL to enable nowait I/O for a specified call socket or VC socket descriptor. (Nowait mode remains in effect until another IPCCONTROL call restores waited mode.) If a process issues a nowait send or receive request, the request will be initiated but its completion cannot be verified until IOWAIT or IODONTWAIT is called. (For a nowait IPCRECVCN call, the data structures for the connection are not created until IOWAIT is called.) IPCCONNECT is always an unblocked call: control returns immediately to the calling process, which must call IPCRECV to complete the connection.

The IPCCONTROL intrinsic itself does not function in asynchronous mode. For example, an IPCCONTROL called with direct access to X.25 in order to send an interrupt packet will not complete until the X.25 protocol receives an interrupt confirmation.

Within the IOWAIT/IODONTWAIT intrinsic, the *filenum* parameter should be given the appropriate call socket/VC socket descriptor value. A value of zero indicates that the IOWAIT intrinsic will wait for the first I/O completion from all sockets or files for which asynchronous I/O requests have been issued. The function value returned by the intrinsic is the descriptor (or file number) for which the I/O has completed (zero if no completion).

The *cstation* (calling station) parameter returns a zero value for any nowait receive request. For a nowait send request, bit one of the parameter (the second highest bit) is returned on (all other bits off). Therefore you can check bit one of the *cstation* parameter to determine whether an input or an output operation completed.

The *tcount* parameter returns the amount of data received after a nowait IPCRECV call. The *target* parameter is not currently used by NetIPC.

The syntax for IOWAIT and IODONTWAIT is given here for convenience. For further information on these intrinsics, please see the *MPE V Intrinsics Reference Manual.*

NOTE

A program does not need Privileged Mode capability in order to make nowait NetIPC I/O requests.

Steps for Programming with Asynchronous I/O

The following summarizes the steps to follow to have your program perform asynchronous I/O:

- Create the call or VC socket with IPCCREATE, IPCCONNECT, or IPCRECVCN.
- Enable nowait I/O with IPCCONTROL.
- Make a IPCRECVCN, IPCRECV, or IPCSEND NetIPC call on the socket. The call will be asynchronous.
- Check the result code returned by the call to see if an error occurred when the call was initiated.
- Call IOWAIT to cause the calling process to wait until the NetIPC call completes or IODONTWAIT to see if the request has completed.
- Once the asynchronous NetIPC call completes do the following:
  - Check the condition code to see if an error occurred. If the condition code=CCE, no error occurred. If the condition code <> CCE, an error occurred.
  - If an error occurred, call IPCCHECK to determine the error code (returned in the *ipcerr* parameter).
  - If IOWAIT or IODONTWAIT was called with *filenum*=0 or no *filenum* specified, check the *fnum* value returned to determine the socket for which I/O completed. (You can compare the *fnum* value with the *calldesc* value returned by IPCREATE and the *vcdesc* value returned by IPCCONNECT and IPCRECVCN.)
  - If both a send and receive request were pending, check the returned *cstation* value to determine if a send completed (bit 1 is on) or a receive completed (bit 1 is off).

Refer to Program 3 in Section 3 of this manual for an example program that uses these steps.

IO[DONT]WAIT

Initiates completion operations for a nowait I/O request.

Syntax

```
fnum := IO[DONT]WAIT ([filenum][,target][,tcount][,cstation])
```

Parameters

fnum (returned function value)
: **16-bit integer.** The socket/VC socket descriptor for which an I/O request has completed. Zero indicates no completion. If a *filenum* of zero is specified, and there are outstanding nowait file access request, *fnum* may return the file number of a file request that completed.

filenum (input)
: **16-bit integer, by value.** The call/VC socket descriptor indicating the socket or connection (i.e. call or VC socket) for which the nowait I/O request was issued. If omitted, or if the value is zero, any nowait NetIPC or file request issued by the calling process may be completed by this intrinsic call.

target
: **Array of 16-bit values, by reference.** Not used by NetIPC.

tcount (output)
: **16-bit integer, by reference.** Returns the amount of data received after a nowait IPCRECV call. The actual data will be in the IPCRECV *data* parameter.

cstation (output)
: **16-bit integer (unsigned), by reference.** Bit one is returned on if the completed request was a send, off if it was a receive. All other bits will be off.

Discussion

Either IOWAIT or IODONTWAIT is needed to complete a NetIPC nowait send or receive request. IOWAIT waits until a request can be completed; IODONTWAIT checks to see if a request can be completed and then immediately returns control to the calling process.

If a nowait IPCRECVCN or IPCRECV request is issued, the *data* and *flags* parameters (if specified) must exist when IOWAIT or IODONTWAIT is called. In other words, these parameters must be global to both intrinsics, the intrinsic which initiates the request and the intrinsic which attempts to complete the request (IO[DONT]WAIT).

All parameters are optional (option variable). In general, the condition codes returned by IOWAIT/IODONTWAIT for socket I/O have the following meanings:

- CCE--Succeeded.
- CCL--Failed.
- CCG--The operation succeeded but a noncritical error (e.g. *flags* parameter out of bounds) occurred.

IOWAIT and IODONTWAIT may be called in split stack mode.

NETIPC EXAMPLES

SECTION 3

This section contains examples of NetIPC program pairs. Example 1 consists of two programs that set up and use a connection to pass data from one program to another. The programs in Example 2 also pass data, and illustrate how one program, called the *server*, can be designed to communicate with multiple remote programs, called *clients*. Example 3 consists of two programs designed to provide access to the X.25 protocol (level 3).

NOTE

It is assumed that these sample programs are started by executing the :RUN command at the local and remote nodes. For an example program that illustrates the use of RPM (Remote Process Management) intrinsics to start processes on HP 3000s, refer to the *NS3000/V User/Programmer Reference Manual.* For methods of starting processes when cross-system applications are involved, refer to "Cross-System NetIPC" in Section 1.

Example 1

The following two programs comprise an example of how to set up and use a connection (virtual circuit). The two programs, running on different nodes, open communication via call sockets. They then establish a connection (between VC sockets) and use this connection to send and receive data. Finally, they terminate their connection.

In this example, the lengths of the data messages are not known. The sending side (Program 1) includes the length of each message as the first two bytes of each message it sends. The receiving side (Program 2) executes two IPCRECV loops for each message: first to receive the length and then to receive the data.

The first program (Program 1):

- looks up the call socket named RALPH located on node JANE and gets back a destination descriptor;
- creates its own call socket;
- sends a connection request to RALPH;
- shuts down its call socket and its destination socket;
- completes the connection;
- executes a loop in which it:
  - reads a line of data;

  - stores the length (number of bytes) of the data in the first part of the message;
  - stores the data itself in the second part of the message;
  - sends the message on the connection, including the message length as the first two bytes of the message;

- sends a "last message" which will be recognized by the receiving program as a termination request;
- receives a "termination confirmation message" and shuts down the connection by releasing its VC socket.

The second program (Program 2):

- creates a call socket and names it RALPH;
- waits to receive a connection request;
- shuts down its call socket;
- executes a loop in which it:

  - calls a procedure that receives a message by executing two IPCRECV loops (the first loop determines the incoming message length and the second loop receives data until all the pieces of the message have been received);
  - prints the message which was received;

- receives a "last message" termination request;
- sends a "termination confirmation message" in response to the termination request;
- receives a *result* parameter value of 64 ("REMOTE ABORTED CONNECTION") in response to a receive request;
- releases its VC socket.

NetIPC Program 1

```
$standard_level 'HP3000', uslinit$
program connection_example1 (input,output);

const
    maxdata = 2000;
    maxmsg = maxdata + 2;
    maxname = 20;
    maxloc = 20;

type
    smallint = -32768..32767;
    datatype =
        record
        len : smallint;
        msg : packed array[1..maxdata] of char;
        end;
    nametype = packed array[1..maxname] of char;
    loctype = packed array[1..maxloc] of char;

var calldesc    : integer;   {2-word integer}
    vcdesc      : integer;
    protocol    : integer;
    socket_kind : integer;
    dest        : integer;
    result      : integer;
    data        : datatype;
    name        : nametype;
    location    : loctype;
    y_len       : integer;
    y_data      : char;
    num_msgs    : integer;
    strdata     : string[maxdata];
    i           : integer;


procedure terminate; intrinsic;


{NetIPC intrinsic declarations}

procedure ipccreate; intrinsic;
procedure ipclookup; intrinsic;
procedure ipcconnect; intrinsic;
procedure ipcrecv; intrinsic;
procedure ipcsend; intrinsic;
procedure ipcshutdown; intrinsic;
procedure ipcerrmsg; intrinsic;
```

```
{error handling procedure}

procedure leave(result: integer);
    var msg: string[80];
        i, len, newresult: integer;
begin
ipcerrmsg(result, msg, len, newresult);
if newresult = 0 then
    begin
    setstrlen(msg, len);
    writeln(msg);          {print error message}
    end
else
    writeln('IpcErrMsg result is ', newresult:1);
terminate;
end;


{main of NetIPC Program 1}

begin

{ look up the call socket RALPH located on node JANE }

name := 'RALPH';
location := 'JANE';
ipclookup( name, 5, location, 4, , dest, protocol, socket_kind, result);
if result <> 0 then leave(result);      {failed}

{ create a call socket; then initiate and complete connection to
  destination socket}

ipccreate(socket_kind, protocol, , , calldesc, result);
if result <> 0 then leave(result);      {failed}
ipcconnect(calldesc, dest, , , vcdesc, result);   {initiate connection}
if result <> 0 then leave(result);      {failed}
ipcshutdown(calldesc);
ipcshutdown(dest);
ipcrecv(vcdesc, , , , , result);        {complete connection}
if result <> 0 then leave(result);      {failed}
```

```
{ prompt for messages and send them }

writeln('Enter "//" to terminate the program.');
setstrlen(strdata, 0);
while strdata <> '//' do
    begin
    prompt('Message? ');
    readln(strdata);                                 {read message}
    data.len := strlen(strdata);                     {store message length}
    strmove(data.len, strdata, 1, data.msg, 1);      {store message}
    ipcsend(vcdesc, data, data.len+2, , , result);   {send message with
                                                      length as first 2 bytes}
    if result <> 0 then leave(result);  {failed}
    end;

{connection shutdown procedure}

data.len := 4;
data.msg := 'END?';                         {termination request}
ipcsend(vcdesc, data, 6, , , result);
writeln('END sent');
if result<> 0 then leave(result);
y_len := 1;
ipcrecv(vcdesc, y_data, y_len, , , result);  {receive 'Y' confirmation}
if (y_data = 'Y') then writeln('Y received');
if (y_data = 'Y') and (result = 0) then
    ipcshutdown(vcdesc)
else
    begin
    writeln('Warning: shutdown not confirmed or result <> 0');
    leave(result);
    end;

end.
```

NetIPC Program 2

```
$standard_level 'HP3000', uslinit$
program connection_example2 (output);

const
   maxdata = 2000;
   maxname = 20;

type
    smallint = -32768..32767;
    datatype = packed array [1..maxdata] of char;
    nametype = packed array [1..maxname] of char;

var calldesc: integer;    {2-word integer}
    vcdesc  : integer;
    dlen    : integer;
    result  : integer;
    data    : datatype;
    name    : nametype;
    len     : smallint;
    datastr : string[maxdata];


procedure terminate; intrinsic;


{NetIPC intrinsic declarations}

procedure ipccreate; intrinsic;
procedure ipcname; intrinsic;
procedure ipcrecvcn; intrinsic;
procedure ipcrecv; intrinsic;
procedure ipcsend; intrinsic;
procedure ipcshutdown; intrinsic;
procedure ipcerrmsg; intrinsic;
```

```
{error handling procedure}

procedure leave(result: integer);
    var msg: string[80];
        i, len, newresult: integer;
begin
ipcerrmsg(result, msg, len, newresult);
if newresult = 0 then
    begin
    setstrlen(msg, len);
    writeln(msg);            {print error message}
    end
else
    writeln('IpcErrMsg result is ', newresult:1);
terminate;
end;


{ The following procedure receives one message which was sent via an
  ipcsend call.  It assumes that the length (number of bytes) of the
  message was sent as the first two bytes of data and that the length
  value does not include those two bytes. }

procedure receive (     connection : integer;
                    var       rbfr : datatype;
                    var       rlen : smallint;
                    var  errorcode : integer    ) ;

const
    head_len = 2;

type
    length_buffer_type = packed array[1..2] of char;
    header_len_type = record case integer of
                        0: ( word: smallint );
                        1: ( byte: length_buffer_type);
                      end;

var i, j           : integer;
    dlen           : integer;
    header_len     : header_len_type;
    tempbfr        : datatype;

begin  { procedure receive }

i:=0;
errorcode := 0;
```

```
while (i < head_len) and (errorcode = 0) do     { get length of message }
    begin
    dlen := head_len - i;
    ipcrecv( connection, tempbfr, dlen, ,, errorcode );
    if errorcode = 0
       then strmove(dlen, tempbfr, 1, header_len.byte, i+1);
    i := i + dlen;
    end;

if errorcode = 0 then
    begin
    rlen := header_len.word;
    i := 0;
    while (i < rlen) and (errorcode = 0) do     { get the message }
        begin
        dlen := header_len.word - i;
        ipcrecv( connection, tempbfr, dlen, , , errorcode );
        if errorcode = 0
           then strmove(dlen, tempbfr, 1, rbfr, i+1);
        i := i + dlen;
        end;
    end
else
    rlen := 0;

end;  { procedure receive }


{main of NetIPC Program 2}

begin

{create a call socket and name it}

ipccreate( 3, 0, , , calldesc, result);
if result <> 0 then
   leave(result);     {failed}
name := 'RALPH';
ipcname( calldesc, name, 5, result);
if result <> 0 then
   leave(result);     {failed}

{wait for a connection request}

ipcrecvcn( calldesc, vcdesc, , , result);
if result <> 0 then
   leave(result);     {failed}
ipcshutdown(calldesc);
```

```
{wait for a message on the connection and print message received}

repeat
    begin
    receive (vcdesc, data, len, result);
    if result <> 0 then leave(result);
    setstrlen(datastr, len);
    strmove(len, data, 1, datastr, 1);
    if datastr <> 'END?' then writeln (datastr);  {print data received}
    end
until datastr = 'END?';

{connection shutdown procedure}

if datastr = 'END?' then writeln('END received');
data := 'Y';
ipcsend( vcdesc, data, 1, , , result );   {confirmation message}
writeln('Y sent');
if result <> 0 then leave(result);
receive(vcdesc, data, len, result );
if result = 64 then
    ipcshutdown(vcdesc)
else
    leave(result );

end.
```

Example 2

This example provides a pair of programs referred to as a server and a client. This server-client pair is a fairly typical model of an application having multiple nodes (the clients) that request information from a database or file on a single system (the server). The server program handles incoming requests from multiple clients on a first-come, first-served basis. These programs, like the programs in Example 1, will work together on the HP 3000. In addition, each program will also work as a cross-system application with a corresponding program written for the HP 1000 or HP 9000.

The following text explains the operation of the client and server programs included in this manual.

The server program (Program 3):

- Creates a well-known call socket with IPCCREATE. (Well-known means that the socket's destination descriptor is at an address that is known by the client program. The address is declared as a constant in both programs).
- Sets the timeout to infinity on the call socket and enables nowait I/O with IPCCONTROL.
- Waits for a connection request from a client by calling IPCRECVCN. When a request is received, a response is automatically sent to the client from the transport or lower layer. (This response is received by the client's IPCRECV call).
- Once the call socket is established, the program enters its main loop. In this loop, it waits for I/O completion on all sockets by calling IOWAIT with *filenum* equal to 0. If some other function was required of the server, a polling technique with IODONTWAIT could have been used instead.
- If I/O is successfully completed on the call socket, (IPCRECVCN completes) a virtual circuit (VC) is established and the procedure HANDLENEWREQUEST does the following:
    - Uses IPCCONTROL to set the timeout to infinity and enable nowait I/O on the VC socket. Setting the timeout to infinity causes the process to wait indefinitely for incoming requests. If some other function were required of the server, a polling technique (in which the server periodically checked for requests) could have been used instead.
    - Calls IPCRECV on the virtual circuit to wait to receive the user name from the client.
    - Calls IPCRECVCN on the call socket to wait for the next connection request (from a new client).

- If IPCRECV successfully completes on a VC socket the procedure PROCESSREAD does the following:

  - Checks length of data received until all of the user name has been received from the client. Because each IPCRECV may obtain an amount of data less than or equal to the amount that has been sent by the client, multiple IPCRECV calls are used, until the correct length (20 bytes) has been received. Note that if the client name were not fixed at 20 bytes, variable length data could be handled through the manipulation of send and receive sizes with the IPCRECVCN intrinsic.

  - Reads the data corresponding to the user name from the data file (procedure READDATA).

  - Sends the data to the client with IPCSEND.

  - Calls IPCRECV again to get the next user name or shut down notification from the client.

- Control returns to the main loop, and the server waits for the next IOWAIT completion.

NOTE

If the I/O completed unsuccessfully on a call socket, the socket is shut down and the program terminates. If the I/O completed unsuccessfully on a VC socket, the socket is deleted and, unless the error was a remote abort, the error message is printed.

The client program (Program 4):

- Prompts the user for the name of the remote node (on which the server resides).
- Creates a call socket with IPCCREATE.
- Creates a destination descriptor for the socket using IPCDEST (using the well-known address).
- Sends a connection request to the client using IPCCONNECT. This request is received by the server's call to IPCRECVCN.
- Changes timeout on the VC socket to infinity with IPCCONTROL. This causes the client to wait for the server's response indefinitely unless the client receives notification that the connection is down.
- Receives the server's response to the connection request through IPCRECV.
- Prompts the user for a user name and reads the name that is entered.

- For any name except "EOT," the client sends the name to the server with IPCSEND. If the name entered is "EOT," the call socket and then the VC socket is shut down.
- The client receives the data from the server's data file corresponding to the user name through IPCRECV. This data is stored.
- Finally, the client prints out the data.

The client program continues to loop until "EOT" is entered in response to a prompt.

NetIPC Program 3 (Server Program)

```
$ STANDARD_LEVEL 'HP3000', USLINIT, TABLES ON, CODE_OFFSETS ON $
$ COPYRIGHT 'Hewlett_Packard Co.' $
PROGRAM server( input, output );

{----------------------------------------------------------------------}
{                                                                      }
{   SERVER: Async Server Sample Program                                }
{                                           Revision: <xxxxxx.xxxx>    }
{----------------------------------------------------------------------}
{
{
{----------------------------------------------------------------------}
{  COPYRIGHT (C) 1987 HEWLETT-PACKARD COMPANY.
{  All rights reserved.  No part of this program may be photocopied,
{  reproduced or translated into another programming language without
{  the prior written consent of the Hewlett-Packard Company.
{----------------------------------------------------------------------}
{
{----------------------------------------------------------------------}
{    Name : Server
{  Source : xxxxxxxxxxx
{    Date : <xxxxxx.xxxx>
{----------------------------------------------------------------------}
{
{
{ PURPOSE:
{    To show the operation of asynchronous NetIPC calls.
{
{ REVISION HISTORY
{}
{ DESCRIPTION
{    The Server uses IPC to receive a user name from a Client and sends
{    information associated with the user name back to the Client.
{    The Server can have connections to 63 Clients.
{
{    General Algorithm:
{    1.  Create a well-known call socket (IPCCreate).
{    2.  Post a nowait IPCRecvCn to receive connection requests sent from
{        clients.
{    3.  When the IPCRecvCn completes, receive the connection and post
{        a nowait IPCRecv to receive the requested user name.
{    4.  Since the IPCRecv may complete before receiving all of the user name,
{        additional IPCRecv calls may have to be posted to receive all of
{        the user name.
{    5.  Once the all of the user name is received, open the file
{        named "datafile."  Scan datafile until the user
{        record and information associated with the user name are found.
{    6.  Call IPCSend (nowait) to send the information associated
{        with the user name.
```

```
{    7.  Post a nowait IPCRecv on the VC to receive next user name or
{        shutdown notification from the remote (the IPCRecv completes
{        with SOCKERR 64, REMOTE ABORT).
{    8.  Upon receipt of shutdown notification, call IPCShutDown to shut the VC.
{                                                                            }
{    Since all IPC calls (except IPCCreate) are done nowait, the main loop
{    calls IOWAIT, determines what type of event completed, and calls the
{    appropriate procedure to handle the event.
{                                                                            }
{    Major Data Structures:
{
{    datafile:  datafile contains a fixed number of records (4).
{    Each record contains a 20-byte user name and an 60-byte information
{    field, i.e.:
{            +------------------------------+
{            | user name |  information      |
{            +------------------------------+
{            1        20 | 1               60
{
{    vc_record:  For each VC, an IPCRecv is posted to receive the user name.
{    Since the IPCRecv may complete without receiving all the data,
{    multiple IPCRecv calls may be necessary to receive the entire name.
{    Since the server may have several VCs open simultaneously,
{    this program allocates a buffer for each open VC to store the
{    user name as it is received, and the number of bytes already received
{    for each VC is counted.  This information is kept in a linked list
{    of vc_records, which has the following format:
{
{   +-------------------------------+    +-------------------------------+
{   | sd | buff_len | buffer | next |    | sd | buff_len | buffer | next |
{   |    |          |        |   o--|--> |    |          |        |   o--|-->
{   +-------------------------------+    +-------------------------------+
{
{   The field sd is the vc socket descriptor returned by the IPC and IOWAIT
{   calls; buff_len is the number of bytes already received by the VC; buffer
{   is used to store the user name; next points to the next vc_record.
{
{   New records are added to the head of the list.
{   The pointer head_ptr points to the head; curr_ptr is used to point
{   to the vc_record for which an event must be processed; prev_ptr is used
{   to update the list when a record is deleted.
{
{   As an alternative, a program with multiple VCs can loop on a single VC
{   once the first IPCRecv call completes, making all the IPCRecv calls
{   necessary to receive the rest of the data.  Only one buffer is
{   required, but the program is tied to one VC until all the data for that
{   VC is received.  Since the program is tied to a single VC, the receive
{   scheme can be the same as one that is used for programs that have only
{   one VC.  (For an example of a receive scheme for a single VC, refer to
{   the Client program example.)                                            }
{                                                                           }
LABEL
   99;
```

```
CONST

   ALL_SOCKETS = 0;                    { used to call IOWAIT on all sockets }
   BUFFERLEN = 20;                     { user name buffer length }
   CALL_SOCKET = 3;
   CCE = 2;
   CCG = 0;
   CCL = 1;
   CHANGE_TIMEOUT = 3;
   ENABLE_NOWAIT = 1;
   FOREVER = TRUE;
   INFINITE_SELECT = -1;
   INFOBUFLEN = 60;                    { information buffer length }
   INT16_LEN = 2;
   MAX_BUFF_SIZE = 1000;
   MAX_RCV_SIZE = 4;
   MAX_SEND_SIZE = 3;
   PROTO_ADDR = 128;
   REMOTE_ABORT = 64;
   TCP = 4;
   TCP_ADDRESS = 31767;
   ZERO = 0;


TYPE
  {}
  ShortInt = -32768..32767;
  byte = 0..255;
  byte_array_type = packed array [1..40] of byte;          { for opt array       }
  Buffer_Type = packed array [1..BUFFERLEN] of char;       { for user name       }
  InfoBufType = packed array [1..INFOBUFLEN] of char;      { for info buffer     }
  name_of_call_array_type = packed array [1..10] of char;  { for reporting IPC   }
                                                           { call that causes    }
                                                           { error               }

  vc_ptr_type = ^vc_record_type;                          { to hold user names }
  vc_record_type = record                                  { received on VCs    }
      sd           : integer;
      buff_len     : integer;
      buffer       : Buffer_Type;
      next         : vc_ptr_type;
      end;
  severity_type = (RECOVERABLE, IRRECOVERABLE);

VAR
  call_name                   : name_of_call_array_type;
  call_sd                     : integer;   { set by IPCCreate          }
  control_value               : ShortInt;          { for IPCControl    }
  cstation                    : ShortInt;  { for IOWAIT--indicates if  }
                                           { receive or send completed }

  curr_sd                     : integer;
  curr_ptr                    : vc_ptr_type;
  dlen                        : Integer;
  head_ptr                    : vc_ptr_type;  { point to head of VC list }
```

```
  infofile                          : Buffer_Type;
  new_vc                            : integer;      { set by IPCRecvCn        }
  opt_data                          : ShortInt;
  opt_num_arguments                 : ShortInt;
  option                            : byte_array_type;
  prev_ptr                          : vc_ptr_type;
  protocol_kind                     : Integer;
  severity                          : severity_type;
  snum                              : ShortInt;
  result                            : Integer;
  short_error                       : ShortInt;      { for OPT calls          }
  socket_kind                       : Integer;
  TempBuff                          : Buffer_Type;   { for IPCRecv            }
  timeout                           : Integer;
  timeout_len                       : Integer;

$ TITLE 'Procedures', PAGE $
PROCEDURE ADDOPT; INTRINSIC;
PROCEDURE INITOPT; INTRINSIC;
PROCEDURE IPCCheck; INTRINSIC;
PROCEDURE IPCControl; INTRINSIC;
PROCEDURE IPCCREATE; INTRINSIC;
PROCEDURE IPCNAME; INTRINSIC;
PROCEDURE IPCRECVCN; INTRINSIC;
PROCEDURE IPCRECV; INTRINSIC;
PROCEDURE IPCSEND; INTRINSIC;
PROCEDURE IPCSHUTDOWN; INTRINSIC;
FUNCTION IOWAIT : ShortInt; INTRINSIC;
PROCEDURE FCHECK; INTRINSIC;
PROCEDURE FCLOSE; INTRINSIC;
FUNCTION FOPEN : ShortInt; INTRINSIC;

{        Internal Procedures          }

PROCEDURE CHECK_FILE;
   FORWARD;
   { Check that the file exists.      }

PROCEDURE ERROR_ROUTINE
  ( where   : name_of_call_array_type;
    what    : integer;
    sd      : integer;
    severity : severity_type);
   FORWARD;
   { If irrecoverable error, print error message and terminate.           }
   { If recoverable error, just print error message.                      }

PROCEDURE HANDLENEWREQUEST
  (VAR new_vc : integer);
   FORWARD;
   { Called when an IPCRecvCn completes--a new client wants to talk to us }

PROCEDURE PROCESSREAD
  (VAR curr_vc : integer);
```

```
   FORWARD;
   { Called when an IPCRecv completes successfully.                          }

PROCEDURE READDATA
(VAR client_buf    : Buffer_Type;      { contains user name requested        }
 VAR output_buf    : InfoBufType );    { will contain information buffer     }
   FORWARD;
  { Called by ProcessRead.  Gets information buffer from datafile.           }

PROCEDURE SETUP;
   FORWARD;
   { Create a TCP call socket using a well-known address }

PROCEDURE SHUTDOWNVC
  (VAR shut_vc : integer);
   FORWARD;
   { Shut down a VC that the client no longer needs (IPCShutdown).  }
   { Called when an error occurs on a VC, including REMOTE_ABORT.   }

$ TITLE 'Check_File', PAGE $
{-------------------------------------------------------------------------}
{ Procedure:  CHECK_FILE                                                  }
{                                                                         }
{ Check that the file can be opened before resetting it to read data.    }
{ Called once, at the beginning of main.                                  }
{-------------------------------------------------------------------------}

PROCEDURE CHECK_FILE;
CONST
   EXISTING_FILE = 1;                  { for FOPEN                   }
   NO_CHANGE = 0;
   SEC_CODE = 0;

VAR
   fnum : ShortInt;

begin
   fnum := FOPEN ( infofile, EXISTING_FILE );
   if ccode <> CCE then  { FOPEN unsuccessful.  Call ERROR_ROUTINE. }
       begin
       FCHECK ( fnum, short_error );
       call_name := 'FOPEN       ';
       result := short_error;
       severity := IRRECOVERABLE;
       ERROR_ROUTINE( call_name, result, 0 , severity );
       end     { ccode <> CCE }

   else        { FOPEN was successful.  Close file and return. }
      FCLOSE ( fnum, NO_CHANGE, SEC_CODE );
end;  { CHECK_FILE }
```

```
$ TITLE 'Error_Routine', PAGE $
{-----------------------------------------------------------------------}
{ Procedure:  ERROR_ROUTINE                                             }
{                                                                       }
{ Called if error <> REMOTE_ABORT.                                      }
{ If irrecoverable error, print error message and terminate.            }
{ If recoverable error (error on IPCSend or IPCRecv I/O completion),    }
{ just print error message.                                             }
{-----------------------------------------------------------------------}
PROCEDURE ERROR_ROUTINE
  ( where   : name_of_call_array_type;
    what    : integer;
    sd      : integer;
    severity : severity_type);

   begin    { Error_Routine }
      begin
      writeln('Server: Error occurred in ', where,' call.' );
      writeln('Server: The error code is: ', what:5,
                 '.  The local descriptor is: ', sd:4 );
      end;

   if severity = IRRECOVERABLE then
      GOTO 99;   { terminate }

   end;    { Error_Routine }

$ TITLE 'HandleNewRequest', PAGE $
{-----------------------------------------------------------------------}
{ Procedure:  HANDLENEWREQUEST                                          }
{                                                                       }
{ Called when an IPCRecvCn completes--a new client wants to talk to us }
{ Allocate and initialize vc_record for the new VC.                     }
{ Set the timeout to infinity with IPCControl for later calls           }
{ Enable NOWAIT IO for the VC socket (IPCControl)                       }
{ Post an IPCRecv on the VC to receive user name                        }
{ Post another IPCRecvCn to receive the next connection request         }
{-----------------------------------------------------------------------}

PROCEDURE HANDLENEWREQUEST
  (VAR new_vc : integer);

   begin    { HandleNewRequest }

   { Get a vc_record for the new VC; add to list; initialize vc_record }
   new( curr_ptr );

   { Add to the head of the list                                       }
   curr_ptr^.NEXT := HEAD_PTR;
   head_ptr := curr_ptr;

   { Initialize vc_record                                              }
   with curr_ptr^ do
      begin
```

```
      sd := new_vc;
      buff_len := 0;
   end;  { with curr_ptr }

   { Set the timeout to infinity with IPCControl for later calls }
   control_value := 0;
   timeout_len := 2;

   opt_num_arguments := 1;
   InitOpt( option, opt_num_arguments, short_error );
   IF short_error <> ZERO THEN
      begin    { error on initopt }
      call_name := 'InitOpt    ';
      result := short_error;
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name, result, call_sd, severity );
      end;     { error on initopt }

   IPCControl( new_vc, CHANGE_TIMEOUT, control_value, timeout_len, , , ,
     result );

   IF result <> ZERO THEN
      begin
      call_name := 'IPCControl';
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, new_vc, severity );
      end;

   { Enable NOWAIT IO for the VC socket (IPCControl)             }

   IPCControl( new_vc, ENABLE_NOWAIT , , , , , , result );

   IF result <> ZERO THEN
      begin
      call_name := 'IPCControl';
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, new_vc, severity );
      end;

{ Post an IPCRecv on the new VC to get user name                       }
   dlen := BUFFERLEN;
   IPCRecv( new_vc, TempBuff, dlen, , , result );
   IF result <> ZERO THEN
      begin
      call_name := 'IPCRecv    ';
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, new_vc, severity );
      end;

{ Post another IPCRecvCn to receive the next connection request       }
   IPCRecvCn( call_sd, new_vc, , , result );
   IF result <> ZERO THEN
      begin
      call_name := 'IPCRevcCn ';
```

```
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, new_vc, severity );
      end;

   end;     { HandleNewRequest }

$ TITLE 'ProcessRead', PAGE $
{-----------------------------------------------------------------------------}
{ Procedure:  PROCESSREAD                                                     }
{                                                                             }
{ Called when an IPCRecv completes.  If all of the user name is received,     }
{ call ReadData and send information buffer (IPCSend).                        }
{ Otherwise, post another IPCRecv to get rest of user name.                   }
{-----------------------------------------------------------------------------}
PROCEDURE PROCESSREAD
  (VAR curr_vc : integer);

VAR
   data_buf         : InfoBufType;
   i                : ShortInt;

   begin    { ProcessRead }
   { Get the name this client wants data for    }

   { Scan linked list for vc_record for curr_vc }
   curr_ptr := head_ptr;
   while (curr_ptr <> nil) and (curr_ptr^.sd <> curr_vc) do
      curr_ptr := curr_ptr^.next;
   if curr_ptr = nil then                { vc_record not found for curr_vc }
      begin                              { INTERNAL ERROR                  }
      call_name := 'Program   ';
      result    := 0;
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, curr_vc, severity );
      end
   else
      { vc_record found, so move data to vc_record buffer                  }
      begin
      for i := 1 to dlen do
         curr_ptr^.buffer[i + curr_ptr^.buff_len] := TempBuff[i];
      { update length (amount of data received)                            }
      curr_ptr^.buff_len := curr_ptr^.buff_len + dlen;
      end;

   if curr_ptr^.buff_len < BUFFERLEN then
{ Not all of the name was received, so post another IPCRecv on the VC    }
{ to receive the rest of the name }
      begin
      dlen := BUFFERLEN - curr_ptr^.buff_len;
      IPCRecv( curr_vc, TempBuff, dlen, , , result );
      IF result <> ZERO THEN
         begin
         call_name := 'IPCRecv   ';
         severity := IRRECOVERABLE;
```

```
        ERROR_ROUTINE( call_name,result, curr_vc, severity );
        end;
     end;  { if curr_ptr^.buff_len < BUFFERLEN }

  if curr_ptr^.buff_len = BUFFERLEN then
  { Received all of the name, so call ReadData to get the data   }
  { we need from the file to send to the client.                 }
  { Also reset buff_len to 0 to receive next name request        }
  begin
  curr_ptr^.buff_len := 0;
  ReadData( curr_ptr^.buffer, data_buf );
  { Send information buffer to client                            }
  IPCSend( curr_vc, data_buf, INFOBUFLEN, , , result );
  IF result <> ZERO THEN
     begin
     call_name := 'IPCSend    ';
     severity := IRRECOVERABLE;
     ERROR_ROUTINE( call_name,result, curr_vc, severity );
     end;

  { Post another IPCRecv on VC to get next user name or shut down           }
  { notification from client.                                               }
  IPCRecv( curr_vc, TempBuff, dlen, , , result );
  IF result <> ZERO THEN
     begin
     call_name := 'IPCRecv    ';
     severity := IRRECOVERABLE;
     ERROR_ROUTINE( call_name,result, curr_vc, severity );
     end;
  end;
end;      { ProcessRead }

$ TITLE 'ReadData', PAGE $
{-----------------------------------------------------------------------}
{ Procedure:  READDATA                                                  }
{                                                                       }
{ Called by ProcessRead.  Input is user name (client_buf).              }
{ Returns information buffer associated with user (output_buf).         }
{-----------------------------------------------------------------------}
PROCEDURE READDATA
(VAR client_buf    : Buffer_Type;
 VAR output_buf    : InfoBufType );

CONST
   LAST_REC    = 4;

VAR
   current_rec     : ShortInt;
   datafile        : TEXT;
   info_buf        : InfoBufType;
   found           : Boolean;
   name_buf        : Buffer_Type;
```

```
begin    { ReadData }

{}
{ Open the file named "datafile".
{ Each record contains a 20-byte user name and an 60-byte information field.
{ Search until the last record is found, or we match the user name the
{ client wants.
{ If there is a match, retrieve the remaining data in the record
{ (the information field), and prepare to send it back.
{ If there is no match, return "name not found" to the client.
{}

found := FALSE;
current_rec := 1;

RESET( datafile, infofile );

WHILE ( NOT found ) AND ( current_rec <= LAST_REC ) DO
   begin    { search the file }

   READLN( datafile, name_buf, info_buf );

   IF client_buf = name_buf THEN
      begin    { found a match }
      {}
      { We found the name the client requested in the file.
      { Set the flag to fall out of the while loop, and
      { get the buffer to be sent to the client.
      {}
      writeln( 'Server: ', client_buf, ' information found.' );

      found := TRUE;
      output_buf := info_buf;

      end;     { found a match }

   { increment to test the next record in the file }
   current_rec := current_rec +1;

   end;     { search the file }

{}
{ We've fallen out of the WHILE loop because there is a match,
{ or we reached the end of the file.  Find out which one it is.
{}

IF NOT found THEN
   begin    { didn't find the requested name }

   {}
   { We didn't find the data in the file.  Put an error
   { message in the data buffer.
   {}
   writeln ('Server: ', client_buf, ' not in file.' );
```

```
      output_buf :=
    'SERVER did not find the requested name in the datafile.    ';

      end;      { didn't find the requested name }


   end;      { ReadData }

$ TITLE 'SetUp', PAGE $
{---------------------------------------------------------------------------}
{ Procedure:  SETUP                                                         }
{                                                                           }
{ Create a TCP call socket using a well-known address                       }
{ Set the timeout to infinity with IPCControl for later calls               }
{ Enable NOWAIT IO for the VC socket (IPCControl)                           }
{ Post an IPCRecvCn to receive connection requests                          }
{ Global Variables:  sets call_sd with IPCCreate;                           }
{                    new_vc is set when IPCRecvCn completes                 }
{---------------------------------------------------------------------------}
PROCEDURE SETUP;

   begin    { SetUp }

   { Set up the opt array for one option                 }
   opt_num_arguments := 1;
   InitOpt( option, opt_num_arguments, short_error );
   IF short_error <> ZERO THEN
      begin     { error on initopt }
      call_name := 'InitOpt    ';
      result := short_error;
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result,call_sd, severity );
      end;      { error on initopt }

   { Add the option for the well-known address for the IPCCreate Call }
   opt_data := TCP_ADDRESS;
   AddOpt( option, 0, PROTO_ADDR, INT16_LEN, opt_data, short_error );
   IF short_error <> ZERO THEN
      begin     { error on AddOpt }
      call_name := 'AddOpt     ';
      result := short_error;
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result,call_sd, severity );
      end;      { error on AddOpt }

   { Prepare to create a call socket }
   socket_kind := CALL_SOCKET;
   protocol_kind := TCP;

   {A call socket is created by calling IPCCREATE.  The value returned
   {in the call_sd parameter will be used to check for an IPCRecvCn
   {}

   IPCCREATE( socket_kind, protocol_kind, ,option, call_sd, result );
```

```
    IF result <> ZERO THEN
       begin
       call_name := 'IPCCreate ';
       severity := IRRECOVERABLE;
       ERROR_ROUTINE( call_name,result,call_sd, severity );
       end;

{ Set the call_sd timeout to infinity with IPCControl for later calls }
control_value := 0;
timeout_len := 2;

IPCControl( call_sd, CHANGE_TIMEOUT, control_value, timeout_len, , , ,
   result );

IF result <> ZERO THEN
   begin
   call_name := 'IPCControl';
   severity := IRRECOVERABLE;
   ERROR_ROUTINE( call_name,result,call_sd, severity );
   end;

   { Enable NOWAIT IO for the call socket                          }

   IPCControl( call_sd, ENABLE_NOWAIT , , , , , , result );

   IF result <> ZERO THEN
      begin
      call_name := 'IPCControl';
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, call_sd, severity );
      end;

{ Post an IPCRecvCn to receive a connection request                 }
   IPCRecvCn( call_sd, new_vc, , , result );
   IF result <> ZERO THEN
      begin
      call_name := 'IPCRevcCn ';
      severity := IRRECOVERABLE;
      ERROR_ROUTINE( call_name,result, new_vc, severity );
      end;

   end;      { SetUp }

$ TITLE 'ShutdownVC', PAGE $
{----------------------------------------------------------------------}
{ Procedure:  SHUTDOWNVC                                               }
{                                                                      }
{ Shut down a VC that the client no longer needs.                      }
{ Called when an error occurs on a VC, including REMOTE_ABORT          }
{----------------------------------------------------------------------}
PROCEDURE SHUTDOWNVC
  (VAR shut_vc : integer);
```

```
begin    { ShutdownVC }
{}
{ The client shut down the VC, or it has gone down due to a
{ networking problem.  Either way, accept the shutdown.
{}
IPCShutdown( shut_vc, , , result );
IF result <> ZERO THEN
   begin
   call_name := 'IPCShutdwn';
   severity := IRRECOVERABLE;
   ERROR_ROUTINE( call_name,result, shut_vc, severity );
   end;

{ Delete vc_record from linked list                                  }
prev_ptr := nil;
curr_ptr := head_ptr;
while (curr_ptr <> nil) and (curr_ptr^.sd <> shut_vc) do
   begin
   prev_ptr := curr_ptr;
   curr_ptr := curr_ptr^.next;
end;          { while curr_ptr <> nil... }
if curr_ptr = nil then                { vc_record not found for shut_vc }
   begin                              { INTERNAL ERROR                  }
   call_name := 'Program   ';
   result    := 0;
   severity := IRRECOVERABLE;
   ERROR_ROUTINE( call_name,result, shut_vc, severity );
   end;
if prev_ptr = nil then                       { deleting first entry   }
   head_ptr := head_ptr^.next
else
   prev_ptr^.next := curr_ptr^.next;         { deleting middle entry  }

{ Deallocate vc_record                                               }
dispose(curr_ptr);
end;     { ShutdownVC }

$ TITLE 'Server MAIN', PAGE $
{-----------------------------------------------------------------------}
{ MAIN                                                                  }
{-----------------------------------------------------------------------}
{ Set up a TCP call socket with a well-known address.  Post a nowait    }
{ IPCRecvCn on the call socket and wait for a connection request (SETUP). }
{                                                                       }
{ Loop forever waiting for I/O completions.                             }
{ Exit the loop if an irrecoverable error (error other than error on    }
{ VC I/O completion) occurs by calling ERROR_ROUTINE with IRRECOVERABLE  }
{ severity.                                                             }
{ The following events can occur:                                       }
{   -IPCRecvCn successfully completes (CCE and snum = call socket desc). }
{    A new Client wants service.                                        }
{      Action:  call HANDLENEWREQUEST                                   }
{   -IPCRecv successfully completes (CCE, snum <> call socket descriptor & }
{    cstation = 0).  A Client is sending us a user name.                }
```

```
{      Action:  call PROCESSREAD                                           }
{   -IPCSend successfully completes (CCE, snum <> call socket descriptor & }
{    cstation <> 0).                                                       }
{      Action:  nothing (nothing needs to be done)                         }
{   -Error occurs on call socket (CC <> CCE and snum = call socket desc).  }
{      Action:  set severity to IRRECOVERABLE and call ERROR_ROUTINE.      }
{               ERROR_ROUTINE will print error message and terminate.      }
{   -Error occurs on VC socket (CC <> CCE and snum <> call socket desc and }
{     result = SOCKERR 64, REMOTE ABORT).                                  }
{    This means that the remote shut down its VC; not really an error.     }
{      Action: call SHUTDOWNVC (IPCShutdown and delete VC record)          }
{   -Error occurs on VC socket (CC <> CCE and snum <> call socket desc and }
{     result <> SOCKERR 64, REMOTE ABORT).                                 }
{    An error occurred on a VC other than a remote abort.                  }
{      Action: set severity to RECOVERABLE and call ERROR_ROUTINE.         }
{              ERROR_ROUTINE will print error message and continue.        }
{              Call SHUTDOWNVC (IPCShutdown and delete VC record).         }
{ Handle each one of these cases in this loop.
{
{}

begin { Server }

head_ptr := nil;
infofile := 'datafile';

CHECK_FILE;  { Make sure datafile can be opened.                            }

SETUP;       { Create a call socket with a well-known address for the       }
             { clients to call.                                             }
             { Post an IPCRecvCn (NOWAIT) to receive connection requests.   }

WHILE FOREVER = TRUE DO       { Loop, waiting for I/O to complete.          }
   begin    { Forever Do }
   snum := IOWAIT(ALL_SOCKETS, , , cstation);  { wait for I/O completion on }
                                               { any one of the sockets     }
   if ccode = CCE then
   { successful completion }
      if snum = call_sd then
      { successful I/O completion on call socket, so must be IPCRecvCn }
         HANDLENEWREQUEST( new_vc )             { new_vc was set to the new VC}
                                                { in the IPCRecvCn call       }
      else
      { successful I/O completion on VC   }
         begin
         curr_sd := snum;
         if cstation = 0 then                   { IPCRecv completed.         }
            PROCESSREAD( curr_sd );             { Client sent us user name.  }
         end;

   if ccode <> CCE then             { error, including shutdown notification }
      begin
      IPCCheck (snum, result, , );    { get IPC error code                  }
         if snum = call_sd then       { error on call socket--terminate     }
```

```
            begin
            call_name := 'IPCRecvCn ';
            severity := IRRECOVERABLE;
            ERROR_ROUTINE( call_name, result, call_sd, severity );
            end
         else
         { Error on VC.  If not REMOTE_ABORT, print message and delete VC. }
         { If REMOTE_ABORT, just delete VC.                                }
            begin
            curr_sd := snum;
            if result <> REMOTE_ABORT then
               begin
               if cstation = 0 then
                  call_name := 'IPCRecv    '
               else
                  call_name := 'IPCSend    ';
               severity := RECOVERABLE;
               ERROR_ROUTINE ( call_name, result, curr_sd, severity );
               end;  { result <> REMOTE_ABORT }

            SHUTDOWNVC( curr_sd );  { error on VC, so delete it               }
            end;   { snum <> call_sd }
      end;  { ccode <> CCE }
   end;     { Forever Do }

99: writeln;

{}
{ We encountered an irrecoverable error (error other than error on VC I/O
{ completion).  The NS cleanup routine will shut down
{ all the sockets we own once the program has terminated.
{}

end.  { Server }
```

NetIPC Program 4 (Client Program)

```
$ STANDARD_LEVEL 'HP3000', USLINIT, TABLES ON, CODE_OFFSETS ON$
$ COPYRIGHT 'Hewlett_Packard Co.' $
PROGRAM Client( input, output );

{-----------------------------------------------------------------------}
{                                                                       }
{   Client: Client Sample Program                                       }
{                                              Revision: <870610.1327>  }
{-----------------------------------------------------------------------}
{
{
{-----------------------------------------------------------------------}
{  COPYRIGHT (C) 1987 HEWLETT-PACKARD COMPANY.
{  All rights reserved.  No part of this program may be photocopied,
{  reproduced or translated into another programming language without
{  the prior written consent of the Hewlett-Packard Company.
{-----------------------------------------------------------------------}
{
{-----------------------------------------------------------------------}
{    Name : Client
{    Date : <870610.1327>
{-----------------------------------------------------------------------}
{
{
{ PURPOSE:
{    Client to correspond with async Server example.
{
{ REVISION HISTORY
{}
{ DESCRIPTION
{    The Client uses IPC to send a user name to the Server and receive
{    information associated with the user name from the Server.
{
{    General Algorithm:
{    1.  Get the name of the remote node from the user.
{    2.  Create a call socket (IPCCreate).
{    3.  Get the path descriptor for the Server's well-known socket
{        (IPCDest).
{    4.  Request connection to the Server (IPCConnect).
{    5.  Receive connection verification (IPCRecv).
{    6.  Loop--ask the user for user name for information retrieval
{        (until the user enters the string literal 'EOT').
{    7.  Send the user name to the Server (IPCSend).
{    8.  Receive the information associated with the user name (IPCReceive).
{}
LABEL
   89,
   99;
```

```
CONST

   BUFFERLEN = 20;
   CALL_SOCKET = 3;
   CHANGE_TIMEOUT = 3;
   FOREVER = TRUE;
   INFINITE_SELECT = -1;
   INFOBUFLEN = 60;
   INT16_LEN = 2;
   LENGTH_OF_DATA = 20;
   MAX_BUFF_SIZE = 1000;
   MAX_RCV_SIZE = 4;
   MAX_SEND_SIZE = 3;
   MAX_SOCKETS = 32;
   INTEGER_LEN = 2;
   TCP = 4;
   TCP_ADDRESS = 31767;                 { Well-known TCP address used by Server }
   ZERO = 0;

TYPE
  {}
  {}
  ShortInt = -32768..32767;
  byte = 0..255;
  byte_array_type = packed array [1..8] of byte;
  buffer_type = packed array [1..BUFFERLEN] of char;
  InfoBufType = packed array [1..INFOBUFLEN] of char;
  name_of_call_array_type = packed array [1..10] of char;

VAR
  buffer_len                    : Integer;
  call_name                     : name_of_call_array_type;
  call_sd                       : integer;
  control_value                 : ShortInt;
  data_buf                      : InfoBufType;
  error_return                  : Integer;
  node_name                     : Buffer_Type;
  node_name_len                 : Integer;
  proto_addr                    : ShortInt;
  protocol_kind                 : Integer;
  req_name_len                  : Integer;
  requested_name                : Buffer_Type;
  socket_kind                   : Integer;
  temp_position                 : ShortInt;
  timeout                       : Integer;
  timeout_len                   : Integer;
  vc_sd                         : Integer;

$TITLE 'IPC Procedures', PAGE $
PROCEDURE IPCConnect; INTRINSIC;
PROCEDURE IPCControl; INTRINSIC;
PROCEDURE IPCCREATE; INTRINSIC;
PROCEDURE IPCNAME; INTRINSIC;
PROCEDURE IPCDEST; INTRINSIC;
```

```
PROCEDURE IPCRECVCN; INTRINSIC;
PROCEDURE IPCRECV; INTRINSIC;
PROCEDURE IPCSEND; INTRINSIC;
PROCEDURE IPCSHUTDOWN; INTRINSIC;

$ TITLE 'Internal Procedures', PAGE $

PROCEDURE GETLEN
(VAR  buffer      :   Buffer_Type;
 VAR  current_pos :   ShortInt;
 VAR  length      :   Integer );
   FORWARD;
   { Get the length of a string.  Return the next postion }

PROCEDURE ERROR_ROUTINE
  (VAR where  : name_of_call_array_type;
       what   : integer;
       sd     : integer);
   FORWARD;

PROCEDURE RECEIVEDATA
  (VAR info_buf : InfoBufType);
   FORWARD;

PROCEDURE SETUP;
   FORWARD;
   { Create a call socket, connect to server using IPCDest }

PROCEDURE SHUTDOWNSOCKETS;
   FORWARD;
   { Shut down the call and vc sockets }


$ TITLE 'Error_Routine', PAGE $
PROCEDURE ERROR_ROUTINE
  (VAR where  : name_of_call_array_type;
       what   : integer;
       sd     : integer);

{-------------------------------------------------------------------}
{ Procedure:  ERROR_ROUTINE                                         }
{                                                                   }
{ Called if result code returned <> 0.                              }
{ Prints error message and terminates.                              }
{-------------------------------------------------------------------}
   BEGIN    { Error_Routine }

   writeln('Client: Error occurred in ', where,' call.' );
   writeln('Client: The error code is: ', what:5,
                  '.  The local descriptor is: ', sd:4 );

   GOTO 89;

   END;    { Error_Routine }
```

```
$ TITLE 'GetLen', PAGE $
PROCEDURE GETLEN
(VAR  buffer      :   Buffer_Type;
 VAR  current_pos :   ShortInt;
 VAR  length      :   Integer );

{----------------------------------------------------------------------}
{ Procedure:  GETLEN                                                   }
{                                                                      }
{ Get the length of a string.  Return the next postion.                }
{----------------------------------------------------------------------}

VAR
   orig_pos :    ShortInt;

   BEGIN    { GetLen }
   {}
   { Find the first blank in the string.  Return the difference
   { between the blank position, and the initial value of current_pos
   {}

   orig_pos := current_pos;

   WHILE buffer[current_pos] <> ' ' DO
      current_pos := current_pos + 1;

   { set the length value for the caller }
   length := current_pos - orig_pos;

   { increment beyond the space, for the next time }
   current_pos := current_pos + 1;

   END;     { GetLen }

$ TITLE 'ReceiveData', PAGE $
PROCEDURE RECEIVEDATA
   (VAR info_buf : InfoBufType );  { on exit, will contain the information     }
                                   { buffer received from the server           }

{----------------------------------------------------------------------}
{ Procedure:  RECEIVEDATA                                              }
{                                                                      }
{ Receives data from Server.  Loops on IPCRecv until total amount of   }
{ data is received.                                                    }
{----------------------------------------------------------------------}

VAR
   temp_buf : InfoBufType; { used for IPCRecv               }
   i        : integer;  { amount of data currently received }
   j        : integer;  { array index for moving data from temp_buf to info_buf}

   BEGIN     { ReceiveData }
   i := 0;
   while i < INFOBUFLEN do
```

```
      begin
      buffer_len := INFOBUFLEN - i;
      IPCRecv( vc_sd, temp_buf, buffer_len, , ,
            error_return );
      IF error_return <> ZERO THEN
         BEGIN      { error on IPCRecv }
         call_name := 'IPCRecv    ';
         Error_Routine( call_name, error_return, vc_sd );
         END        { error on IPCRecv }

      ELSE
      { error_return = ZERO; IPCRecv successful }
      { move data to info_buf (output buffer)    }
         begin
         for j := 1 to buffer_len do
            info_buf[i + j] := temp_buf[j];
         i := i + buffer_len;        { update amount of data received     }
         end;
      end;     {while                  }

   END;     { ReceiveData }

$ TITLE 'SetUp', PAGE $
PROCEDURE SETUP;
{-------------------------------------------------------------------------}
{ Procedure:  SETUP                                                       }
{                                                                         }
{ Create a TCP call socket (IPCCreate).                                    }
{ Create destination descriptor for Server's well-known call socket (IPCDest). }
{ Establish VC with Server (IPCConnect).                                   }
{ Set the VC timeout to infinity (IPCControl).                             }
{ Call IPCRecv to verify the Server received the connect request           }
{ (wait for the Server to call IPCRecvCn).                                 }
{-------------------------------------------------------------------------}

VAR
   destdesc     : Integer;

   BEGIN    { SetUp }

   { Prepare to create a call socket }
   socket_kind := CALL_SOCKET;
   protocol_kind := TCP;


   {}
   {A call socket is created by calling IPCCREATE.  The value returned
   {in the call_sd parameter will be used in the subsequent calls.
   {}
   IPCCreate( socket_kind, protocol_kind, , , call_sd, error_return );

   IF error_return <> ZERO THEN
      BEGIN
      call_name := 'IPCCreate ';
```

```
   Error_Routine( call_name,error_return, call_sd );
   END;

{}
{ The server is waiting on a well-known address (TCP_ADDRESS).  Create the
{ destination descriptor for the socket from the remote node.
{}
proto_addr := TCP_ADDRESS;

IPCDest( socket_kind, node_name, node_name_len, protocol_kind,
   proto_addr, INTEGER_LEN, , , destdesc, error_return );
IF error_return <> ZERO THEN
   BEGIN
   call_name := 'IPCDest   ';
   Error_Routine( call_name,error_return, destdesc );
   END;

{ Now connect to the server }
IPCConnect( call_sd, destdesc, , ,
                       vc_sd, error_return );
IF error_return <> ZERO THEN
   BEGIN
   call_name := 'IPCConnect';
   Error_Routine( call_name,error_return, destdesc );
   END;

{ Set the timeout to infinity with IPCControl for later calls }
control_value := 0;
timeout_len := 2;

IPCControl( vc_sd, CHANGE_TIMEOUT, control_value, timeout_len,
   , , , error_return );

IF error_return <> ZERO THEN
   BEGIN
   call_name := 'IPCControl';
   Error_Routine( call_name,error_return, vc_sd );
   END;

{}
{ Verify the server received the connect request.  Wait for the
{ server to do an IPCRecvCn.
{}
IPCRecv( vc_sd, data_buf, buffer_len, , , error_return );
IF error_return <> ZERO THEN
   BEGIN
   call_name := 'IPCRecv   ';
   Error_Routine( call_name,error_return, vc_sd );
   END;

END;     { SetUp }
```

```
$ TITLE 'ShutdownSockets', PAGE $
PROCEDURE SHUTDOWNSOCKETS;
{------------------------------------------------------------------------}
{ Procedure:  SHUTDOWNSOCKETS                                            }
{                                                                        }
{ Shuts down the VC and call sockets.                                    }
{ Entered after ERROR_ROUTINE completes.                                 }
{------------------------------------------------------------------------}

VAR
   result    :            Integer;

   BEGIN    { ShutdownSockets }
   {}
   { We are terminating this program.  Clean up the allocated
   { sockets.
   {}

   IPCShutdown( vc_sd, , , result );
   { Don't worry about errors here, since there isn't much we can do. }

   IPCShutdown( call_sd, , , result );
   { Don't worry about errors here, since there isn't much we can do. }

   END;     { ShutdownSockets }

$TITLE 'Client MAIN', PAGE $
BEGIN { Client }

{------------------------------------------------------------------------}
{ MAIN                                                                   }
{                                                                        }
{ Prompt user for remote node name.                                      }
{ Create a call socket and connect to the server (SETUP).                }
{ Loop (until user enters 'EOT').                                        }
{   Prompt user for name.                                                }
{   Send user name to Server (IPCSend).                                  }
{   Receive information buffer associated with name from Server          }
{       (RECEIVEDATA).                                                   }
{ Shutdown call and VC sockets (SHUTDOWNSOCKETS).                        }
{------------------------------------------------------------------------}
node_name_len := 0;
requested_name := '';

{ Ask the user for the NS node name of the remote node }
Prompt( 'Client: Enter the remote node name: ' );
Readln( node_name );

temp_position := 1;
GetLen( node_name, temp_position, node_name_len );

SETUP;                { Create a call socket and connect to the server }
```

```
WHILE requested_name <> 'EOT' DO
   BEGIN    { loop for name }

   { Ask the user for a name to be retrieved }
   Prompt( 'Client: Enter name for data retrieval (or EOT to exit): ' );
   Readln( requested_name );
   req_name_len := BUFFERLEN;

   IF requested_name <> 'EOT' THEN
      BEGIN    { continue processing }

      { Ask for the name the user requested }
      IPCSend( vc_sd, requested_name, req_name_len, , ,
            error_return );

      { Get information buffer from server.                   }
      RECEIVEDATA( data_buf );

      { Print out the data received }
      Writeln( 'Client data is: ', data_buf );

      END;      { continue processing }
   END;      { loop for name }
89:

{ Clean up the call and vc sockets }
SHUTDOWNSOCKETS;

99:


END.  { Client }
```

Example 3

Example three includes a pair of programs designated requestor (X25CHECK) and server (X25SERV) using access to X.25 at level 3. These programs will work together on HP 3000 systems over an NS X.25 3000/V Link. The programs' functions are described in the comments included with the program listings. Note that these programs are simplified versions of the programs released with the NS X.25 3000/V link product.

NetIPC Program 5 (X.25 Requestor Program)

```
{

    }


    {*************************************************************}
    {          Declarations for X52CHECK and X25SERVR              }
    {*************************************************************}
    CONST
       c_prot_addr_x25chk = 31000;  {X25CHECK protocol address}
       c_prot_addr_server = 31001;  {X25SERV  protocol address}
               {These decimal addresses are in the range 30767..32767 where PM }
               { is not required }
       c_patern='abcdefghijklmnopqrstuvwxyz0123456789';
       c_buffer_len = 36;
       c_nb_loop =10;
       c_calling_add_code = 141;
       c_prot_add_code    = 128;
       c_net_name_code    = 140;
       c_clear_rcvd       = 67;     {SOCKERR for a CLEAR packet received}


    TYPE
        shint = -32768..32767;
        nibble = 0..15;
        byte  = 0..255;
        rc_type = (done,
                   error,
                   no_vc_desc,
                   no_dest_desc,
                   no_call_desc);

        event_type = (i_addopt,
                      i_create,
                      i_dest,
                      i_connect,
                      i_recv_call_conf,
                      i_send,
                      i_recv,
                      i_shut_source,
                      i_shut_dest,
                      i_shut_connection);
        event_msg_type = array [event_type] of string[80];

        opt_type = packed record                          {             }
                    length : shint;                       {             }
                    num_entries : shint;                  {Declarations}
                    data : packed array [1..256] of shint;{             }
                   end;                                   {     for     }
        buffer_type = string [c_buffer_len] ;             {             }
```

```
                                                              {  NetIPC     }
    socket_type  = (call,destination,vc);                     {             }
    name_type = string [50];                                  {             }
    name_len = shint;


CONST
  c_event_msg = event_msg_type
             ['construction of option record',
              'creation of the local call descriptor',
              'creation of the destination descriptor',
              'CALL packet sending',
              'CALL CONF packet reception',
              'DATA packet sending',
              'DATA packet reception',
              'shutdown of call descriptor',
              'shutdown of destination descriptor',
              'CLEAR packet sending'];




VAR
  rc               : rc_type;
  result           : integer;
  r                : shint;
  p_call_desc      : integer;
  p_vc_desc        : integer;
  p_dest_desc      : integer;
  p_retry          : boolean;
  p_set_up_time    : integer;
  p_transit_time   : integer;


{*****************************************************************}
{*******    Declaration for the NetIPC intrinsics          ******}
{*****************************************************************}

PROCEDURE Addopt       ;INTRINSIC;
PROCEDURE Initopt      ;INTRINSIC;
PROCEDURE Readopt      ;INTRINSIC;

PROCEDURE IPCControl   ;INTRINSIC;
PROCEDURE IPCCreate    ;INTRINSIC;
PROCEDURE IPCDest      ;INTRINSIC;
PROCEDURE IPCConnect   ;INTRINSIC;
PROCEDURE IPCRecvcn    ;INTRINSIC;
PROCEDURE IPCRecv      ;INTRINSIC;
PROCEDURE IPCSend      ;INTRINSIC;
PROCEDURE IPCShutdown  ;INTRINSIC;
PROCEDURE IPCErrmsg    ;INTRINSIC;
PROCEDURE GETPRIVMODE  ;INTRINSIC;
PROCEDURE GETUSERMODE  ;INTRINSIC;
```

```
{******  Other intrinsics used in the programs                  ******}

PROCEDURE quit          ;INTRINSIC;
FUNCTION timer:integer ;INTRINSIC;
{


}
{


}

{*******************************************************************}
{                                                                   }
{ SOURCE       :     CHECK                                          }
{                                                                   }
{ DESCRIPTION  :                                                    }
{  Simplified version.                                              }
{  This program  checks that connections to remote nodes or even    }
{  to local node can  be   actually achieved. It also allows to     }
{  estimate the performances of the network. It communicates with   }
{  program X25SERV that runs on remote nodes.                       }
{  X25CHECK sends 10 times a message to the remote server which     }
{  echoes them back.                                                }
{  It checks for both connection and communication errors.          }
{  This version of X25CHECK is not compatible with the version of   }
{  the product (doesn't work with the official server).             }
{*******************************************************************}


$GLOBAL$
PROGRAM x25chk (input,output);

$include 'decl'$

FUNCTION ask_y_n : boolean;
var
  c : string [1];

begin {ask_y_n}
    repeat
      writeln;
      prompt ('Do you want to run the test once again?(y/n) > ');
      readln (c);
    until (c='y') or (c='Y') or (c='n') or (c='N') or (c='');
    if (c='y') or (c='Y') then ask_y_n := true
    else ask_y_n := false;
end;  {ask_y_n}

PROCEDURE check (result : integer;
                 event  : event_type);
```

```
var
  msg : string [80];
  len : integer;
  r   : integer;


begin   {check}
  IPCErrmsg  (result,msg,len,r);
  setstrlen (msg,len);
  if r <> 0 then
  begin
    writeln ('Can''t get the error message ...');
    QUIT (123);
  end
  else
  begin
    writeln ('An error occured during ',c_event_msg [event]);
    writeln ('with the following identification : ');
    writeln (msg);
    p_retry := ask_y_n;
  end;
end;  {check}


{----------------------INIT_desc-----------------------------------------}
{ Create call descriptor with  dedicated protocol relative address       }
{ Create destination desc   to connect with the server                   }
{------------------------------------------------------------------------}


PROCEDURE init_desc ( var rc : rc_type);

var
  j,
  prot_addr      : shint;
  opt            : opt_type;
  net_name,
  node_name      : string [8];
  net_name_len,
  node_name_len : shint;


begin
                                        {-----------------------------------}
                                        { Creation of the call descriptor.  }
                                        {-----------------------------------}

    Initopt (opt,2,r);
    if r <> 0 then
    begin
       check (r,i_addopt);
       rc   := no_call_desc;
```

```
    end
    else
    begin  {initopt}
      prot_addr := c_prot_addr_x25chk;
      Addopt (opt,0,c_prot_add_code,2,prot_addr,r);
      if r <> 0 then
      begin
         check (r,i_addopt);
         rc  := no_call_desc;
      end
      else
      begin
        prompt('Enter the name of the network you are working on > ');
        readln (net_name);
        net_name_len := strlen(net_name);
        Addopt (opt,1,c_net_name_code,net_name_len,net_name,r);
        if r <> 0 then
        begin
          check (r,i_addopt);
          rc := no_call_desc;
        end
        else
        begin
          IPCCreate (3,2,,opt,p_call_desc,result);
          if result <>0 then
          begin
            check (result,i_create);
            rc := no_call_desc;
          end
          else
          begin
                                          {------------------------------------}
                                          { Creation of the destination desc   }
                                          {------------------------------------}

            writeln;
            prompt ('Enter the name of the node you want to check > ');
            readln (node_name);
            node_name_len := strlen(node_name);
            prot_addr := c_prot_addr_server;
            IPCDest (3,node_name,node_name_len,2,prot_addr,2,,,
                     p_dest_desc,result);
            if result <> 0 then
            begin
               check (result,i_dest);
               rc := no_dest_desc;
            end;{else dest}
          end;{else create}
        end;{else addopt}
      end;{else addopt}
  end;{else initopt}
end;{init_desc}
```

```
{-----------------------------CONNECT------------------------------}
{ Send CALL to the server and wait for CALL CONF                   }
{ Evaluate  the set up time                                        }
{------------------------------------------------------------------}

PROCEDURE connect ( var rc : rc_type);

var
    chrono : integer;

begin

    chrono := timer;
                                       {-----------------------------------}
                                       { Send CALL packet to remote server }
                                       {-----------------------------------}
    IPCConnect (p_call_desc,p_dest_desc,,,p_vc_desc,result);
    if result <> 0 then
    begin
      check (result,i_connect);
      rc := no_vc_desc;
    end
    else
    begin
      writeln ('CALL packet sent ...');
                                       {-----------------------------------}
                                       {Get CALL CONF packet from the server}
                                       {-----------------------------------}
      IPCRecv (p_vc_desc,,,,,result);
      p_set_up_time := timer-chrono;
      if result <> 0 then
      begin
        check (result,i_recv_call_conf);
        rc := error;
      end
      else
      begin
        writeln ('CALL CONF packet received ...');
        writeln;
      end;
                                       {-----------------------------------}
                                       { The connection is now opened.     }
                                       {-----------------------------------}
    end; {else connect}
end; {connect}


PROCEDURE data_transfer ( var rc : rc_type);

var
  buffer        : buffer_type;
  buffer_len    : integer;
```

```
  chrono         : integer;
  i              : shint;



{-------------------------DATA_TRANSFER----------------------------}
{ PURPOSE : Manage  the data transfer with the server              }
{           Evaluate  the transit time                             }
{------------------------------------------------------------------}

begin  {data transfer}
  i := 1;
  chrono := timer;

  while (i <= c_nb_loop) and (rc = done) do
  begin
    buffer     := c_patern;
    buffer_len := c_buffer_len;

                                       {------------------------------------}
                                       { Send data packet on the line.      }
                                       {------------------------------------}
    IPCSend (p_vc_desc,buffer,buffer_len,,,result);
    writeln ('DATA packet sent ...');

    if result <> 0 then
    begin
        check (result,i_send);
        rc := error;
    end
    else
    begin
                                       {------------------------------------}
                                       { Receive data packet echoed by the  }
                                       { remote server.                     }
                                       {------------------------------------}
      IPCRecv (p_vc_desc,buffer,buffer_len,,,result);
      writeln ('DATA packet received ...');
      writeln;
      if result <> 0 then
      begin
        check (result,i_recv);
        rc := error;
      end
      else
        i := i+1;
    end;{else send}
  end;{while}
  p_transit_time := timer - chrono;

end;{data transfer}
```

```
{------------------------SHUTDOWN---------------------------------}
{ PURPOSE : Shutdown call, destination and vc descriptor          }
{           according to the value of rc.                         }
{           Display  the results of set up and transit time       }
{           Ask  to retry                                         }
{-----------------------------------------------------------------}

PROCEDURE shutdown;

begin
    if rc <= error then
    begin
                                                  {-----------------------------------}
                                                  { Shutdown the vc descriptor.       }
                                                  { Send CLEAR on the line.           }
                                                  {-----------------------------------}
      IPCShutdown (p_vc_desc,,,result);
      if result <> 0 then check (result,i_shut_connection);
      writeln ('CLEAR packet sent ...');
    end;

    if rc <= no_vc_desc then
    begin
                                                  {-----------------------------------}
                                                  { Shutdown the destination desc.    }
                                                  {-----------------------------------}
      IPCShutdown (p_dest_desc,,,result);
      if result <> 0 then check (result,i_shut_dest);
    end;
    if rc <= no_dest_desc then
    begin
                                                  {-----------------------------------}
                                                  { Shutdown the call descriptor.     }
                                                  {-----------------------------------}

      IPCSHUTDOWN (p_call_desc,,,result);
      if result <> 0 then check (result,i_shut_source)
    end;

    if rc = done then
    begin
                                                  {-----------------------------------}
                                                  { Display the results.              }
                                                  {-----------------------------------}

  writeln ('The following figures have been measured on the network :');
  writeln ('           Set up  time : ',p_set_up_time:10,' ms');
  writeln ('           Transit time : ',(p_transit_time/(c_nb_loop*2)):10:0,
                                          ' ms');

      p_retry := ask_y_n ;
    end;

end;{shutdown}
```

```
BEGIN
  p_retry := false;
  repeat
    rc := done;
    INIT_DESC (RC);
    if rc = done then
    begin
       CONNECT (rc);
       if rc = done then
       begin
          DATA_TRANSFER (rc);
       end;
    end;
    SHUTDOWN;
  until p_retry = false;
END.

{


}
```

NetIPC Program 6 (X.25 Server Program)

{

```
}
{*****************************************************************}
{                                                                 }
{ SOURCE      :   X25SERV                                         }
{                                                                 }
{ DESCRIPTION :                                                   }
{                                                                 }
{ The purpose of that program is to answer to a remote program    }
{ X25CHEK which verifies that the connections have been actually  }
{ established.                                                    }
{ The server receives messages and echoes them to the remote      }
{ calling node.                                                   }
{ The server has a dedicated protocol relative address.           }
{ This version of X25SERV is not compatible with the version of   }
{ the product.                                                    }
{*****************************************************************}

program x25serv (input,output);
$include 'decl'$ {include file of type and constants}

{---------------------------Check_init----------------------------}
{ PURPOSE : Checks the results of IPC calls. Used during the initi- }
{           alization phase when errors are not discarded but dis-  }
{           played to the operator.                                 }
{                                                                   }
{-------------------------------------------------------------------}

PROCEDURE check_init (result:integer);

VAR
  msg     : string [80];
  msg_len : integer;
  r       : integer;

BEGIN
  if result <> 0 then
  begin
    IPCErrmsg (result,msg,msg_len,r);
    setstrlen(msg,msg_len);
    if r <> 0 then
    begin
      writeln('Can''t get the error message');
      QUIT (123);
    end{if}
    else
    begin
      writeln('X25SERV: error occured during initialization of the');
```

```
      writeln('        server with the following identification:');
      writeln (msg);
      QUIT (125);
    end;{else}
  end;{if}
END;{check_init}


PROCEDURE create_descriptor;
var
  prot_addr     : shint;
  opt           : opt_type;
  net_name      : name_type;
  net_name_len : shint;
  wrtdata       : shint;

begin {create_descriptor}
                                      {------------------------------------}
                                      { Creation of the descriptor dedicated}
                                      { to the server.                      }
                                      {------------------------------------}

  Initopt (opt,2);
  prot_addr := c_prot_addr_server;
  Addopt (opt,0,c_prot_add_code,2,prot_addr,result);
  check_init (result);

  prompt ('Enter the name of the network you are working on > ');
  readln (net_name);
  net_name := strltrim (net_name);
  net_name := strrtrim (net_name); {eliminates blanks}
                                  {usefull when server is run from a stream}
  net_name_len:= strlen (net_name);
  Addopt (opt,1,c_net_name_code,net_name_len,net_name,result);
  check_init(result);

  IPCCreate (3,2,,opt,p_call_desc,result);
  check_init (result);
  writeln('Call descriptor : ',p_call_desc);
                                      {-----------------------------------}
                                      { Disable the timer on the call     }
                                      { descriptor.                       }
                                      {-----------------------------------}

  wrtdata := 0  ;
  IPCControl (p_call_desc,3,wrtdata,2,,,,result);
  check_init (result);

end; {create_descriptor}

PROCEDURE echo;
```

```
var
  opt              : opt_type;
  calling_address : packed array [1..16] of nibble;
  i,
  option_code,
  addr_len,
  data_len         : shint;
  buffer           : buffer_type;
  buffer_len       : integer;

begin {echo}

                                        {-----------------------------------}
                                        { Initialize an option field to get  }
                                        { the calling node address.          }
                                        {-----------------------------------}

  Initopt (opt,1);
  Addopt  (opt,0,c_calling_add_code,8,calling_address,r);

                                        {-----------------------------------}
                                        { Wait for a connection request.     }
                                        { ie Incoming CALL.                  }
                                        {-----------------------------------}
  IPCRecvcn (p_call_desc,p_vc_desc,,opt,result);
  if result = 0 then
  begin
    writeln('Call Received.........');

                                        {-----------------------------------}
                                        { Get the calling address from the   }
                                        { CALL pkt.                          }
                                        {-----------------------------------}
    data_len := 8;
    option_code := c_calling_add_code;
    Readopt (opt,0,option_code,data_len,calling_address,r);
    writeln ('Calling node address = ');
    addr_len := calling_address [1];   {the first nibble contains the addr le
    for i:= 2 to addr_len+1 do write (calling_address [i]:1);
    writeln ;

                                        {-----------------------------------}
                                        { Loop on data transfer.             }
                                        {-----------------------------------}
    i:= 1;
    while (i <= c_nb_loop) and (result = 0) do
    begin
      buffer_len := c_buffer_len;
                                        {-----------------------------------}
                                        { Receive pkt from X25CHECK.         }
                                        {-----------------------------------}
      IPCRecv (p_vc_desc,buffer,buffer_len,,,result);
      if result = 0 then
```

```
      begin
        writeln('Data packet received..........');

                                      {------------------------------------}
                                      { Echo the same buffer.              }
                                      {------------------------------------}
        IPCSend (p_vc_desc,buffer,buffer_len,,,result);
        if result = 0 then i:=i+1;
      end;{if}
    end; {while}
  end;
end;{echo }


PROCEDURE shutdown_connection;
var
  buffer          : buffer_type;
  buffer_len      : integer;

begin
                                      {------------------------------------}
                                      { End of connection.                 }
                                      { Wait for X25CHECK to CLEAR first   }
                                      {------------------------------------}

  if result = 0 then
  begin
    buffer_len := 1;
    IPCRecv (p_vc_desc,buffer,buffer_len,,,result);
                                      {------------------------------------}
                                      { This IPCRECV should complete with  }
                                      { an error indicating a CLEAR recvd. }
                                      {------------------------------------}

    if result = c_clear_rcvd then

                                      {------------------------------------}
                                      { We can shutdown the vc descriptor  }
                                      {------------------------------------}
      IPCShutdown (p_vc_desc,,,result);
  end;

end;{shutdown_connection}

PROCEDURE shutdown_call_desc;
begin {shutdown_call_desc}
  IPCShutdown (p_call_desc,,,result);
end;  {shutdown_call_desc}

begin   {main server}

  CREATE_DESCRIPTOR;
```

```
      while true do  {endless loop}
      begin
        ECHO;
        SHUTDOWN_CONNECTION;
      end;

      SHUTDOWN_CALL_DESC;


    end.  {main server}
    {


}
```

IPC INTERPRETER (IPCINT)

APPENDIX

A

The IPC interpreter (IPCINT) is a software utility provided with the NS X.25 3000/V link product. IPCINT can be used as a learning tool for programmers and as a troubleshooting tool by network administrators.

IPCINT executes NetIPC intrinsics one at a time in response to commands typed at the keyboard. IPCINT can only be used for X.25 direct access to level 3.

Using IPCINT

To use IPCINT you must have an NS X.25 link up and running on a local HP 3000 node and on a remote node. In order to exercise the intrinsics between nodes, the remote node must be running either IPCINT or an X.25 direct access to level 3 server program.

You must have NA or NM capability to run IPCINT. To use IPCINT you enter `RUN IPCINT.NET.SYS` at the MPE-V prompt. At the IPCINT prompt (>) you can enter a NetIPC intrinsic abbreviation or `E` to exit.

You will be prompted for parameters required for the intrinsic you specified. The intrinsic is executed by IPCINT and any output parameters or errors returned are displayed. IPCINT creates a log file called IPCLOG to contain the actions of each intrinsic executed.

Comparison of IPCINT to Programmatic NetIPC

The following examples show the difference between programmatic access and IPCINT used to execute the `IPCCREATE` intrinsic.

Example: Programmatic Access to X.25 Level 3

For a program using direct access to X.25 level 3, a call to `IPCCREATE` can be specified as follows:

```
IPCCREATE (3,2,,opt,calldesc,result)
```

The value 3 for parameter *socketkind* specifies a call socket. The value 2 (for parameter *protocol*) indicates the protocol access is X.25. At a minimum, the *opt* array would contain the X.25 network name, and optionally either define a catch-all socket (*opt* code 144, bit 2) or specify a protocol relative address (*opt* code 128). The *calldesc* will contain the call socket descriptor, and *result* will contain an error (if any).

Example: IPCINT for X.25 Direct Access to Level 3

For example, to execute the IPCCREATE intrinsic using IPCINT, enter CR from the IPCINT prompt (see example below). You are prompted for the IPCREATE X.25 parameters. In this example, no catch-all socket is specified; therefore, a protocol relative address is specified. The network name is a required parameter. The network name X25NET is used in this example. After the required parameters are entered, press (RETURN) and the IPCCREATE intrinsic is executed.

```
> CR
Protocol: 2
Catch All Socket (Y/N)? N
Protocol Relative Address: 31000
Network name (8 chars): X25NET
-----> Executing: IPCCREATE
CALL = 6
```

SYNTAX OF IPCINT

The following paragraphs describe the syntax of IPCINT commands. This includes:

- Abbreviations for the intrinsics.
- Pseudovariables for socket descriptors.
- Prompts for parameters.
- Call user data field.

Abbreviated Intrinsic Names

The IPCINT program uses abbreviations for NetIPC intrinsics. Table A-1 shows the supported IPC intrinsics and the IPCINT abbreviations.

TABLE A-1. NetIPC Intrinsics IPCINT Abbreviations

Intrinsic	IPCINT Abbreviation
IPCCREATE	CR
IPCNAME	NAME
IPCNAMERASE	NAMERASE
IPCDEST	DEST
IPCGIVE	GIVE
IPCGET	GET
IPCCONNECT	CN
IPCRECVCN	RCN
IPCRECV	R
IPCSEND	S
IPCCONTROL	CTR
IPCSHUTDOWN	SHUT
IOWAIT	WAIT
IODONTWAIT	NOWAIT
IPCCHECK	CHECK
IPCERRMSG	ERR

Pseudovariables

Three pseudovariables are used by IPCINT to store the most recently assigned socket descriptors as follows:

```
Pseudovariable      socket descriptor
--------------      -----------------
      C                call
      D                destination
      V                virtual circuit
```

The pseudovariable names can be overidden by the user.

Prompts for Parameters

When you enter the IPCINT abbreviation for the selected intrinsic, IPCINT prompts you for the required parameter values which you then enter from the keyboard. Default values are provided for most input parameters. The parameter names correspond approximately to those used in the reference portion of this manual. IPCINT prompts for X.25 *opt* array parameters without your having to use the INITOPT or ADDOPT intrinsics. You are also prompted for X.25 *flags* parameter bit settings. Prompts requiring a Y/N (yes/no) answer default to N (no).

Output parameters are displayed on the screen following execution of the called intrinsic.

Call User Data Field

The call user data field can be entered as a concatenated ASCII string enclosed in single quotes. Hexadecimal digits can be included in an ASCII string by preceding the digits with an h. For example, h45'hello'h10 can be entered which represents a string of hexadecimal 45, the word "hello" followed by hexadecimal 10.

SAMPLE IPCINT SESSION

The following example describes the steps to create a call socket, send and receive data over a connection, and then close the socket using IPCINT on a local node. This sample session assumes a remote node is also using IPCINT. The remote node running IPCINT sends the local node a message as described in step 7.

The steps below follow the SVC requestor processing example in Figure 1-8 (Section 1). The remote node should follow the steps in the SVC server processing example in Figure 1-9 (Section 1).

User input is underlined in the examples provided. For detailed information about NetIPC intrinsic parameters refer to the intrinsic descriptions in Section 2. Intrinsic parameter names that differ from the names used as prompts in IPCINT are included in parentheses in the discussion of the examples.

Step 1

Run the IPCINT program from the MPE-V prompt. A log of the session will be written to a file named IPCLOG.

```
(1)     :RUN IPCINT.NET.SYS
        IPCINT (A.01.04) (c) COPYRIGHT Hewlett-Packard Company 1988.
        > > > > IPC Interpreter
```

To exit IPCINT at any time enter E at the IPCINT prompt (>).

Step 2

Enter the IPCINT abbreviation for the desired intrinsic (see Table A-1). In this example, CR for IPCCREATE is entered.

You are prompted for all required input parameters. You must enter 2 for X.25 direct access at the `Protocol` prompt. In this example, enter Y (yes) to create a catch-all socket (*opt* code 144, bit 2). Enter the network name configured for your network at the `Network name` (*opt* code 140) prompt.

After entering all required parameters, the intrinsic is executed. The call socket descriptor (*calldesc*) is returned in the pseudovariable "C".

The output parameters are interepreted and displayed. In this example, a call socket has been created as a catch-all socket.

```
(2)     > CR
        Protocol: 2
        Catch All socket (Y/N)? Y
        Network name (8 chars): X25net
        -----> Executing  : IPCCREATE
        CALL =          6
        Catch All socket
```

Step 3

Execute the IPCDEST intrinsic by entering DEST at the prompt. You are prompted for the remote Node name (*location*) where the destination socket will be created. In this example, RAINBOW is used. Enter a protocol relative address (*protoaddr*) in the decimal range 30767 to 32767 for the remote address. In this example, 7000 is used. The IPCDEST intrinsic is executed and a destination descriptor (*destdesc*) will be returned in pseudovariable "D".

```
(3)     > DEST
        Node name (50 chars): RAINBOW
        Protocol relative address (30767..32767): 31000
        -----> Executing  : IPCDEST
        DEST = - 1
```

Step 4

In order to execute this step, the remote node server program or IPCINT must have already executed, an IPCCREATE followed by an IPCRECVCN. The remote waits for the local to send the connection request. IPCINT provides a timeout so the IPCRECVCN will not wait indefinitely.

Execute IPCCONNECT by entering CN at the prompt. You are prompted for the call socket descriptor. To use the default, press RETURN which is the value returned in pseudovariable "C" by the previous call to IPCCREATE.

You are prompted for the destination socket descriptor. To use the default, press RETURN which is the value returned in pseudovariable "D" by the previous call to IPCDEST.

You are prompted for access to the call user data (CUD) field (*opt* 144, protocol flags, bit 17). In this example, Y (yes) is entered. Selecting "yes" allows you to enter up to 16 bytes of user data at the 16 bytes of CUD prompt (*opt* code 2).

Next, you are prompted for a facility set name (*opt* code 142). To use the default configured for your network, press RETURN. The IPCCONNECT intrinsic is executed and a virtual socket descriptor is returned.

In the example, the statement, "No address in CUD" refers to the fact that you requested full access to the CUD.

```
(4)     > CN
        Call socket desc (32 bit integer /C/D/V): RETURN
        Destination socket desc (32 bit integer /C/D/V): RETURN
        Full access to CUD (Y/N)? Y
        16 bytes of CUD (ascii '',hexa: hFC...):hFCAA0001
        Facility name (8 chars): RETURN
        -----> Executing  : IPCCONNECT
        VC =           7
        No address in CUD
```

Step 5

Execute IPCRECV by entering R at the prompt to receive the response to the previous connection request.

The default value for the VC socket descriptor is the value returned in the last IPCCONNECT (or in the case of an incoming call, by IPCRECVCN). This value is the default for any subsequent IPCSEND or IPCRECV calls.

To use default values, press RETURN. Buffer length (*dlen*) defaults to 4096 bytes. Preview data and Destroy data (*flags* 30 and 29) default to no (N). Data offset (*opt* code 8) is defaulted to none.

```
(5)     > R
        VC socket desc (32 bit integer /C/D/V): RETURN
        Buffer length (bytes): RETURN
        Preview data (Y/N)?    RETURN
        Destroy data (Y/N)?    RETURN
        Data offset (bytes): RETURN
        -----> Executing  : IPCRECV
        MAX_LEN = 4096
        RECV_LEN = 0
        BUFFER = ''
```

Note that there is no data returned in "Buffer" because the function of this call to IPCRECV is to accept the connection request from the remote node.

Step 6

Execute a call to IPCSEND by entering S at the prompt.

Enter a value for the buffer length. IPCINT will send a string of characters equal to the number of bytes specified. If you enter 0 for buffer length, you will be prompted to enter the contents of the data you are sending. You can specify up to 80 characters of data. At the Buffer prompt enter the data to send. In this example, 'Hello from local' is entered.

Pressing RETURN at the VC socket desc prompt which defaults to the VC socket descriptor returned by the previous call to IPCCONNECT (in this example). To use default values, press RETURN. Q bit set and D bit set (*opt* code 144, bit 19 and bit 18) are defaulted to no (N). Data offset (*opt* code 8) defaults to none.

```
(6)    > S
        Buffer length (bytes): 0
        Buffer (ascii:'',hexa;hFC...): 'Hello from local'
        VC socket desc (32 bit integer /C/D/V): RETURN
        Q bit set (Y/N): RETURN
        D bit set (Y/N)? RETURN
        Data offset (bytes): RETURN
        -----> Executing  : IPCSEND
```

In order for the remote node to receive the sent data, an IPCRECV must be executed from the remote node with IPCINT (or a server program).

Step 7

Before executing step 7, the remote must execute IPCSEND to send data to the local node (see step 6, IPCSEND).

Execute IPCRECV to receive data by entering R at the prompt. Step 7 assumes a remote node using IPCINT has sent you a message.

Press RETURN to use the default VC socket descriptor (*vcdesc*). To use default values, press RETURN. Buffer length (*dlen*) defaults to 4096 bytes. Preview data and Destroy data (*flags* 30 and 29) default to no (N). Data offset (*opt* code 8) is defaulted to none.

Values returned by IPCRECV include data sent from the remote displayed at the prompt: Buffer = (*data*), length of the received data (*dlen*), and the buffer length input displayed as MAX_LEN (*dlen*, from input).

```
(7)    > R
       VC socket desc (32 bit integer /C/D/V): RETURN
       Buffer length (bytes): RETURN
       Preview data (Y/N)?    RETURN
       Destroy data (Y/N)?    RETURN
       Data offset (bytes): RETURN
       -----> Executing  : IPCRECV
       MAX_LEN = 4096
       RECV_LEN = 17
       BUFFER = 'Hello from remote'
```

Step 8

Execute IPCSHUTDOWN to shutdown the socket by entering SHUT at the prompt.

At the descriptor prompt, enter a descriptor (C, D or V) in order to indicate which socket needs to be shutdown. In this example, the VC socket descriptor, V is entered.

You are prompted for a reason code (*opt* code 143). In this example, 100 is entered which will cause a clear packet to be sent. The clear packet will contain a cause code zero (0), and diagnostic code 100. (IPCCONTROL is used to access cause and diagnostic codes.)

```
(8)    > SHUT
       Descriptor (32 bit integer /C/D/V): V
       Reason code (16 bit decimal): 100
       -----> Executing  : IPCSHUTDOWN
```

Step 9

Exit from the IPCINT program by entering E at the prompt.

```
(9)     > E
```

Cause and Diagnostic Codes

APPENDIX **B**

Cause and diagnostic codes can be inserted and read from X.25 packets using NetIPC intrinsics. The following tables show possible cause and diagnostic codes generated by NS X.25 3000/V which is a subset of the CCITT (1980 X.25 recommendation) specified value.

CAUSE CODES

If NS X.25 3000/V is configured as a DTE, the cause code will always be set to zero (0). If the node is configured as a DCE, the following tables show the values included in restart, clear and reset packets.

Table B-1. Cause codes for Restart Packets

Cause Code	Meaning
1	Local procedure error
7	Network operational

Table B-2. Cause Codes for Clear Packets

Cause Code	Meaning
1	Number busy
3	Invalid facility request
19	Local procedure error
25	Reverse charging acceptance not subscribed
41	Fast select acceptence not subscribed

Table B-3. Cause Codes for Reset Packets

Cause Code	Meaning
5	Local procedure error

DIAGNOSTIC CODES IN X.25 CLEAR PACKETS

The following lists the diagnostic codes sent and received in X.25 clear packets. The IPCCONTROL intrinsic can be used to insert cause and diagnostic codes that will be included in clear packets sent by the X.25 protocol. You can include diagnostic codes with the IPCSHUTDOWN intrinsic that will be included in the clear packet sent by the X.25 protocol. This function is only available with SVCs.

TABLE B-4. X.25 DIAGNOSTIC CODES SENT/RECEIVED IN CLEAR PACKETS

Diagnostic Code	Meaning/Cause
0	No additional information
1	Invalid P(S)
2	Invalid P(S)
16	Invalid packet type
17	Invalid packet type for state R1
18	Invalid packet type for state R2
19	Invalid packet type for state R3
20	Invalid packet type for state P1
21	DTE received an unexpected packet while waiting for a CALL CONF. (Invalid packet type for state P2.)
22	DCE received an unexpected packet while waiting for a CALL CONF. (Invalid packet type for state P3.)
23	Invalid packet type for state P4

TABLE B-4. X.25 DIAGNOSTIC CODES SENT/RECEIVED IN CLEAR PACKETS (cont'd)

Diagnostic Code	Meaning/Cause
24	Unexpected packet or CALL packet received in state P5 (after a CALL COLLISION occurred.
25	Invalid packet type for state P6
27	Invalid packet type for state D1
28	Invalid packet type for state D2
29	Invalid packet type for state D3
32	Packet not allowed
33	Unidentifiable packet
34	An incoming CALL was received on a one-way outgoing SVC.
35	Packet type invalid on a PVC
36	Packet on an unassigned logical channel
37	Reject not supported
38	Calling address length was too short in received CALL packet.
39	A CALL packet was received that was greater than the valid length. Call user data field was too long or fast select was requested.

TABLE B-4. X.25 DIAGNOSTIC CODES SENT/RECEIVED IN CLEAR PACKETS (cont'd)

Diagnostic Code	Meaning/Cause
40	D bit facility requested but not configured.
41	Restart with non-zero in bits 1-4, 9-16
43	Unauthorized interrupt confirmation
44	Unauthorized interrupt
48	Timer expired
49	Timer expired for incoming call
50	Timer expired for clear indication
51	Timer expired for reset indication
52	Timer expired for restart indication
64	(1) Invalid facility field length. : Either too short or does not match the buffer length. (2) Facility set was not found in path table or in facility tables. (3) Access not allowed because of LUG. (4) No free entry was found in connection table. (5) DCE rejected the CALL because it detected a CALL COLLISION.

TABLE B-4. X.25 DIAGNOSTIC CODES SENT/RECEIVED IN CLEAR PACKETS (cont'd)

Diagnostic Code	Meaning/Cause
65	(1) The facility requested is not supported or allowed here : • Reverse charge in CALL CONF packet. • Fast select. • Throughput class negotiation (not configured). • Closed User Group facility in CALL CONF packet (not allowed). • Bilateral closed user group (not supported). • Packet size negotiation (not configured). • Window size negotiation (not configured). • RPOA facility (not supported). (2) Invalid facility code used.
66	(1) Invalid facility length. (2) Value is out of range in facilities for window size, packet size, or throughput class. (3) Reverse charging is requested but not configured.
67	Invalid BCD digit in called address field.
68	Invalid BCD digit in calling address field.
69	Facility field too long (> 63 bytes).

Error Messages

APPENDIX C

This appendix includes the mapping of X.25 SOCKERRs to protocol module errors, and the complete table of possible NetIPC errors (SOCKERRs).

X.25 Direct Access SOCKERR to PMERR Mapping

In the IPCCHECK intrinsic, both socket errors (SOCKERRs) and the corresponding protocol module errors (*pmerrs*) are returned. the following SOCKERRs are mapped to *pmerrs*. Other SOCKERRs can be returned to NetIPC with a corresponding *pmerr* of zero (0).

SOCKERR 46 : UNABLE TO INTERPRET RECEIVED PATH REPORT.

PMERR = 5 Intrinsics : IPCConnect

Cause : The address key corresponding to the remote node name in the
network directory has not been found in the path tables.
Action : Check consistency between configuration file and network directory.

PMERR = 41 Intrinsics : IPCConnect

Cause : The address key corresponding to the remote node does not
belong to the network the IPCCreate has been issued against.
Action : Check configuration or issue IPCCreate on the correct network.

SOCKERR 50 : INVALID DATA LENGTH.

PMERR = 20 Intrinsics : IPCSend

Cause : The requested send length has been found invalid.
Action : Verify that the buffer length matches the requested length.
The length cannot be equal to 0.

SOCKERR 54 : INVALID CALL SOCKET DESCRIPTOR.

PMERR = 39 Intrinsics : IPCShutdown (call socket descriptor)

Cause : Attempted release of a non-existent call
socket.
Action : Note the running environment and submit an SR.

SOCKERR 55 : EXCEEDED PROTOCOL MODULE'S SOCKET LIMIT.

PMERR = 1 Intrinsics : IPCCreate

Cause : All call socket entries in the X.25 internal tables are in use.
Action : Remember to release call sockets when no IPCConnect and IPCRecvcn are expected .

PMERR = 45 Intrinsics : IPCConnect

Cause : All connection entries in X.25 internal tables are in use.
Action : Remember to shut the VC's that are no longer in use.

SOCKERR 59 : SOCKET TIMEOUT.

PMERR = 33 Intrinsics : IPCControl

Cause : The reset timer expired before a reset confirmation packet was received.
Action : None. Informative .

PMERR = 34 Intrinsics : IPCControl

Cause : The interrupt timer expired before an interrupt confirmation packet was received.
Action : None. Informative .

SOCKERR 65 : CONNECTION ABORTED BY LOCAL PROTOCOL MODULE.

PMERR = 21 Intrinsics : IPCShutdown (Virtual circuit descriptor)

Cause : The X.25 level 3 virtual circuit already was cleared when the NetIPC call was issued.
Action : No action. The virtual circuit socket has been properly closed.

PMERR = 36 Intrinsics : IPCRecv, IPCSend, IPCControl

Cause : The inactivity timer has timed out.
Action : Shutdown the connection before re-opening it.

SOCKERR 66 : INVALID CONNECTION DESCRIPTOR.

PMERR = 38 Intrinsics : IPCShutdown (Virtual circuit descriptor)

Cause : An attempt has been done to release a non-existent virtual circuit socket.
Action : Note the running environment and submit an SR.

SOCKERR 67 : CONNECTION FAILURE DETECTED.

PMERR = 2 Intrinsics : IPCSend, IPCRecv, IPCControl

Cause : A clear packet has been received. The remote system or network aborted the connection.
Action : Retrieve the cause/diagnostic field with IPCControl, and issue IPCShutdown on the virtual circuit.

SOCKERR 106 : ADDRESS CURRENTLY IN USE BY ANOTHER SOCKET.

PMERR = 4 Intrinsics : IPCCreate

Cause : The requested protocol relative address is already used by another process through another IPCCreate call.
Action : Use another protocol relative address or wait for previous process to release its call socket.

SOCKERR 107 : TRANSPORT IS GOING DOWN.

PMERR = 7 Intrinsics : IPCRecvcn

Cause : A NETCONTROL STOP command has been issued. All call sockets must be shut.
Action : Issue an IPCShutdown on call socket.

PMERR = 8 Intrinsics : IPCRecv, IPCSend, IPCControl

Cause : A NETCONTROL STOP has been issued; the X.25 protocol module is not in a state to accept any request.
Action : Issue an IPCShutdown on virtual circuit socket.

PMERR = 9 Intrinsics : IPCCreate

Cause : The X.25 protocol module is not in a state to accept the creation of new call sockets because a NETCONTROL STOP command has ben issued.
Action : Issue an IPCShutdown on the call socket.

SOCKERR 111 : INTERNAL SOFTWARE ERROR DETECTED.

PMERR = 40   Intrinsics : IPCConnect

Cause : Internal error. (Unable to allocate a receive list)
Action : Submit an SR.

PMERR = 48   Intrinsics : IPCControl, IPCConnect, IPCRecv, IPCSend

Cause : Internal error.
Action : Critical X.25 error. Submit an SR.

SOCKERR 116 : DESTINATION UNREACHABLE.

PMERR = 13   Intrinsics : IPCConnect

Cause : The facility set associated with address key has not been found in the internal tables.
Action : Verify that the correspondance between address keys and facility sets in the configuration file is correct.

PMERR = 17   Intrinsics : IPCConnect

Cause : Outgoing access not allowed. Some X.25 address or address keys are configured in the Outgoing LUG table in confirmation files, and requested outgoing address has not been found in the table.
Action : Check configuration of Outgoing Local User Group if necessary.

SOCKERR 117 : ATTEMPT TO ESTABLISH CONNECTION FAILED.

PMERR = 19   Intrinsics : IPCRecv completing IPCConnect

Cause : The virtual circuit failed to be opened. On receipt of the call configuration packet, a clear packet has been sent by the local system.
Possible causes are :

1. Incompatible facilities with the other end.
2. No call confirmation has been received within allowed timeframe, causing a clear packet to be sent.
3. A reset packet was received instead of a call confirmation. This may be due to confirmation problem (PVC and SVC mapping error)

Action : Issue IPCShutdown on virtual circuit, correct cause if necessary and re-issue IPConnect.

PMERR = 35 Intrinsics : IPCRecv completing IPCConnect (on DTE)

Cause : No call confirmation has been received within allowed timeframe. A clear packet had to be sent that has not been answered by a clear confirmation. The clear timer expired twice.
Action : Check connection to the remote node.

PMERR = 37 Intrinsics : IPCRecv completing IPCConnect (on DCE)

Cause : No call confirmation has been received within allowed timeframe. A clear packet had to be sent that has not been answered by a clear confirmation. The clear timer expired twice. The local side sent a diagnostic packet.
Action : Check connection to the remote node.

SOCKERR 143 : INVALID FACILITIES SET OPT RECORD ENTRY.

PMERR = 14 Intrinsics : IPCConnect

Cause : The facility set passed as a parameter has not been found in the internal facility table for an SVC.
Action : Use SVC facility sets defined in configuration.

PMERR = 15 Intrinsics : IPCConnect

Cause : The facility set passed as a parameter has not been found in the internal facility table for a PVC.
Action : Use PVC facility sets defined in configuration.

SOCKERR 146 : RESET EVENT OCCURRED ON X.25 CONNECTION.

PMERR = 10 Intrinsics : IPCSend

Cause : A reset packet was sent internally because of an internal error or because of resource shortage (mainly buffers).
Action : Re-issue call if necessary. Check buffer usage. Adjust buffer confirmation to usage.

PMERR = 11 Intrinsics : IPCSend, IPCRecv

Cause : An unsolicited reset packet was received.
Action : Use IPCCONTROL (request 12) to retrieve the cause/diagnostic field.

SOCKERR 153 : SOCKET IS ALREADY IN USE.

PMERR = 3   Intrinsics : IPCCreate

Cause : A single socket per network interface can be created
with the catch-all capability.
Action : Wait for catch-all socket to be released.

SOCKERR 156 : INTERRUPT EVENT OCCURRED ON X.25 CONNECTION.

PMERR = 12   Intrinsics : IPCRecv, IPCSend

Cause : An interrupt packet was received.
Action : Use IPCCONTROL (request 12) to get interrupt data.

SOCKERR 157 : ALL OUTGOING SWITCHED VIRTUAL CIRCUITS ARE BUSY.

PMERR = 16   Intrinsics : IPCConnect

Cause : No more free LCN one-way outgoing or two-ways SVC.
Action : Wait for LCN to be freed and re-issue call.

SOCKERR 158 : CONNECTION REQUEST REJECTED BY REMOTE.

PMERR = 18   Intrinsics : IPCRecv

Cause : The outgoing call packet has been answered by a clear packet.
Action : Use IPCCONTROL (request 12) to retrieve the cause/diagnostic field.
Take action depending on cause/diagnostic using table given.

SOCKERR 159 : INVALID X.25 D BIT SETTING.

PMERR = 22 Intrinsics : IPCSend

Cause : User requested an X.25 packet to be sent with the D-bit set
while the facility set in use does not allow it.
Action : Use a facility set allowing D-bit usage.

SOCKERR 160 : INCOMPATIBLE WITH PROTOCOL STATE.

PMERR = 24 Intrinsics : IPCSend

Cause : Data cannot be sent on the line. A reset packet had been issued by local protocol module, and the reset confirmation packet has not yet been received.
Action : Wait then re-issue the call if necessary.

PMERR = 25 Intrinsics : IPCSend

Cause : Data cannot be sent on the line. A reset request packet has been received and the protocol module is waiting on the IPC user response to generate the reset confirmation packet.
Action : Complete pending IPCSend(s) if any, or issue an IPCReceive to complete the reset/reset configuration sequence.

PMERR = 26 Intrinsics : IPCControl (interrupt)

Cause : Interrupt packets cannot be sent on the line. A reset packet has been issued by the local protocol module, and the reset configuration packet has not yet been received.
Action : Issue an IPCRecv to complete the reset/reset configuration sequence.

PMERR = 27 Intrinsics : IPCControl (interrupt)

Cause : An interrupt packet is not allowed in the current state.
Action : Wait for the protocol module to be in a proper state.

PMERR = 28 Intrinsics : IPCControl (reset)

Cause : A reset packet is not allowed in the current state.
Action : Wait for the protocol module to be in a proper state.

PMERR = 31 Intrinsics : IPCConnect

Cause : An IPC connection request is invalid in the current state : level 2 is down or level 3 is not established.
Action : Wait for level 3 to be ready again.

SOCKERR 162 : X.25 PERMANENT VIRTUAL CIRCUIT DOES NOT EXIST.

PMERR = 47   Intrinsics : IPCConnect

Cause : The PVC was not found.
Action : Check if the PVC is configured.

SOCKERR 163 : PERMANENT VIRTUAL CIRCUIT ALREADY ESTABLISHED.

PMERR = 32   Intrinsics : IPCConnect

Cause : This PVC is already established.
Action : Wait for owner to close the PVC before using it again.

SOCKERR 168 : RESTART EVENT OCCURRED ON X.25 CONNECTION.

PMERR = 43   Intrinsics : IPCSend, IPCRecv, IPCControl

Cause : Connection has been aborted because a restart packet was received.
Action : Issue an IPCShutdown to shut the virtual circuit.

PMERR = 44   Intrinsics : IPCSend, IPCRecv, IPCControl

Cause : A restart packet has been sent by the local protocol module.
Action : Issue an IPCShutdown to shut the virtual circuit and
wait for the restart procedure to complete.

NETWORK INTERPROCESS COMMUNICATION ERRORS (SOCKERRS)

NetIPC errors are (32-bit) integers that are returned in the *result* parameter of NetIPC intrinsics when the intrinsic execution fails. (A result of 0 indicates that the intrinsic succeeded.) In addition, NetIPC errors and Transport Protocol (Transmission Control Protocol, and X.25 protocol) errors are returned in the IPCCHECK intrinsic: NetIPC errors in the *ipcerr* parameter and Transport Protocol errors in the *pmerr* parameter.

"Submitting an SR" (service request) is documented in the *NS3000/V Error Message and Recovery Manual.*

NetIPC ERRORS (SOCKERRS)

Message		Cause	Action
0	SUCCESSFUL COMPLETION. (SOCKERR 0)	No error was detected.	None.
1	INSUFFICIENT STACK SPACE. (SOCKERR 1)	Area between S and Z registers is not sufficient for execution of the intrinsic.	:PREP your program file with a greater MAXDATA value.
3	PARAMETER BOUNDS VIOLATION. (SOCKERR 3)	A specified parameter is out of bounds.	Check all parameters to make certain they are between the user's DL and S registers. If an array is specified, make certain all of it is within bounds.
4	TRANSPORT HAS NOT BEEN INITIALIZED. (SOCKERR 4)	A :NETCONTROL was not issued to bring up the transport.	Notify your operator.
5	INVALID SOCKET TYPE. (SOCKERR 5)	Specified socket type parameter is of an unknown value.	Check and modify your socket type parameter.
6	INVALID PROTOCOL. (SOCKERR 6)	Specified protocol parameter is of an unknown value.	Check and modify protocol parameter.
7	ERROR DETECTED IN *flags* PARAMETER. (SOCKERR 7)	An unsupported bit in the *flags* parameter was set, or a nonprivileged user set a privileged bit.	Make certain the bit is off before calling the intrinsic.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
8 INVALID OPTION IN THE *opt* RECORD. (SOCKERR 8)	An unsupported option was specified in the *opt* record, or a nonprivileged user attempted to specify a privileged option.	Check your *opt* record and remove or modify the option.
9 PROTOCOL IS NOT ACTIVE. (SOCKERR 9)	A :NETCONTROL has not been issued to activate the requested protocol module.	Notify your operator.
10 PROTOCOL DOES NOT SUPPORT THE SPECIFIED SOCKET TYPE. (SOCKERR 10)	The type of socket you are trying to create is not supported by the protocol to be used.	Use a different socket type or protocol.
13 UNABLE TO ALLOCATE AN ADDRESS. (SOCKERR 13)	No addresses were available for dynamic allocation.	Wait a while and try again. See "Submitting an SR".
14 ADDRESS OPTION ERROR. (SOCKERR 14)	The address option in the *opt* record has an error in it (e.g., invalid length or is in the privileged range).	Check the values being placed in the *opt* record.
15 ATTEMPT TO EXCEED LIMIT OF SOCKETS PER PROCESS. (SOCKERR 15)	User has already reached the limit of 64 sockets per process.	Shut down any sockets which are not being used or have been aborted.
16 PATH DESCRIPTORS OR PATH DESCRIPTOR EXTENSIONS UNAVAILABLE. (SOCKERR 16)	1. Transport's path cache or path descriptor table is full. 2. Network Interface (NI) was not started. 3. IP address is incorrect either in the network directory, or the routing information in the configuration file.	1. Contact your operator to see if the table can be expanded. 2. Start the NI. 3. Use NMMGR to correct the network directory or configuration file. (After correcting the configuration file you must issue a :NSCONTROL UPDATE.)

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
18 FORMAT OF THE *opt* RECORD IS INCORRECT. (SOCKERR 18)	NetIPC was unable to parse the specified *opt* record.	Check your INITOPT and ADDOPT calls.
19 ERROR DETECTED WITH MAXIMUM MESSAGE SIZE OPTION. (SOCKERR 19)	Maximum message size option in the *opt* record had an error associated with it (e.g., too many bytes specified, invalid message size value).	Check the values being placed in the *opt* record.
20 ERROR WITH DATA OFFSET OPTION. (SOCKERR 20)	Data offset option in the *opt* record had an error associated with it (e.g., too many bytes specified).	Check the values being placed in the *opt* record.
21 DUPLICATE *opt* RECORD OPTION SPECIFIED. (SOCKERR 21)	The same *opt* record option was specified twice.	Remove the redundant call.
24 ERROR DETECTED IN MAXIMUM CONNECTION REQUESTS QUEUED OPTION. (SOCKERR 24)	Maximum connection requests queued option in the *opt* record had an error associated with it (e.g., too many bytes specified, bad value).	Check the values being placed in the *opt* record.
25 SOCKETS NOT INITIALIZED; NO GLOBAL DATA SEGMENT. (SOCKERR 25)	Error occurred attempting to initialize NetIPC, or network management is still initializing.	Try again. If it still fails, see "Submitting an SR".
26 UNABLE TO ALLOCATE A DATA SEGMENT. (SOCKERR 26)	The attempt to create a data segment failed because the DST table was full or there was not enough virtual memory.	Contact your operator to see if these tables can be expanded.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
27 REQUIRED PARAMETER NOT SPECIFIED. (SOCKERR 27)	A required parameter was not supplied in an option variable intrinsic call.	Check your calling sequence.
28 INVALID NAME LENGTH. (SOCKERR 28)	Specified name length was too large or negative.	Check your name length parameter. Shorten the name if necessary.
29 INVALID DESCRIPTOR. (SOCKERR 29)	Specified descriptor is not a valid socket, connection, or destination descriptor.	Check the value being specified.
30 UNABLE TO NAME CONNECTION SOCKETS. (SOCKERR 30)	The socket descriptor given in the IPCNAME call was for a VC socket; VC sockets may not be named.	Check if the correct descriptor was specified.
31 DUPLICATE NAME. (SOCKERR 31)	Specified name was previously given.	Use a different name.
32 NOT CALLABLE IN SPLIT STACK. (SOCKERR 32)	The particular NetIPC intrinsic cannot be called from split stack.	Recode to call the intrinsic from the stack. Vectored data may be required.
33 INVALID NAME. (SOCKERR 33)	Name is too long or has a negative length.	Check the name's length. Shorten the name if necessary.
34 CRITICAL ERROR PREVIOUSLY REPORTED; MUST SHUTDOWN SOCKET. (SOCKERR 34)	NetIPC previously detected and reported an irrecoverable error; most likely it was initiated by the protocol module.	The socket can no longer be used. Call IPCSHUTDOWN to clean up.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
35 ATTEMPT TO EXCEED LIMIT OF NAMES PER SOCKET. (SOCKERR 35)	A socket can have only four names; the caller attempted to give it a fifth.	Use no more than four names.
36 TABLE OF NAMES IS FULL. (SOCKERR 36)	Socket registry or give table is full.	Shut down unused sockets, call IPCNAMERASE on any sockets that no longer need to be looked up, or get given sockets. See if the operator can configure more PCBs. See "Submitting an SR".
37 NAME NOT FOUND. (SOCKERR 37)	Name was not previously specified in an IPCNAME or IPCGIVE call; IPCNAMERASE or IPCGET was previously issued with the name; or socket no longer exists.	Check names specified, make sure names were properly agreed on, determine if a timing problem exists.
38 USER DOES NOT OWN THE SOCKET. (SOCKERR 38)	Attempted to erase a name of a socket you do not own.	Have the owner of the socket call IPCNAMERASE.
39 INVALID NODE NAME SYNTAX. (SOCKERR 39)	Syntax of the node name is invalid.	Check the node name being supplied.
40 UNKNOWN NODE. (SOCKERR 40)	Unable to resolve the specified node name as an NS node name.	Check the node name to see if it is correct. The node name may be valid but the specified node's transport may not be active.
41 ATTEMPT TO EXCEED PROCESS LIMIT OF DESTINATION DESCRIPTORS. (SOCKERR 41)	User has already reached the limit of 261 destination descriptors per process.	Call IPCSHUTDOWN on any unneeded destination descriptors.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
43 UNABLE TO CONTACT THE REMOTE REGISTRY SERVER. (SOCKERR 43)	Send to remote socket registry process failed. This is often caused by the fact that the PXP protocol module is not active on the local node.	Contact your operator. If unable to resolve the problem, see "Submitting an SR".
44 NO RESPONSE FROM REMOTE REGISTRY SERVER. (SOCKERR 44)	No reply was received from the remote registry process. This is often due to the remote node not having initialized its tranport.	Contact your operator. If unable to resolve the problem, see "Submitting an SR".
46 UNABLE TO INTERPRET RECEIVED PATH REPORT. (SOCKERR 46)	Unable to interpret the information returned by the remote socket registry process regarding the looked-up socket.	See "Submitting an SR".
47 INVALID MESSAGE RECEIVED FROM REMOTE SERVER. (SOCKERR 47)	The message received from the remote registry process does not appear to be a valid socket registry message.	See "Submitting an SR".
50 INVALID DATA LENGTH. (SOCKERR 50)	Specified data length parameter is too long or negative.	Check and modify the value.
51 INVALID DESTINATION DESCRIPTOR. (SOCKERR 51)	Supplied destination descriptor value is not that of a valid destination descriptor.	Verify that you are passing an active destination descriptor.
52 SOURCE AND DESTINATION SOCKET PROTOCOL MISMATCH. (SOCKERR 52)	The source socket is not of the same protocol as the socket described by the destination descriptor.	Validate that you are using the correct destination descriptor. Make certain both processes have agreed on the same protocol. Determine the correct socket was looked up.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
53 SOURCE AND DESTINATION SOCKET TYPE MISMATCH. (SOCKERR 53)	The source socket cannot be used for communication with the socket described by the destination descriptor.	Validate that you are using the correct destination descriptor. Make certain both processes have agreed on the same method of communication. Determine the correct socket was looked up.
54 INVALID CALL SOCKET DESCRIPTOR. (SOCKERR 54)	Specified descriptor is not for a call socket.	Validate the value being passed.
55 EXCEEDED PROTOCOL MODULE'S SOCKET LIMIT. (SOCKERR 55)	Protocol module being used cannot create any more sockets.	Contact your operator; the limit may be configurable.
57 ATTEMPT TO EXCEED LIMIT OF NOWAIT SENDS OUTSTANDING. (SOCKERR 57)	User tried to send data too many times in nowait mode without calling IOWAIT.	Call IOWAIT to complete a send. The limit is 7.
58 ATTEMPT TO EXCEED LIMIT OF NOWAIT RECEIVES OUTSTANDING. (SOCKERR 58)	User tried to issue too many consecutive nowait receives without calling IOWAIT.	Call IOWAIT to complete a receive. The limit is 1.
59 SOCKET TIMEOUT. (SOCKERR 59)	The socket timer popped before data was received.	If this is not desired, call IPCCONTROL to increase or disable the timeout.
60 UNABLE TO ALLOCATE AN AFT. (SOCKERR 60)	User has no space for allocating an active file table entry (socket descriptor).	Close unnecessary files or sockets. Run the :PREP program with a greater MAXDATA segment size. Run the program with the NOCB option.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
62 CONNECTION REQUEST PENDING; CALL IPCRECV TO COMPLETE. (SOCKERR 62)	User called IPCCONNECT without a subsequent IPCRECV before issuing the current request.	Call IPCRECV.
63 WAITING CONFIRMATION; CALL IPCCONTROL TO ACCEPT/REJECT. (SOCKERR 63)	IPCRECV called with deferred connection option. IPCCONTROL has not been called to accept/reject.	The call IPCCONTROL with accept/reject option.
64 REMOTE ABORTED THE CONNECTION. (SOCKERR 64)	Remote protocol module aborted the connection. This will occur when a peer has called IPCSHUTDOWN on the connection.	Call IPCSHUTDOWN to clean up your end of the connection.
65 CONNECTION ABORTED BY LOCAL PROTOCOL MODULE. (SOCKERR 65)	Local protocol module encountered some error which caused it to abort the connection.	Call IPCSHUTDOWN to clean up your end of the connection. See "Submitting an SR."
66 INVALID CONNECTION DESCRIPTOR. (SOCKERR 66)	Supplied value is not that of a valid VC socket (connection) descriptor.	Check the value being given.
67 CONNECTION FAILURE DETECTED. (SOCKERR 67)	An event occurred which caused the local protocol module to determine that the connection is no longer up (e.g., retransmitted data was never acknowledged).	Call IPCSHUTDOWN to clean up your end of the connection.
68 RECEIVED A GRACEFUL RELEASE OF THE CONNECTION. (SOCKERR 68)	Informational message.	Do not attempt to receive any more data.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
69 MUTUALLY EXCLUSIVE *flags* OPTIONS SPECIFIED. (SOCKERR 69)	Bits in the *flags* parameter were set which indicate requests for mutually exclusive options.	Check and clear the appropriate bits.
70 CAN'T GIVE SHARED CONNECTIONS. (SOCKERR 70)	Transferring connections and shared connections are mutually exclusive actions.	Only attempt to transfer or share connections.
71 I/O OUTSTANDING. (SOCKERR 71)	Attempted an operation with nowait I/O outstanding.	Call IOWAIT to complete the I/O or IPCCONTROL to abort any receives.
74 INVALID IPCCONTROL REQUEST CODE. (SOCKERR 74)	Request code is unknown or a nonprivileged user requested a privileged option.	Validate the value being passed.
75 UNABLE TO CREATE A PORT FOR LOW LEVEL I/O. (SOCKERR 75)	Unable to create an entity used for communication between NetIPC and the protocol module. This error might occur if you are trying to open a large number of connections and do not have enough PCBs configured.	Contact your operator to see if the number of PCBs could be increased. (Number of connections divided by two is a good estimate). See "Submitting an SR".
76 INVALID TIMEOUT VALUE. (SOCKERR 76)	Value specified for the timeout is negative.	Modify the value.
77 INVALID WAIT/NOWAIT MODE. (SOCKERR 77)	Mode of socket cannot be used.	Use IPCCONTROL to specify correct mode.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
78 TRACING NOT ENABLED. (SOCKERR 78)	Attempted to turn off trace when tracing was not on.	Remove the call.
79 INVALID TRACE FILE NAME. (SOCKERR 79)	Requested trace file name is not valid.	Validate and modify the trace file name.
80 ERROR IN TRACE DATA LENGTH OPTION. (SOCKERR 80)	An error was detected in the option specifying the maximum amount of data to be traced (e.g., negative value, too large, too many bytes used to specify the value).	Modify the values being used.
81 ERROR IN NUMBER OF TRACE FILE RECORDS OPTION. (SOCKERR 81)	An error was detected in the option specifying the maximum amount of records to be in the trace file (e.g., negative or too large a value, too many bytes used to specify the value).	Modify the values being used.
82 TRACING ALREADY ENABLED. (SOCKERR 82)	Attempted to turn on tracing when tracing already enabled.	Remove the call or turn off trace before the call.
83 ATTEMPT TO TURN ON TRACE FAILED. (SOCKERR 83)	Network Management was unable to enable tracing.	Call IPCCHECK; the protocol module error returned will be the Network Management error number. Consult your Network Management manual for the appropriate action to take.
84 PROCESS HAS NO LOCAL SOCKET DATA STRUCTURES. (SOCKERR 84)	IPCCHECK was called, but the user had no sockets or destination descriptors, and therefore no data structure for retaining error codes.	None, but no NetIPC or protocol module errors are available.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
85 INVALID SOCKET ERROR NUMBER. (SOCKERR 85)	IPCERRMSG was called with an invalid NetIPC error code.	Check the value being passed.
86 UNABLE TO OPEN ERROR CATALOG SOCKCAT.NET.SYS. (SOCKERR 86)	The error message catalog does not exist, it is opened exclusively, or the caller does not have access rights to the file.	Notify your operator.
87 GENMESSAGE FAILURE; NOT A MESSAGE CATALOG. (SOCKERR 87)	MAKECAT was not successfully run on the message catalog SOCKCAT.NET.SYS.	Notify your operator.
88 INVALID REQUEST SOCKET DESCRIPTOR. (SOCKERR 88)	Internal error.	See "Submitting an SR".
89 INVALID REPLY SOCKET DESCRIPTOR (SOCKERR 88)	Internal error.	See "Submitting an SR".
91 WOULD EXCEED LIMIT OF REPLIES EXPECTED. (SOCKERR 91)	Internal error.	See "Submitting an SR".
92 MUST REPLY TO BEFORE RECEIVING ANOTHER REQUEST. (SOCKERR 92)	Internal error.	See "Submitting an SR".

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
93 INVALID SEQUENCE NUMBER. (SOCKERR 93)	Internal error.	See "Submitting an SR".
94 NO OUTSTANDING REQESTS. (SOCKERR 94)	Internal error.	See "Submitting an SR".
95 RECEIVED AN UNSOLICITED REPLY. (SOCKERR 95)	Internal error.	See "Submitting an SR".
97 WOULD EXCEED LIMIT OF SHARED CONNECTIONS. (SOCKERR 97)	Internal error.	See "Submitting an SR".
96 INTERNAL BUFFER MANAGER ERROR. (SOCKERR 96)	Attempted use of the buffer manager by NetIPC or the protocol module resulted in an error.	See "Submitting an SR".
98 INVALID DATA SEGMENT INDEX IN VECTORED DATA. (SOCKERR 98)	Data segment index value in the vectored data array is not valid.	Check the value being supplied.
99 INVALID BYTE COUNT IN VECTORED DATA. (SOCKERR 99)	The count of data in the vectored data array is invalid.	Check the values being given.
100 TOO MANY VECTORED DATA DESCRIPTORS. (SOCKERR 100)	More than two data locations were specified in the vectored data array.	Limit the number to two per operation. Use multiple sends or receives if necessary.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
101 INVALID VECTORED DATA TYPE. (SOCKERR 101)	Type of vectored data is unknown (must be a 0, 1, or 2) or the data type is for a data segment (1 or 2) and the user is not privileged.	Check the value being used.
102 UNABLE TO GRACEFULLY RELEASE THE CONNECTION. (SOCKERR 102)	1. Protocol module does not support graceful release. 2. Process tried to release connection that was not in correct state. 3. Output pending.	Check command sequence.
103 USER DATA NOT SUPPORTED DURING CONNECTION ESTABLISHMENT. (SOCKERR 103)	User data option is not supported for IPCRECV or IPCCONNECT.	Do not use user data option.
104 CAN'T NAME A REQUEST SOCKET. (SOCKERR 104)	Internal error.	See "Submitting an SR".
105 NO REPLY RECEIVED. (SOCKERR 105)	Internal error.	See "Submitting an SR".
106 ADDRESS CURRENTLY IN USE BY ANOTHER SOCKET. (SOCKERR 106)	Address being specified for use is already being used.	If you are a privileged user trying to specify a well known address, or try again later. If you are nonprivileged, then see "Submitting an SR".
107 TRANSPORT IS GOING DOWN. (SOCKERR 107)	The transport is being shut down.	Call IPCSHUTDOWN on all sockets and destination descriptors.
108 USER HAS RELEASED CONNECTION; UNABLE TO SEND DATA. (SOCKERR 108)	Process tried to send after initiating a graceful release.	Check command sequence.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
109 PEER HAD RELEASED THE CONNECTION; UNABLE TO RECEIVE DATA. (SOCKERR 109)	Process tried to receive after remote initiated graceful release.	Check command sequence.
110 UNANTICIPATED ERROR. (SOCKERR 110)	NetIPC received a protocol module error which it was unable to map.	Call IPCCHECK to get the protocol module error. Call IPCSHUTDOWN to clean up. See "Submitting an SR".
111 INTERNAL SOFTWARE ERROR DETECTED. (SOCKERR 111)	Internal error.	See "Submitting an SR".
112 NOT PERMITTED WITH SOFTWARE INTERRUPTS ENABLED. (SOCKERR 112)	Internal error.	See "Submitting an SR".
113 INVALID SOFTWARE INTERRUPT PROCEDURE LABEL. (SOCKERR 113)	Internal error.	See "Submitting an SR".
114 CREATION OF SOCKET REGISTRY PROCESS FAILED. (SOCKERR 114)	Possible causes include: 1. Resource limitations or 2. Socket registry program missing.	1. Retry later. 2. Contact your HP representative for assistance.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Cause	Action
116 DESTINATION UNREACHABLE. (SOCKERR 116)	The transport was unable to route the packet to the destination. This may be caused by: 1. Network Interface (NI) was not started or 2. IP address is incorrect either in the network directory, or the routing information in the configuration file.	 1. Start the NI. 2. Use NMMGR to correct the network directory or configuration file. (After correcting the configuration file you must issue a :NETCONTROL UPDATE.)
117 ATTEMPT TO ESTABLISH CONNECTION FAILED. (SOCKERR 117)	Protocol module was unable to set up the requested connection. This may be caused by the remote protocol module not being active.	Notify your operator.
118 INCOMPATIBLE VERSIONS. (SOCKERR 118)	NetIPC software was incompatible with the software being executed by the remote registry process.	Notify your operator.
119 ERROR IN BURST SIZE OPTION. (SOCKERR 119)	An unsupported option was specified in the *opt* record, or a nonprivileged user attempted to specify a privileged option.	Check your *opt* record and remove or modify the option.
120 ERROR IN WINDOW UPDATE THRESHOLD OPTION. (SOCKERR 120)	An unsupported option was specified in the *opt* record, or a nonprivileged user attempted to specify a privileged option.	Check your *opt* record and remove or modify the option.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Meaning/Cause	Action
124 ENTRY NUMBER NOT VALID FOR SPECIFIED OPT RECORD. (SOCKERR 124)	User error. Entry number of option is either negative or higher than specified in the INITOPT opt value.	Correct and reissue command.
125 INVALID OPTION DATA LENGTH. (SOCKERR 125)	User error. Data length for option either negative or too high.	Correct and reissue command.
126 INVALID NUMBER OF EVENTUAL OPT RECORD ENTRIES. (SOCKERR 126)	Number of option entries is either too high or negative. Either an internal restriction or a user mistake.	Correct the entry by making the number positive or smaller in value.
127 UNABLE TO READ ENTRY FROM OPT RECORD. (SOCKERR 127)	The option record indicates that the entry is not valid or the buffer supplied by the user was too small to hold all of the data.	Check entry number, make sure the option record has not been written over and check output buffer length.
131 PROTOCOL MODULE DOES NOT HAVE SUFFICIENT RESOURCES. (SOCKERR 131)	Protocol module is temporarily out of buffers or internal data descriptors.	Retry later when the system load is lighter.
141 X.25 NETWORK NAME INCORRECTLY SPECIFIED. (SOCKERR 141)	Using direct access to X.25, network name not specified or incorrect.	The network name (option code 140) must be specified in the IPCCREATE call for X.25 access. Network name must be 1 to 8 characters in length.
142 INVALID CALL USER DATA OPT RECORD ENTRY. (SOCKERR 142)	The length of the call user data is invalid for the transport protocol type.	Check length of call user data opt in the opt array. It must be greater than 1 for IPCCONNECT and 4 for IPCRECVCN. The maximum length is protocol specific.
143 INVALID FACILITIES SET OPT RECORD ENTRY. (SOCKERR 143)	The facility set passed as a parameter has not been found in the internal facility table for a switched virtual circuit (SVC) or permanent virtual circuit (PVC).	Use SVC or PVC facility sets defined in configuration.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Meaning/Cause	Action
144 INVALID CALLING NODE OPT ENTRY. (SOCKERR 144)	The user may request the address of the calling node. Address of 8 bytes will be returned.	The length of the option entry must be exactly 8 bytes.
145 INVALID REASON CODE. (SOCKERR 145)	A reason code (option 143) was specified for an IPCSHUTDOWN on a connection that is not using direct access to X.25.	Omit the invalid reason code from the IPCSHUTDOWN call.
146 RESET EVENT OCCURRED ON X.25 CONNECTION. (SOCKERR 146)	1. A reset packet was sent internally because of an internal error or because of resource shortage (mainly buffers). 2. An unsolicitated reset packet was received.	1. Re-issue the call if necessary. Check buffer usage and adjust buffer configuration to usage. 2. Use IPCCONTROL request 12 to retrieve cause/diagnosic field.
151 COULD NOT OBTAIN A SEMAPHORE. (SOCKERR 151)	The attempt to obtain a semaphore before sending a message to the protocol module failed.	See "Submitting an SR".
153 SOCKET IS ALREADY IN USE. (SOCKERR 153)	A single socket per network interface can be created with the catch-all capability.	Wait for catch-all socket to be released.
155 INVALID X.25 FLAG OPT RECORD ENTRY. (SOCKERR 155)	For direct access to X.25 (level 3) the *opt* record flags (code 144) parameter in IPCCREATE, IPCCONNECT, IPCRECVCN or IPCSEND is improperly set, or the length is incorrect.	Check the call containing the *opt* record flags parameter and correct the entry.
156 INTERRUPT EVENT OCCURRED ON X.25 CONNECTION. (SOCKERR 156)	An interrupt packet was received.	Use IPCCONTROL request 12 to retrieve interrupt data.
157 ALL OUTGOING SWITCHED VIRTUAL CIRCUITS ARE BUSY. (SOCKERR 157)	No more free LCN one-way outgoing or two-ways SVC.	Wait for LCN to be free and re-issue call.

NetIPC ERRORS (SOCKERRS) (cont'd)

Message	Meaning/Cause	Action
158 CONNECTION REQUEST REJECTED BY REMOTE. (SOCKERR 158)	The remote node received the connection request and rejected it. (An outgoing call packet was answered by a clear packet.)	The call may be retried later. Use IPCCONTROL request 12 to retrieve cause/diagnostic field.
159 INVALID X.25 D BIT SETTING. (SOCKERR 159)	User requested an X.25 packet to be sent with the D-bit set while the facility set in use does not allow it.	Use a facility set allowing D-bit usage.
160 INCOMPATIBLE WITH PROTOCOL STATE. (SOCKERR 160)	The user requested an operation which is not supported by the protocol module.	Verify the sequence of intrinsic calls.
162 X.25 PERMANENT VIRTUAL CIRCUIT DOES NOT EXIST. (SOCKERR 162)	The permanent virtual circuit (PVC) was not found.	Check if the PVC is configured.
163 PERMANENT VIRTUAL CIRCUIT ALREADY ESTABLISHED. (SOCKERR 163)	A connection request was issued on a PVC which is in use by another process.	Select a different PVC or retry later.
164 ADDRESS VALUE IS OUT OF RANGE. (SOCKERR 164)	Address specified in opt parameter is out of range.	Specify an address in the range 30767 to 32767.
165 INVALID ADDRESS LENGTH. (SOCKERR 165)	An invalid address length was specified in the *opt* parameter.	The address length is 2 bytes. (For non-privileged users)
166 CONNECTION NOT IN VIRTUAL CIRCUIT WAIT CONFIRM STATE. (SOCKERR 166)	Attempt was made to accept or reject a connection that is open or in the process of closing.	Use flags parameter in IPCRECVCN to defer acceptence or rejection of the connection request.
167 TIMEOUT NOT ALLOWED ON SHARED CONNECTION. (SOCKERR 167)	Attempt to set a send time out on a shared connection.	Use IPCCONTROL to disallow sharing of the connection or do not attempt to set send time out on this connection.
168 RESTART EVENT OCCURRED ON X.25 CONNECTION. (SOCKERR 168)	Connection has been aborted because a restart packet was received or was sent.	Issue an IPCSHUTDOWN on the virtual circuit. Wait for the Restart procedure to complete.

INDEX

A

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

V

W

X

SPECIAL CHARACTERS

Product Line Sales/Support Key

Key Product Line
A Analytical
CM Components
C Computer Systems
E Electronic Instruments & Measurement Systems
M Medical Products
P Personal Computation Products
*** Sales only for specific product line**
**** Support only for specific product line**

IMPORTANT: These symbols designate general product line capability. They do not insure sales or support availability for all products within a line, at all locations. Contact your local sales office for information regarding locations where HP support is available for specific products.

HEADQUARTERS OFFICES

If there is no sales office listed for your area, contact one of these headquarters offices.

ASIA
Hewlett-Packard Asia Ltd.
47/F, 26 Harbour Rd.,
Wanchai, **HONG KONG**
G.P.O. Box 863, Hong Kong
Tel: 5-8330833
Telex: 76793 HPA HX
Cable: HPASIAL TD

CANADA
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 069-8644

EASTERN EUROPE
Hewlett-Packard Ges.m.b.h.
Lieblgasse 1
P.O.Box 72
A-1222 **VIENNA,** Austria
Tel: (222) 2500-0
Telex: 1 3 4425 HEPA A

NORTHERN EUROPE
Hewlett-Packard S.A.
V. D. Hooplaan 241
P.O.Box 999
NL-118 LN 15 **AMSTELVEEN**
The Netherlands
Tel: 20 5479999
Telex: 18919 hpner

SOUTH EAST EUROPE
Hewlett-Packard S.A.
World Trade Center
110 Avenue Louis-Casai
1215 Cointrin, **GENEVA,** Switzerland
Tel: (022) 98 96 51
Telex: 27225 hpser
Mail Address:
P.O. Box
CH-1217 Meyrin 1
GENEVA
Switzerland

MIDDLE EAST AND CENTRAL AFRICA
Hewlett-Packard S.A.
Middle East/Central
Africa Sales H.Q.
7, rue du Bois-du-Lan
P.O. Box 364
CH-1217 Meyrin 1
GENEVA
Switzerland
Tel: (022) 83 12 12
Telex: 27835 hmea ch
Telefax: (022) 83 15 35

UNITED KINGDOM
Hewlett-Packard Ltd.
Nine Mile Ride
WOKINGHAM
Berkshire, RG113LL
Tel: 0344 773100
Telex: 848805/848814/848912

UNITED STATES OF AMERICA
Customer Information Center
(800) 752-0900
6:00 AM to 5 PM Pacific Time

EASTERN USA
Hewlett-Packard Co.
4 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370

MIDWESTERN USA
Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800

SOUTHERN USA
Hewlett-Packard Co.
2000 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500

WESTERN USA
Hewlett-Packard Co.
5161 Lankershim Blvd.
NORTH HOLLYWOOD, CA 91601
Tel: (818) 505-5600

OTHER INTERNATIONAL AREAS
Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK

ALGERIA
Hewlett-Packard Trading S.A.
Bureau de Liaison Alger
Villa des Lions
9, Hai Galloul
DZ-BORDJ EL BAHRI
Tel: 76 03 36
Telex: 63343 dlion dz

ANGOLA
Telectra Angola LDA
Empresa Técnica de Equipamentos
16 rue Cons. Julio de Vilhema
LUANDA
Tel: 35515,35516
Telex: 3134
E,P

ARGENTINA
Hewlett-Packard Argentina S.A.
Montaneses 2140/50
1428 **BUENOS AIRES**
Tel: 541-11-1441
Telex: 22796 HEW PAC-AR
A,C,E,P

Biotron S.A.C.I.M.e.I.
Av. Paso Colon 221, Piso 9
1399 **BUENOS AIRES**
Tel: 541-333-490,
541-322-587
Telex: 17595 BIONAR
M

Laboratorio Rodriguez
Corswant S.R.L.
Misiones, 1156 - 1876
Bernal, Oeste
BUENOS AIRES
Tel: 252-3958, 252-4991
A

Intermaco S.R.L.
Florida 537/71
Galeria Jardin - Local 28
1005 **BUENOS AIRES**
Tel: 393-4471/1928
Telex: 22796 HEW PAC-AR
P (Calculators)

Argentina Esanco S.R.L.
A/ASCO 2328
1416 **BUENOS AIRES**
Tel: 541-58-1981, 541-59-2767
Telex: 22796 HEW PAC-AR
A

All Computers S.A.
Montaneses 2140/50 5 Piso
1428 **BUENOS AIRES**
Tel: 781-4030/4039/783-4886
Telex: 18148 Ocme
P

AUSTRALIA

Adelaide, South Australia Office
Hewlett-Packard Australia Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 61-8-272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,C,CM,E,P

Brisbane, Queensland Office
Hewlett-Packard Australia Ltd.
10 Payne Road
THE GAP, Queensland 4061
Tel: 61-7-300-4133
Telex: 42133
Cable: HEWPARD Brisbane
A,C,CM,E,M,P

Canberra, Australia Capital Territory Office
Hewlett-Packard Australia Ltd.
Thynne Street, Fern Hill Park
BRUCE, A.C.T. 2617
P.O. Box 257,
JAMISON, A.C.T. 2614
Tel: 61-62-80-4244
Telex: 62650
Cable: HEWPARD Canberra
C,CM,E,P

Melbourne, Victoria Office
Hewlett-Packard Australia Ltd.
31-41 Joseph Street
P.O. Box 221
BLACKBURN, Victoria 3130
Tel: 61-3-895-2895
Telex: 31-024
Cable: HEWPARD Melbourne
A,C,CM,E,M,P

Perth, Western Australia Office
Hewlett-Packard Australia Ltd.
Herdsman Business Park
CLAREMONT, W.A. 6010
Tel: 61-9-383-2188
Telex: 93859
Cable: HEWPARD Perth
C,CM,E,P

Sydney, New South Wales Office
Hewlett-Packard Australia Ltd.
17-23 Talavera Road
P.O. Box 308
NORTH RYDE, N.S.W. 2113
Tel: 61-2-888-4444
Telex: 21561
Cable: HEWPARD Sydney
A,C,CM,E,M,P

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Verkaufsbuero Graz
Grottenhofstrasse 94
A-8052 **GRAZ**
Tel: 43-316-291-5660
Telex: 312375
C,E

Hewlett-Packard Ges.m.b.h.
Lieblgasse 1
P.O. Box 72
A-1222 **VIENNA**
Tel: 43-222-2500
Telex: 134425 HEPA A
A,C,CM,E,M,P

BAHRAIN
Green Salon
P.O. Box 557
MANAMA
Tel: 255503-250950
Telex: 84419
P

Wael Pharmacy
P.O. Box 648
MANAMA
Tel: 256123
Telex: 8550 WAEL BN
E,M

Zayani Computer Systems
218 Shaik Mubarak Building
Government Avenue
P.O. Box 5918
MANAMA
Tel: 276278
Telex: 9015 plans bn
P

BELGIUM
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 **BRUSSELS**
Tel: (02) 32-2-761-31-11
Telex: 23494 hewpac
A,C,CM,E,M,P

BERMUDA
Applied Computer Technologies
Atlantic House Building
P.O. Box HM 2091
Par-La-Ville Road
HAMILTON 5
Tel: 295-1616
Telex: 380 3589/ACT BA
P

BOLIVIA
Arrellano Ltda
Av. 20 de Octubre #2125
Casilla 1383
LA PAZ
Tel: 368541
M

BRAZIL
Hewlett-Packard do Brasil S.A.
Alameda Rio Negro, 750-I. AND.
ALPHAVILLE
06400 Barueri SP
Tel: (011) 421.1311
Telex: (011) 71351 HPBR BR
Cable: HEWPACK Sao Paulo
CM,E

Hewlett-Packard do Brasil S.A.
Praia de Botafago 228-A-614
6. AND.-CONJ. 601
Edificio Argentina - Ala A
22250 **RIO DE JANEIRO, RJ**
Tel: (021) 552-6422
Telex: 21905 HPBR BR
Cable: HEWPACK Rio de Janeiro
E

Van Den Cientifica Ltda.
Rua Jose Bonifacio, 458
Todos os Santos
20771 **RIO DE JANEIRO, RJ**
Tel: (021) 593-8223
Telex: 33487 EGLB BR
A

ANAMED I.C.E.I. Ltda.
Rua Vergueiro, 360
04012 **SAO PAULO, SP**
Tel: (011) 572-1106
Telex: 24720 HPBR BR
M

Datatronix Electronica Ltda.
Av. Pacaembu 746-C11
SAO PAULO, SP
Tel: (118) 260111
CM

BRUNEI
Komputer Wisman Sdn Bhd
G6, Chandrawaseh Cmplx,
Jalan Tutong
P.O. Box 1297,
BANDAR SERI BEGAWAN
NEGARA BRUNI DARUSSALAM
Tel: 673-2-2000-70/26711
C,E,P

CAMEROON
Beriac
B. P. 23
DOUALA
Tel: 420153
Telex: 5351
C,P

CANADA
Alberta
Hewlett-Packard (Canada) Ltd.
3030 3rd Avenue N.E.
CALGARY, Alberta T2A 6T7
Tel: (403) 235-3100
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
11120-178th Street
EDMONTON, Alberta T5S 1P2
Tel: (403) 486-6666
A,C,CM,E,M,P

British Columbia
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND,
British Columbia V6X 2W8
Tel: (604) 270-2277
Telex: 610-922-5059
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
121 - 3350 Douglas Street
VICTORIA, British Columbia V8Z 3L1
Tel: (604) 381-6616
C

Manitoba
Hewlett-Packard (Canada) Ltd.
1825 Inkster Blvd.
WINNIPEG, Manitoba R2X 1R3
Tel: (204) 694-2777
A,C,CM,E,M,P*

New Brunswick
Hewlett-Packard (Canada) Ltd.
814 Main Street
MONCTON, New Brunswick E1C 1E6
Tel: (506) 855-2841
C

Nova Scotia
Hewlett-Packard (Canada) Ltd.
Suite 111
900 Windmill Road
DARTMOUTH, Nova Scotia B3B 1P7
Tel: (902) 469-7820
C,CM,E*,M,P*

Ontario
Hewlett-Packard (Canada) Ltd.
3325 N. Service Rd., Unit W03
BURLINGTON, Ontario L7N 3G2
Tel: (416) 335-8644
C,M*

Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 069-83644
A,C,CM,E,M,P

Hewlett-Packard (Canada) Ltd.
2670 Queensview Dr.
OTTAWA, Ontario K2B 8K1
Tel: (613) 820-6483
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
3790 Victoria Park Ave.
WILLOWDALE, Ontario M2H 3H7
Tel: (416) 499-2550
C,E

Quebec
Hewlett-Packard (Canada) Ltd.
17500 Trans Canada Highway
South Service Road
KIRKLAND, Quebec H9J 2X8
Tel: (514) 697-4232
Telex: 058-21521
A,C,CM,E,M,P*

Hewlett-Packard (Canada) Ltd.
1150 rue Claire Fontaine
QUEBEC CITY, Quebec G1R 5G4
Tel: (418) 648-0726
C

Hewlett-Packard (Canada) Ltd.
130 Robin Crescent
SASKATOON, Saskatchewan S7L 6M7
Tel: (306) 242-3702
C

CHILE
ASC Ltda.
Austria 2041
SANTIAGO
Tel: 223-5946, 223-6148
Telex: 392-340192 ASC CK
C,P

Jorge Calcagni y Cia
Av. Italia 634 Santiago
Casilla 16475
SANTIAGO 9
Tel: 9-011-562-222-0222
Telex: 392440283 JCYCL CZ
CM,E,M

Metrolab S.A.
Monjitas 454 of. 206
SANTIAGO
Tel: 395752, 398296
Telex: 340866 METLAB CK
A

Olympia (Chile) Ltda.
Av. Rodrigo de Araya 1045
Casilla 256-V
SANTIAGO 21
Tel: 225-5044
Telex: 340892 OLYMP
Cable: Olympiachile Santiagochile
C,P

CHINA, People's Republic of
China Hewlett-Packard Co., Ltd.
47/F China Resources Bldg.
26 Harbour Road
HONG KONG
Tel: 5-8330833
Telex: 76793 HPA HX
Cable: HP ASIA LTD
A*,M*

China Hewlett-Packard Co., Ltd.
P.O. Box 9610, Beijing
4th Floor, 2nd Watch Factory Main
Shuang Yu Shou, Bei San Huan Road
Hai Dian District
BEIJING
Tel: 33-1947 33-7426
Telex: 22601 CTSHP CN
Cable: 1920 Beijing
A,C,CM,E,M,P

China Hewlett-Packard Co., Ltd.
CHP Shanghai Branch
23/F Shanghai Union Building
100 Yan An Rd. East
SHANG-HAI
Tel: 265550
Telex: 33571 CHPSB CN
Cable: 3416 Shanghai
A,C,CM,E,M,P

COLOMBIA
Instrumentación
H. A. Langebaek & Kier S.A.
Carrerra 4A No. 52A-26
Apartado Aereo 6287
BOGOTA 1, D.E.
Tel: 212-1466
Telex: 44400 INST CO
Cable: AARIS Bogota
CM,E,M

Nefromedicas Ltda.
Calle 123 No. 9B-31
Apartado Aereo 100-958
BOGOTA D.E., 10
Tel: 213-5267, 213-1615
Telex: 43415 HEGAS CO
A

Compumundo
Avenida 15 # 107-80
BOGOTA D.E.
Tel: 57-214-4458
Telex: 39645466 MARCO
P

Carvajal, S.A.
Calle 29 Norte No. 6A-40
Apartado Aereo 46
CALI
Tel: 9-011-57-3-621888
Telex: 39655650 CUJCL CO
C,E,P

CONGO
Seric-Congo
B. P. 2105
BRAZZAVILLE
Tel: 815034
Telex: 5262

COSTA RICA
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSÉ
Tel: 9-011-506-243-820
Telex: 3032367 GALGUR CR
CM,E,M

O. Fischel R. Y. Cia. S.A.
Apartados 434-10174
SAN JOSE
Tel: 23-72-44
Telex: 2379
Cable: OFIR
A

CYPRUS
Telerexa Ltd.
P.O. Box 1152
Valentine House
8 Stassandrou St.
NICOSIA
Tel: 45 628, 62 698
Telex: 5845 tlrx cy
E,M,P

DENMARK
Hewlett-Packard A/S
Kongevejen 25
DK-3460 **BIRKEROD**
Tel: 45-02-81-6640
Telex: 37409 hpas dk
A,C,CM,E,M,P

Hewlett-Packard A/S
Rolighedsvej 32
DK-8240 **RISSKOV,** Aarhus
Tel: 45-06-17-6000
Telex: 37409 hpas dk
C,E

DOMINICAN REPUBLIC
Microprog S.A.
Juan Tomás Mejía y Cotes No. 60
Arroyo Hondo
SANTO DOMINGO
Tel: 565-6268
Telex: 4510 ARENTA DR (RCA)
P

ECUADOR
CYEDE Cia. Ltda.
Avenida Eloy Alfaro 1749
y Belgica
Casilla 6423 CCI
QUITO
Tel: 9-011-593-2-450975
Telex: 39322548 CYEDE ED
E,P

Medtronics
Valladolid 524 Madrid
P.O. 9171, **QUITO**
Tel: 2-238-951
Telex: 2298 ECUAME ED
A

Hospitalar S.A.
Robles 625
Casilla 3590
QUITO
Tel: 545-250, 545-122
Telex: 2485 HOSPTL ED
Cable: HOSPITALAR-Quito
M

Ecuador Overseas Agencies C.A.
Calle 9 de Octubre #818
P.O. Box 1296, Guayaquil
QUITO
Tel: 306022
Telex: 3361 PBCGYE ED
M

EGYPT
Sakrco Enterprises
P.O. Box 259
ALEXANDRIA
Tel: 802908, 808020, 805302
Telex: 54333
C

International Engineering Associates
6 El Gamea Street
Agouza
CAIRO
Tel: 71-21-68134-80-940
Telex: 93830 IEA UN
Cable: INTEGASSO
E

Sakrco Enterprises
70 Mossadak Street
Dokki, Giza
CAIRO
Tel: 706 440, 701 087
Telex: 9337
C

S.S.C. Medical
40 Gezerat El Arab Street
Mohandessin
CAIRO
Tel: 803844, 805998, 810263
Telex: 20503 SSC UN
M*

EL SALVADOR
IPESA de El Salvador S.A.
29 Avenida Norte 1223
SAN SALVADOR
Tel: 9-011-503-266-858
Telex: 301 20539 IPESA SAL
A,C,CM,E,P

ETHIOPIA
Seric-Ethiopia
P.O. Box 2764
ADDIS ABABA
Tel: 185114
Telex: 21150
C,P

FINLAND
Hewlett-Packard Finland
Field Oy
Niittylanpolku IO
00620 **HELSINKI**
Tel: (90) 757-1011
Telex: 122022 Field SF
CM

Hewlett-Packard Oy
Piispankalliontie 17
02200 **ESPOO**
Tel: (90) 887-21
Telex: 121563 HEWPA SF
A, C, E, M, P

FRANCE
Hewlett-Packard France
Z.I. Mercure B
Rue Berthelot
13763 Les Milles Cedex
AIX-EN-PROVENCE
Tel: 33-42-59-4102
Telex: 410770F
A,C,E,M

Hewlett-Packard France
64, Rue Marchand Saillant
F-61000 **ALENCON**
Tel: (33) 29 04 42
C**

Hewlett-Packard France
Batiment Levitan
2585, route de Grasse
Bretelle Autoroute
06600 **ANTIBES**
Tel: (93) 74-59-19
C

SALES & SUPPORT OFFICES
Arranged alphabetically by country

FRANCE (Cont'd)

Hewlett-Packard France
28 Rue de la République
Boite Postale 503
25026 **BESANÇON CEDEX, FRANCE**
Tel: (81) 83-16-22
Telex: 361157
C,E*

Hewlett-Packard France
ZA Kergaradec
Rue Fernand Forest
F-29239 **GOUEESNOU**
Tel: (98) 41-87-90
E

Hewlett-Packard France
Chemin des Mouilles
Boite Postale 162
69131 **ECULLY** Cedex (Lyon)
Tel: 33-78-33-8125
Telex: 310617F
A,C,E,M,P*

Hewlett-Packard France
Parc d'activités du Bois Briard
2 Avenue du Lac
F-91040 **EVRY** Cedex
Tel: 3311/6077 9660
Telex: 692315F
C

Hewlett-Packard France
Application Center
5, avenue Raymond Chanas
38320 **EYBENS** (Grenoble)
Tel: (76) 62-57-98
Telex: 980124 HP GRENOB EYBE
C

Hewlett-Packard France
Rue Fernand. Forest
Z.A. Kergaradec
29239 **GOUESNOU**
Tel: (98) 41-87-90

Hewlett-Packard France
Parc Club des Tanneries
Batiment B4
4, Rue de la Faisanderie
67381 **LINCOLSHEIM**
(Strasbourg)
Tel: (88) 76-15-00
Telex: 890141F
C,E*,M*,P*

Hewlett-Packard France
Centre d'affaires Paris-Nord
Bâtiment Ampère
Rue de la Commune de Paris
Boite Postale 300
93153 **LE BLANC-MESNIL**
Tel: (1) 865-44-52
Telex: 211032F
C,E,M

Hewlett-Packard France
Parc d'activités Cadéra
Quartier Jean-Mermoz
Avenue du Président JF Kennedy
33700 **MÉRIGNAC** (Bordeaux)
Tel: 33-56-34-0084
Telex: 550105F
C,E,M

Hewlett-Packard France
3, Rue Graham Bell
BP 5149
57074 **METZ** Cedex
Tel: (87) 36-13-31
Telex: 860602F
C,E

Hewlett-Packard France
Miniparc-ZIRST
Chemin du Vieux Chêne
38240 **MEYLAN** (Grenoble)
Tel: (76) 90-38-40
980124 HP Grenobe
C

Hewlett-Packard France
Bureau vert du Bois Briand
Cheman de la Garde
- CP 212 212
44085 **NANTES** Cedex
Tel: (40) 50-32-22
Telex: 711085F
A,C,E,CM*,P

Hewlett-Packard France
125, Rue du Faubourg Bannier
45000 **ORLÉANS**
Tel: 33-38-62-2031
E,P*

Hewlett-Packard France
Zone Industrielle de Courtaboeuf
Avenue des Tropiques
91947 **LES ULIS** Cedex (Orsay)
Tel: 33-6-907 7825
Telex: 600048F
A,C,CM,E,M,P**

Hewlett-Packard France
15, Avenue de L'Amiral-Bruix
75782 **PARIS** Cedex 16
Tel: 33-15-02-1220
Telex: 613663F
C,P*

Hewlett-Packard France
242 Ter, Ave J Mermoz
64000 **PAU**
Tel: 33-59-80-3802
Telex: 550365F
C,E*

Hewlett-Packard France
6, Place Sainte Croix
86000 **POITIERS**
Tel: 33-49-41-2707
Telex: 792335F
C, E*

Hewlett-Packard France
47, Rue de Chativesle
51100 **REIMS**
Tel: 33-26-88-6919
C, P*

Hewlett-Packard France
Parc d'activités de la Poterie
Rue Louis Kerautel-Botmel
35000 **RENNES**
Tel: 33-99-51-4244
Telex: 740912F
A*,C,E,M,P*

Hewlett-Packard France
98 Avenue de Bretagne
76100 **ROUEN**
Tel: 33-35-63-5766
Telex: 770035F
C,E

Hewlett-Packard France
4, Rue Thomas-Mann
Boite Postale 56
67033 **STRASBOURG** Cedex
Tel: (88) 28-56-46
Telex: 890141F
C,E,M,P*

Hewlett-Packard France
Le Péripole III
3, Chemin du Pigeonnier de la Cépière
31081 **TOULOUSE** Cedex
Tel: 33-61-40-1112
Telex: 531639F
A,C,E,M,P*

Hewlett-Packard France
Les Cardoulines
Batiment B2
Route des Dolines
Parc d'activite de Valbonne
Sophia Antipolis
06560 **VALBONNE** (Nice)
Tel: (93) 65-39-40
C

Hewlett-Packard France
9, Rue Baudin
26000 **VALENCE**
Tel: 33-75-42-7616
C**

Hewlett-Packard France
Carolor
ZAC de Bois Briand
57640 **VIGY** (Metz)
Tel: (8) 771 20 22
C

Hewlett-Packard France
Parc d'activité des Prés
1, Rue Papin Cedex
59658 **VILLENEUVE D'ASCQ**
Tel: 33-20-91-4125
Telex: 160124F
C,E,M,P

Hewlett-Packard France
Parc d'activités Paris-Nord 11
Boite Postale 60020
95971 Roissy Charles de Gaulle
VILLEPINTE
Tel: (1) 48 63 80 80
Telex: 211032F
C,E,M,P*

GABON

Sho Gabon
P.O. Box 89
LIBREVILLE
Tel: 721 484
Telex: 5230

GERMAN FEDERAL REPUBLIC

Hewlett-Packard GmbH
Vertriebszentrum Mitte
Hewlett-Packard-Strasse
D-6380 **BAD HOMBURG**
Tel: (06172) 400-0
Telex: 410 844 hpbhg
A,C,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Keithstrasse 2-4
D-1000 **BERLIN** 30
Tel: (030) 21 99 04-0
Telex: 018 3405 hpbln d
A,C,E,M,P

Hewlett-Packard GmbH
Verbindungsstelle Bonn
Friedrich-Ebert-Allee 26
5300 **BONN**
Tel: (0228) 234001
Telex: 8869421

Hewlett-Packard GmbH
Vertriebszentrun Südwest
Schickardstrasse 2
D-7030 **BÖBLINGEN**
Postfach 1427
Tel: (07031) 645-0
Telex: 7265 743 hep
A,C,CM,E,M,P

Hewlett-Packard GmbH
Zeneralbereich Mktg
Herrenberger Strasse 130
D-7030 **BÖBLINGEN**
Tel: (07031) 14-0
Telex: 7265739 hep

Hewlett-Packard GmbH
Geschäftsstelle
Schleefstr. 28a
D-4600 **DORTMUND**-41
Tel: (0231) 45001
Telex: 822858 hepdod
A,C,E

Hewlett-Packard gmbH
Reparaturzentrum Frankfurt
Berner Strasse 117
6000 **FRANKFURT/MAIN 60**
Tel: (069) 500001-0
Telex: 413249 hpffm

Hewlett-Packard GmbH
Vertriebszentrum Nord
Kapstadtring 5
D-2000 **HAMBURG** 60
Tel: 49-40-63-804-0
Telex: 021 63 032 hphh d
A,C,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Heidering 37-39
D-3000 **HANNOVER** 61
Tel: (0511) 5706-0
Telex: 092 3259 hphan
A,C,CM,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Rosslauer Weg 2-4
D-6800 **MANNHEIM**
Tel: 49-0621-70-05-0
Telex: 0462105 hpmhm
A,C,E

Hewlett-Packard GmbH
Geschäftsstelle
Messerschmittstrasse 7
D-7910 **NEU ULM**
Tel: 49-0731-70-73-0
Telex: 0712816 HP ULM-D
A,C,E*

Hewlett-Packard GmbH
Geschäftsstelle
Emmericher Strasse 13
D-8500 **NÜRNBERG** 10
Tel: (0911) 5205-0
Telex: 0623 860 hpnbg
C,CM,E,M,P

Hewlett-Packard GmbH
Vertriebszentrum Ratingen
Berliner Strasse 111
D-4030 **RATINGEN** 4
Postfach 31 12
Tel: (02102) 494-0
Telex: 589 070 hprad
A,C,E,M,P

Hewlett-Packard GmbH
Vertriebszentrum Muchen
Eschenstrasse 5
D-8028 **TAUFKIRCHEN**
Tel: 49-89-61-2070
Telex: 0524985 hpmch
A,C,CM,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Ermlisallee
7517 **WALDBRONN** 2
Postfach 1251
Tel: (07243) 602-0
Telex: 782 838 hepk
A,C,E

GREAT BRITAIN
See United Kingdom

GREECE

Hewlett-Packard A.E.
178, Kifissias Avenue
6th Floor
Halandri-**ATHENS**
Greece
Tel: 301116473 360, 301116726 090
Telex: 221 286 HPHLGR
A,C,CM**,E,M,P

Kostas Karaynnis S.A.
8, Omirou Street
ATHENS 133
Tel: 32 30 303, 32 37 371
Telex: 215962 RKAR GR
A,C*,CM,E

Impexin
Intelect Div.
209 Mesogion
11525 **ATHENS**
Tel: 6474481/2
Telex: 216286
P

Haril Company
38, Mihalakopoulou
ATHENS 612
Tel: 7236071
Telex: 218767
M*

Hellamco
P.O. Box 87528
18507 **PIRAEUS**
Tel: 4827049
Telex: 241441
A

GUATEMALA

IPESA DE GUATEMALA
Avenida Reforma 3-48, Zona 9
GUATEMALA CITY
Tel: 316627, 317853,66471/5
9-011-502-2-316627
Telex: 3055765 IPESA GU
A,C,CM,E,M,P

HONG KONG

Hewlett-Packard Hong Kong, Ltd.
G.P.O. Box 795
5th Floor, Sun Hung Kai Centre
30 Harbour Road, Wan Chai
HONG KONG
Tel: 852-5-832-3211
Telex: 66678 HEWPA HX
Cable: HEWPACK HONG KONG
E,C,P

CET Ltd.
10th Floor, Hua Asia Bldg.
64-66 Gloucester Road
HONG KONG
Tel: (5) 200922
Telex: 85148 CET HX
CM

Schmidt & Co. (Hong Kong) Ltd.
18th Floor, Great Eagle Centre
23 Harbour Road, Wanchai
HONG KONG
Tel: 5-8330222
Telex: 74766 SCHMC HX
A,M

ICELAND

Hewlett-Packard Iceland
Hoefdabakka 9
112 **REYKJAVIK**
Tel: 354-1-67-1000
Telex: 37409
A,C,CM,E,M,P

INDIA

Computer products are sold through Blue Star Ltd.All computer repairs and maintenance service is done through Computer Maintenance Corp.

Blue Star Ltd.
B. D. Patel House
Near Sardar Patel Colony
AHMEDABAD 380 014
Tel: 403531, 403532
Telex: 0121-234
Cable: BLUE FROST
A,C,CM,E

Blue Star Ltd.
40/4 Lavelle Road
BANGALORE 560 001
Tel: 57881, 867780
Telex: 0845-430 BSLBIN
Cable: BLUESTAR
A,C*,CM,E

Blue Star Ltd.
Band Box House
Prabhadevi
BOMBAY 400 025
Tel: 4933101, 4933222
Telex: 011-71051
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
BOMBAY 400 025
Tel: 422-6155
Telex: 011-71193 BSSS IN
Cable: FROSTBLUE
A,CM,E,M

Blue Star Ltd.
Kalyan, 19 Vishwas Colony
Alkapuri, **BORODA**, 390 005
Tel: 65235, 65236
Cable: BLUE STAR
A

Blue Star Ltd.
7 Hare Street
P.O. Box 506
CALCUTTA 700 001
Tel: 230131, 230132
Telex: 031-61120 BSNF IN
Cable: BLUESTAR
A,M,C,E

Blue Star Ltd.
133 Kodambakkam High Road
MADRAS 600 034
Tel: 472056, 470238
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
13 Community Center
New Friends Colony
NEW DELHI 110 065
Tel: 682547
Telex: 031-2463
Cable: BLUEFROST
A,C*,CM,E,M

Blue Star Ltd.
15/16 C Wellesley Rd.
PUNE 411 011
Tel: 22775
Cable: BLUE STAR
A

Blue Star Ltd.
2-2-47/1108 Bolarum Rd.
SECUNDERABAD 500 003
Tel: 72057, 72058
Telex: 0155-459
Cable: BLUEFROST
A,C,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthunkuzhi
TRIVANDRUM 695 013
Tel: 65799, 65820
Telex: 0884-259
Cable: BLUESTAR
E

Computer Maintenance Corporation Ltd.
115, Sarojini Devi Road
SECUNDERABAD 500 003
Tel: 310-184, 345-774
Telex: 031-2960
C**

INDONESIA

BERCA Indonesia P.T.
P.O.Box 496/Jkt.
Jl. Abdul Muis 62
JAKARTA
Tel: 21-373009
Telex: 46748 BERSAL IA
Cable: BERSAL JAKARTA
P

BERCA Indonesia P.T.
P.O.Box 2497/Jkt
Antara Bldg., 12th Floor
Jl. Medan Merdeka Selatan 17
JAKARTA-PUSAT
Tel: 21-340417
Telex: 46748 BERSAL IA
A,C,E,M,P

BERCA Indonesia P.T.
Jalan Kutai 24
SURABAYA
Tel: 67118
Telex: 31146 BERSAL SB
Cable: BERSAL-SURABAYA
A*,E,M,P

IRAQ

Hewlett-Packard Trading S.A.
Service Operation
Al Mansoor City 9B/3/7
BAGHDAD
Tel: 551-49-73
Telex: 212-455 HEPAIRAQ IK
C

IRELAND

Hewlett-Packard Ireland Ltd.
Temple House, Temple Road
Blackrock, Co. **DUBLIN**
Tel: 88/333/99
Telex: 30439
C,E,P

Hewlett-Packard Ltd.
75 Belfast Rd, Carrickfergus
Belfast BT38 8PH
NORTHERN IRELAND
Tel: 09603-67333
Telex: 747626
M

ISRAEL

Eldan Electronic Instrument Ltd.
P.O.Box 1270
JERUSALEM 91000
16, Ohaliav St.
JERUSALEM 94467
Tel: 533 221, 553 242
Telex: 25231 AB/PAKRD IL
A,M

Computation and Measurement Systems (CMS) Ltd.
11 Masad Street
67060
TEL-AVIV
Tel: 388 388
Telex: 33569 Motil IL
C,CM,E,P

ITALY

Hewlett-Packard Italiana S.p.A
Traversa 99C
Via Giulio Petroni, 19
I-70124 **BARI**
Tel: (080) 41-07-44
C,M

Hewlett-Packard Italiana S.p.A.
Via Emilia, 51/C
I-40011 **BOLOGNA** Anzola Dell'Emilia
Tel: 39-051-731061
Telex: 511630
C,E,M

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87
Telex: 970291
C

Hewlett-Packard Italiana S.p.A.
Via G. di Vittorio 10
20094 **CORSICO** (Milano)
Tel: 39-02-4408351

Hewlett-Packard Italiana S.p.A.
Viale Brigata Bisagno 2
16129 **GENOVA**
Tel: 39-10-541141
Telex: 215238

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07
Telex: 215238
C,E

Hewlett-Packard Italiana S.p.A.
Via G. di Vittorio 9
I-20063 **CERNUSCO SUL NAVIGLIO**
(Milano)
Tel: (02) 923691
Telex: 334632
A,C,CM,E,M,P

Hewlett-Packard Italiana S.p.A.
Via Nuova Rivoltana 95
20090 **LIMITO** (Milano)
Tel: 02-92761

Hewlett-Packard Italiana S.p.A.
Via Nuova San Rocco a Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
Telex: 710698
A**,C,E,M

Hewlett-Packard Italiana S.p.A.
Via Orazio 16
80122 **NAPOLI**
Tel: (081) 7611444
Telex: 710698

Hewlett-Packard Italiana S.p.A.
Via Pellizzo 15
35128 **PADOVA**
Tel: 39-49-664-888
Telex: 430315
A,C,E,M

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA EUR**
Tel: 39-65-48-31
Telex: 610514
A,C,E,M,P*

Hewlett-Packard Italiana S.p.A.
Via di Casellina 57/C
500518 **SCANDICCI-FIRENZE**
Tel: 39-55-753863
C,E,M

Hewlett-Packard Italiana S.p.A.
Corso Svizzera, 185
I-10144 **TORINO**
Tel: 39-11-74-4044
Telex: 221079
A*,C,E

IVORY COAST

S.I.T.E.L.
Societe Ivoirienne de Telecommunications
Bd. Giscard d'Estaing
Carrefour Marcory
Zone 4.A.
Boite postale 2580
ABIDJAN 01
Tel: 353600
Telex: 43175
E

S.I.T.I.
Immeuble "Le General"
Av. du General de Gaulle
01 BP 161
ABIDJAN 01
Tel: 321227
Telex: 22149
C,P

JAPAN

Yokogawa-Hewlett-Packard Ltd.
152-1, Onna
ATSUGI, Kanagawa, 243
Tel: (0462) 25-0031
C,CM,E

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei Bldg. 6F
3-1 Motochiba-Cho
CHIBA, 280
Tel: (0472) 25 7701
C,E

Yokogawa-Hewlett-Packard Ltd.
Yasuda-Seimei Hiroshima Bldg.
6-11, Hon-dori, Naka-ku
HIROSHIMA, 730
Tel: (082) 241-0611

Yokogawa-Hewlett-Packard Ltd.
Towa Building
2-2-3 Kaigan-dori, Chuo-ku
KOBE, 650
Tel: (078) 392-4791
C,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi 82 Bldg.
3-4 Tsukuba
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
C,CM,E

Yokogawa-Hewlett-Packard Ltd.
Asahi Shinbun Daiichi Seimei Bldg.
4-7, Hanabata-cho
KUMAMOTO, 860
Tel: 96-354-7311
C,E

Yokogawa-Hewlett-Packard Ltd.
Shin-Kyoto Center Bldg.
614, Higashi-Shiokoji-cho
Karasuma-Nishiiru
KYOTO, 600
Tel: 075-343-0921
C,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Bldg.
1-4-73, Sanno-maru
MITO, Ibaraki 310
Tel: (0292) 25-7470
C,CM,E

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei Kokubun Bldg.
7-8 Kokubun, 1 Chome, Sendai
MIYAGI, 980
Tel: (0222) 25-1011
C,E

Yokogawa-Hewlett-Packard Ltd.
Gohda Bldg. 2F
1-2-10 Gohda Okaya-Shi
Okaya-Shi
NAGANO, 394
Tel: (0266) 23 0851
C,E

Yokogawa-Hewlett-Packard Ltd.
Nagoya Kokusai Center Building
1-47-1, Nagono, Nakamura-ku
NAGOYA, AICHI 450
Tel: (052) 571-5171
C,CM,E,M

Yokogawa-Hewlett-Packard Ltd.
Sai-Kyo-Ren Building
1-2 Dote-cho
OOMIYA-SHI SAITAMA 330
Tel: (0486) 45-8031

JAPAN (Cont'd)

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 5-4-20 Nishi-Nakajima
4-20 Nishinakajima, 5 Chome,
Yodogawa-ku
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
C,CM,E,M,P*

Yokogawa-Hewlett-Packard Ltd.
1-27-15, Yabe
SAGAMIHARA Kanagawa, 229
Tel: 0427 59-1311

Yokogawa-Hewlett-Packard Ltd.
Hamamtsu Motoshiro-Cho Daichi
Seimei Bldg 219-21, Motoshiro-Cho
Hamamatsu-shi
SHIZUOKA, 430
Tel: (0534) 56 1771
C,E

Yokogawa-Hewlett-Packard Ltd.
Shinjuku Daiichi Seimei Bldg.
2-7-1, Nishi Shinjuku
Shinjuku-ku,**TOKYO** 163
Tel: 03-348-4611
C,E,M

Yokogawa Hewlett-Packard Ltd.
9-1, Takakura-cho
Hachioji-shi, **TOKYO**, 192
Tel: 81-426-42-1231
C,E

Yokogawa-Hewlett-Packard Ltd.
3-29-21 Takaido-Higashi, 3 Chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
C,CM,E,P*

Yokogawa Hokushin Electric
Corporation
Shinjuku-NS Bldg. 10F
4-1 Nishi-Shinjuku 2-Chome
Shinjuku-ku
TOKYO, 163
Tel: (03) 349-1859
Telex: J27584
A

Yokogawa Hokushin Electric Corp.
9-32 Nokacho 2 Chome
Musashino-shi
TOKYO, 180
Tel: (0422) 54-1111
Telex: 02822-421 YEW MTK J
A

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei
Utsunomiya Odori Building
1-5 Odori, 2 Chome
UTSUNOMIYA, Tochigi 320
Tel: (0286) 33-1153
C,E

Yokogawa-Hewlett-Packard Ltd.
Yasuda Seimei Nishiguchi Bldg.
30-4 Tsuruya-cho, 3 Chome
Kanagawa-ku, **YOKOHAMA** 221
Tel: (045) 312-1252
C,CM,E

JORDAN

Scientific and Medical Supplies Co.
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
C,E,M,P

KENYA

ADCOM Ltd., Inc., Kenya
P.O.Box 30070
NAIROBI
Tel: 331955
Telex: 22639
E,M

KOREA

Samsung Hewlett-Packard Co. Ltd.
Dongbang Yeoeuido Building
12-16th Floors
36-1 Yeoeuido-Dong
Youngdeungpo-Ku
SEOUL
Tel: 784-4666, 784-2666
Telex: 25166 SAMSAN K
C,CM,E,M,P

Young In Scientific Co., Ltd.
Youngwha Building
547 Shinsa Dong, Kangnam-Ku
SEOUL 135
Tel: 546-7771
Telex: K23457 GINSCO
A

Dongbang Healthcare
Products Co. Ltd.
Suite 301 Medical Supply Center
Bldg. 1-31 Dongsungdong
Jong Ro-gu, **SEOUL**
Tel: 764-1171, 741-1641
Telex: K25706 TKBKO
Cable: TKBEEPKO
M

KUWAIT

Al-Khaldiya Trading & Contracting
P.O. Box 830
SAFAT
Tel: 424910, 411726
Telex: 22481 AREEG KT
Cable: VISCOUNT
E,M,A

Gulf Computing Systems
P.O. Box 25125
SAFAT
Tel: 435969
Telex: 23648
P

Photo & Cine Equipment
P.O. Box 270
SAFAT
Tel: 2445111
Telex: 22247 MATIN KT
Cable: MATIN KUWAIT
P

W.J. Towell Computer Services
P.O. Box 5897
SAFAT
Tel: 2462640/1
Telex: 30336 TOWELL KT
C

LEBANON

Computer Information Systems S.A.L.
Chammas Building
P.O. Box 11-6274 Dora
BEIRUT
Tel: 89 40 73
Telex: 42309 chacis le
C,E,M,P

LIBERIA

Unichemicals Inc.
P.O. Box 4509
MONROVIA
Tel: 224282
Telex: 4509
E

LUXEMBOURG

Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 **BRUSSELS**
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,C,CM,E,M,P

MADAGASCAR

Technique et Precision
12, rue de Nice
P.O. Box 1227
101 **ANTANANARIVO**
Tel: 22090
Telex: 22255
P

MALAYSIA

Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
9th Floor
Chung Khiaw Bank Building
46, Jalan Raja Laut
50736 **KUALA LUMPUR, MALAYSIA**
Tel: 03-2986555
Telex: 31011 HPSM MA
A,C,E,M,P*

Protel Engineering
P.O.Box 1917
Lot 6624, Section 64
23/4 Pending Road
Kuching, **SARAWAK**
Tel: 36299
Telex: 70904 PROMAL MA
Cable: PROTELENG
A,E,M

MALTA

Philip Toledo Ltd.
Kirkirkara P.O. Box 11
Notabile Rd.
MRIEHEL
Tel: 447 47, 455 66, 4915 25
Telex: Media MW 649
E,M,P

MAURITIUS

Blanche Birger Co. Ltd.
18, Jules Koenig Street
PORT LOUIS
Tel: 20828
Telex: 4296
P

MEXICO

Hewlett-Packard de Mexico,
S.A. de C.V.
Rio Nio No. 4049 Desp. 12
Fracc. Cordoba
JUAREZ
Tel: 161-3-15-62
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Condominio Kadereyta
Circuito del Mezon No. 186 Desp. 6
COL. DEL PRADO - 76030 Qro.
Tel: 463-6-02-71
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Monti Morelos No. 299
Fraccionamiento Loma Bonita 45060
GUADALAJARA, Jalisco
Tel: 36-31-48-00
Telex: 0684 186 ECOME
P

Microcomputadoras
Hewlett-Packard, S.A.
Monti Pelvoux 115
LOS LOMAS, Mexico, D.F.
Tel: 520-9127
P

Microcomputadoras Hewlett-Packard,
S.A. de C.V.
Monte Pelvoux No. 115
Lomas de Chapultepec, 11000
MEXICO, D.F.
Tel: 520-9127
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Monte Pelvoux No. 111
Lomas de Chapultepec
11000 **MEXICO, D.F.**
Tel: 5-40-62-28, 72-66, 50-25
Telex: 17-74-507 HEWPACK MEX
A,C,CM,E,M,P

Hewlett-Packard De Mexico (Polanco)
Avenida Ejercito Nacional #579
2da/3er piso
Colonia Granada 11560
MEXICO D.F.
Tel: 254-4433
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Czda. del Valle
409 Ote. 4th Piso
Colonia del Valle
Municipio de Garza
Garcia Nuevo Leon
66220 **MONTERREY**, Nuevo León
Tel: 83-78-42-40
Telex: 382410 HPMY
C

Infograficas y Sistemas
del Noreste, S.A.
Rio Orinoco #171 Oriente
Despacho 2001
Colonia Del Valle
MONTERREY
Tel: 559-4415, 575-3837
Telex: 483164
A,E

Hewlett-Packard de Mexico,
S.A. de C.V.
Blvd. Independencia No. 2000 Ote.
Col. Estrella
TORREON, COAH.
Tel: 171-18-21-99
P

MOROCCO

Etablissement Hubert Dolbeau & Fils
81 rue Karatchi
B.P. 11133
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2, rue Agadir
Boite Postale 156
CASABLANCA 01
Tel: 272093, 272095
Telex: 23 739
P

Sema-Maroc
Dept. Seric
6, rue Lapebie
CASABLANCA
Tel: 260980
Telex: 21641
C,P

NETHERLANDS

Hewlett-Packard Nederland B.V.
Startbaan 16
NL-1187 XR **AMSTELVEEN**
P.O. Box 667
NL-1180 AR **AMSTELVEEN**
Tel: (020) 547-6911
Telex: 13 216 HEPA NL
A,C,CM,E,M,P

Hewlett-Packard Nederland B.V.
Bongerd 2
P.O. Box 41
NL 2900AA **CAPELLE A/D IJSSEL**
Tel: 31-20-51-6444
Telex: 21261 HEPAC NL
C,E

Hewlett-Packard Nederland B.V.
Pastoor Petersstraat 134-136
P.O. Box 2342
NL 5600 CH **EINDHOVEN**
Tel: 31-40-32-6911
Telex: 51484 hepae nl
C,E,P

NEW ZEALAND

Hewlett-Packard (N.Z.) Ltd.
5 Owens Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 64-9-687-159
Cable: HEWPAK Auckland
C,CM,E,P*

Hewlett-Packard (N.Z.) Ltd.
184-190 Willis Street
WELLINGTON
P.O. Box 9443
Courtenay Place, **WELLINGTON 3**
Tel: 64-4-887-199
Cable: HEWPACK Wellington
C,CM,E,P

Northrop Instruments & Systems Ltd.
369 Khyber Pass Road
P.O. Box 8602
AUCKLAND
Tel: 794-091
Telex: 60605
A,M

Northrop Instruments & Systems Ltd.
110 Mandeville St.
P.O. Box 8388
CHRISTCHURCH
Tel: 488-873
Telex: 4203
A,M

Northrop Instruments & Systems Ltd.
Sturdee House
85-87 Ghuznee Street
P.O. Box 2406
WELLINGTON
Tel: 850-091
Telex: NZ 3380
A,M

NIGERIA
Elmeco Nigeria Ltd.
45 Saka Tirubu St.
Victoria Island
LAGOS
Tel: 61-98-94
Telex: 20-117
E

NORTHERN IRELAND
See United Kingdom

NORWAY
Hewlett-Packard Norge A/S
Folke Bernadottes vei 50
P.O. Box 3558
N-5033 **FYLLINGSDALEN** (Bergen)
Tel: 0047/5/16 55 40
Telex: 76621 hpnas n
C,E,M

Hewlett-Packard Norge A/S
Osterndalen 16-18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: 47-2-17-1180
Telex: 76621 hpnas n
A,C,CM,E,M,P

Hewlett-Packard Norge A/S
Boehmergt. 42
Box 2470
N-5037 **SOLHEIMSVIK**
Tel: 0047/5/29 00 90

OMAN
Khimjil Ramdas
P.O. Box 19
MUSCAT/SULTANATE OF OMAN
Tel: 795 901
Telex: 3489 BROKER MB MUSCAT
P

Suhail & Saud Bahwan
P.O.Box 169
MUSCAT/SULTANATE OF OMAN
Tel: 734 201-3
Telex: 5274 BAHWAN MB
E

Imtac LLC
P.O. Box 9196
MINA AL FAHAL/SULTANATE OF OMAN
Tel: 70-77-27, 70-77-23
Telex: 3865 Tawoos On
A,C,M

PAKISTAN
Mushko & Company Ltd.
House No. 16, Street No. 16
Sector F-6/3
ISLAMABAD
Tel: 824545
Telex: 54001 Muski Pk
Cable: FEMUS Islamabad
A,E,P*

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
KARACHI 0302
Tel: 524131, 524132
Telex: 2894 MUSKO PK
Cable: COOPERATOR Karachi
A,E,P*

PANAMA
Electronico Balboa, S.A.
Calle Samuel Lewis, Ed. Alfa
Apartado 4929
PANAMA CITY
Tel: 9-011-507-636613
Telex: 368 3483 ELECTRON PG
CM,E,M,P

PERU
Cia Electro Médica S.A.
Los Flamencos 145, Ofc. 301/2
San Isidro
Casilla 1030
LIMA 1
Tel: 9-011-511-4-414325, 41-3705
Telex: 39425257 PE PB SIS
CM,E,M,P

SAMS S.A.
Arenida Republica de Panama 3534
San Isidro, **LIMA**
Tel: 9-011-511-4-229332/413984/413226
Telex: 39420450 PE LIBERTAD
A,C,P

PHILIPPINES
The Online Advanced Systems Corp.
2nd Floor, Electra House
115-117 Esteban Street
P.O. Box 1510
Legaspi Village, Makati
Metro **MANILA**
Tel: 815-38-10 (up to 16)
Telex: 63274 ONLINE PN
A,C,E,M,P

PORTUGAL
Mundinter Intercambio
Mundial de Comércio S.A.R.L.
Av. Antonio Augusto Aguiar 138
Apartado 2761
LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

Soquimica
Av. da Liberdade, 220-2
1298 **LISBOA** Codex
Tel: 56-21-82
Telex: 13316 SABASA
A

Telectra-Empresa Técnica de
Equipmentos Eléctricos S.A.R.L.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
LISBON 1
Tel: (19) 68-60-72
Telex: 12598
CM,E

C.P.C.S.I.
Rua de Costa Cabral 575
4200 **PORTO**
Tel: 499174/495173
Telex: 26054
C,P

PUERTO RICO
Hewlett-Packard Puerto Rico
101 Muñoz Rivera Av
Esu. Calle Ochoa
HATO REY, Puerto Rico 00918
Tel: (809) 754-7800
A,C,CM,M,E,P

QATAR
Computer Arabia
P.O. Box 2750
DOHA
Tel: 428555
Telex: 4806 CHPARB
P

Nasser Trading & Contracting
P.O.Box 1563
DOHA
Tel: 422170
Telex: 4439 NASSER DH
M

SAUDI ARABIA
Modern Electronics Establishment
Hewlett-Packard Division
P.O. Box 281
Thuobah
AL-KHOBAR 31952
Tel: 895-1760, 895-1764
Telex: 671 106 HPMEEK SJ
Cable: ELECTA AL-KHOBAR
C,E,M

Modern Electronics Establishment
Hewlett-Packard Division
P.O. Box 1228
Redec Plaza, 6th Floor
JEDDAH
Tel: 644 96 28
Telex: 4027 12 FARNAS SJ
Cable: ELECTA JEDDAH
A,C,CM,E,M,P

Modern Electronics Establishment
Hewlett-Packard Division
P.O.Box 22015
RIYADH 11495
Tel: 491-97 15, 491-63 87
Telex: 202049 MEERYD SJ
C,E,M

Abdul Ghani El Ajou Corp.
P.O. Box 78
RIYADH
Tel: 40 41 717
Telex: 200 932 EL AJOU
P

SCOTLAND
See United Kingdom

SENEGAL
Societe Hussein Ayad & Cie.
76, Avenue Georges Pompidou
B.P. 305
DAKAR
Tel: 32339
Cable: AYAD-Dakar
E

Moneger Distribution S.A.
1, Rue Parent
B.P. 148
DAKAR
Tel: 215 671
Telex: 587
P

Systeme Service Conseil (SSC)
14, Avenue du Parachois
DAKAR ETOILE
Tel: 219976
Telex: 577
C,P

SINGAPORE
Hewlett-Packard Singapore (Sales) Pte. Ltd.
1150 Depot Road
SINGAPORE, 0410
Tel: 4731788
Telex: 34209 HPSGSO RS
Cable: HEWPACK, Singapore
A,C,E,M,P

Dynamar International Ltd.
Unit 05-11 Block 6
Kolam Ayer Industrial Estate
SINGAPORE 1334
Tel: 747-6188
Telex: 26283 RS
CM

SOUTH AFRICA
Hewlett-Packard So Africa (Pty.) Ltd.
P.O. Box 120
Howard Place, **CAPE PROVINCE**
7450 South Africa
Tel: 27 121153-7954
Telex: 57-20006
A,C,CM,E,M,P

Hewlett-Packard So Africa (Pty.) Ltd.
2nd Floor Juniper House
92 Overport Drive
DURBAN 4067
Tel: 27-31-28-4178
Telex: 6-22954
C

Hewlett-Packard So Africa (Pty.) Ltd.
Shop 6 Linton Arcade
511 Cape Road
Linton Grange
PORT ELIZABETH 6001
Tel: 27141130 1201
Telex: 24-2916
C

Hewlett-Packard So Africa (Pty.) Ltd.
Fountain Center
Kalkoen Str.
Monument Park Ext 2
PRETORIA 0105
Tel: (012) 45 5725
Telex: 32163
C,E

Hewlett-Packard So Africa (Pty.) Ltd.
Private Bag Wendywood
SANDTON 2144
Tel: 27-11-802-5111, 27-11-802-5125
Telex: 4-20877 SA
Cable: HEWPACK Johannesburg
A,C,CM,E,M,P

SPAIN
Hewlett-Packard Española, S.A.
Calle Entenza, 321
E.-BARCELONA 29
Tel: 3/322 24 51, 321 73 54
Telex: 52603 hpbee
A,C,E,M,P

Hewlett-Packard Española, S.A.
Calle San Vicente S/N
Edificio Albia II-7B
48001 **BILBAO**
Tel: 4/423 83 06
A,C,E,M

Hewlett-Packard Española, S.A.
Crta. N-VI, Km. 16, 400
Las Rozas
E-MADRID
Tel: (1) 637.00.11
Telex: 23515 HPE
C,M

Hewlett-Packard Española, S.A.
Avda. S. Francisco Javier, S/N
Planta 10. Edificio Sevilla 2
E-SEVILLA 5, SPAIN
Tel: 54/64 44 54
Telex: 72933
A,C,M,P

Hewlett-Packard Española, S.A.
Isabel La Catolica, 8
E-46004 **VALENCIA**
Tel: 34-6-361 1354
Telex: 63435
C,P

Hewlett-Packard Española, S.A.
Av. de Zugazarte, 8
Las Arenas-Guecho
E-48930 **VIZCAYA**
VIZCAYA
Tel: 34-423-83 06
Telex: 33032

SWEDEN
Hewlett-Packard Sverige AB
Östra Tullgatan 3
S-20011 **MALMÖ**
Box 6132
Tel: 46-40-702-70
Telex: (854) 17886 (via Spånga office)
C,P

Hewlett-Packard Sverige AB
Elementvagen 16
S-7022 7 **ÖREBRO**
Tel: 49-019-10-4820
Telex: (854) 17886 (via Spånga office)
C

Hewlett-Packard Sverige AB
Skalholtsgatan 9, Kista
P.O. Box 19
S-16393 **SPÅNGA**
Tel: (08) 750-2000
Telex: (854) 17886
Telefax: (08) 7527781
A,C,CM,E,M,P

Hewlett-Packard Sverige AB
Box 266
Topasgatan 1A
S-42123 **VÄSTRA-FRÖLUNDA**
(Gothenburg)
Tel: 46-031-89-1000
Telex: (854) 17886 (via Spånga office)
A,C,CM,E,M,P

SUDAN
Mediterranean Engineering & Trading Co. Ltd.
P.O. Box 1025
KHARTOUM
Tel: 41184
Telex: 24052
C,P

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASEL**
Tel: 41-61-33-5920
A,C,E,P

Hewlett-Packard (Schweiz) AG
7, rue du Bois-du-Lan
Case postale 365-1366
CH-1217 **MEYRIN** 1
Tel: (0041) 22-83-11-11
Telex:27333 HPAG CH
A,C,CM,E,M,P

SWITZERLAND (Cont'd)
Hewlett-Packard (Schweiz) AG
Allmend 2
CH-8967 **WIDEN**
Tel: 41-57-31-2111
Telex: 53933 hpag ch
Cable: HPAG CH
A,C,CM,E,M,P

Hewlett-Packard (Schweiz) AG
Schwamendingenstrasse 10
CH-8050 **ZURICH**
Tel: 41-1-315-8181
Telex: 823 537 HPAG CH
C,P

SYRIA
General Electronic Inc.
Nuri Basha Ahnaf Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 44-19-88
Cable: ELECTROBOR DAMASCUS
E

Middle East Electronics
P.O.Box 2308
Abu Rumaneh
DAMASCUS
Tel: 33 45 92
Telex: 411 771 Meesy
M

TAIWAN
Hewlett-Packard Taiwan Ltd.
THM Office
2, Huan Nan Road
CHUNG LI, Taoyuan
Tel: (034) 929-666
C

Hewlett-Packard Taiwan Ltd.
Kaohsiung Office
11/F, 456, Chung Hsiao 1st Road
KAOHSIUNG
Tel: (07) 2412318
C,E

Hewlett-Packard Taiwan Ltd.
8th Floor, Hewlett-Packard Building
337 Fu Hsing North Road
TAIPEI
Tel: (02) 712-0404
Telex: 24439 HEWPACK
Cable:HEWPACK Taipei
A,C,CM,E,M,P

Ing Lih Trading Co.
3rd Floor, No. 7, Sect. 2
Jen Ai Road
TAIPEI 100
Tel: (02) 394-8191
Telex: 22894 SANKWANG
A

THAILAND
Unimesa Co. Ltd.
30 Patpong Ave., Suriwong
BANGKOK 5,
Tel: 235-5727, 234-0991/3
Telex: 84439 Simonco TH
Cable: UNIMESA Bangkok
A,C,E,M

Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671
Telex: 87699-BEQUIPT TH
Cable: BUSIQUIPT Bangkok
P

TOGO
Societe Africaine De Promotion
Immeuble Sageb
Rue d'Atakpame
P.O. Box 4150
LOME
Tel: 21-62-88
Telex: 5357
P

TRINIDAD & TOBAGO
Caribbean Telecoms Ltd.
Corner McAllister Street &
Eastern Main Road, Laventille
P.O. Box 732
PORT-OF-SPAIN
Tel: 624-4213
Telex: 22561 CARTEL WG
Cable: CARTEL, PORT OF SPAIN
CM,E,M,P

Computer and Controls Ltd.
P.O. Box 51
1 Taylor Street
PORT-OF-SPAIN
Tel: (809) 622-7719/622-7985
Telex: 38722798 COMCON WG
LOOGO AGENCY 1264
A,P

Feral Assoc.
8 Fitzgerald Lane
PORT-OF-SPAIN
Tel: 62-36864, 62-39255
Telex: 22432 FERALCO
Cable: FERALCO
M

TUNISIA
Tunisie Electronique S.A.R.L.
31 Avenue de la Liberte
TUNIS
Tel: 280-144
C,E,P

Tunisie Electronique S.A.R.L.
94, Av. Jugurtha, Mutuelleville
1002 **TUNIS-BELVEDERE**
Tel: 280144
Telex: 13238
C,E,P

Corema S.A.
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY
E.M.A
Mediha Eldem Sokak No. 41/6
Yenisehir
ANKARA
Tel: 319175
Telex: 42321 KTX TR
Cable: EMATRADE ANKARA
M

Teknim Company Ltd.
Iran Caddesi No. 7
Karaklidere
ANKARA
Tel: 275800
Telex: 42155 TKNM TR
C,E

Kurt & Kurt A.S.
Mithatpasa Caddesi No. 75
Kat 4 Kizilay
ANKARA
Tel: 318875/6/7/8
Telex: 42490 MESR TR
A

Saniva Bilgisayar Sistemleri A.S.
Buyukdere Caddesi 103/6
Gayrettepe
ISTANBUL
Tel: 1673180
Telex: 26345 SANI TR
C,P

Best Inc.
Esentepe, Gazeteciler Sitesi
Keskin Kalem
Sokak 6/3, Gayrettepe
ISTANBUL
Tel: 172 1328, 173 3344
Telex: 42490
A

UNITED ARAB EMIRATES
Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 591181
Telex: 68136 EMITAC EM
Cable: EMITAC SHARJAH
E,C,M,P,A

Emitac Ltd.
P.O. Box 2711
ABU DHABI
Tel: 820419-20
Cable: EMITACH ABUDHABI

Emitac Ltd.
P.O. Box 8391
DUBAI,
Tel: 377591

Emitac Ltd.
P.O. Box 473
RAS AL KHAIMAH
Tel: 28133, 21270

UNITED KINGDOM
ENGLAND
Hewlett-Packard Ltd.
Miller House
The Ring, **BRACKNELL**
Berks RG12 1XN
Tel: 44/344/424-898
Telex: 848733
E

Hewlett-Packard Ltd.
Elstree House, Elstree Way
BOREHAMWOOD, Herts WD6 1SG
Tel: 01 207 5000
Telex: 8952716
C,E

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton **BRISTOL**, Avon BS8 2BN
Tel: 44-272-736 806
Telex: 444302
C,E,P

Hewlett-Packard Ltd.
9 Bridewell Place
LONDON EC4V 6BS
Tel: 44-01-583-6565
Telex: 298163
C,P

Hewlett-Packard Ltd.
Pontefract Road
NORMANTON, West Yorkshire WF6 1RN
Tel: 44/924/895 566
Telex: 557355
C,P

Hewlett-Packard Ltd.
The Quadrangle
106-118 Station Road
REDHILL, Surrey RH1 1PS
Tel: 44-737-686-55
Telex: 947234
C,E,P

Hewlett-Packard Ltd.
Avon House
435 Stratford Road
Shirley, **SOLIHULL**, West Midlands
B90 4BL
Tel: 44-21-745-8800
Telex: 339105
C,E,P

Hewlett-Packard Ltd.
Heathside Park Road
Cheadle Heath, Stockport
SK3 ORB, United Kingdom
Tel: 44-061-428-0828
Telex: 668068
A,C,E,M,P

Hewlett-Packard Ltd.
Harmon House
No. 1 George Street
UXBRIDGE, Middlesex UX8 1YH
Tel: 895 720 20
Telex: 893134/5
C,CM,E,M,P

Hewlett-Packard Ltd.
King Street Lane
Winnersh, **WOKINGHAM**
Berkshire RG11 5AR
Tel: 44/734/784774
Telex: 8471789
A,C,E,M,P

NORTHERN IRELAND
Hewlett-Packard (Ireland) Ltd.
Carrickfergus Industrial Centre
75 Belfast Road, Carrickfergus
CO. ANTRIM BT38 8PM
Tel: 09603 67333
C,E

Cardiac Services Company
95A Finaghy Road South
BELFAST, BT10 OBY
Tel: 0232-625566
Telex: 747626
M

SCOTLAND
Hewlett-Packard Ltd.
1/3 Springburn Place
College Milton North
EAST KILBRIDE, G74 5NU
Tel: 041-332-6232
Telex: 779615
C,E

Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
Tel: 031 331 1188
Telex: 72682 HPSQFYG
C,CM,E,M,P

UNITED STATES
Hewlett-Packard Co.
Customer Information Center
Tel: (800) 752-0900
Hours: 6:00 AM to 5:00 PM
Pacific Time

Alabama
Hewlett-Packard Co.
2100 Riverchase Center
Building 100 - Suite 118
BIRMINGHAM, AL 35244
Tel: (205) 988-0547
A,C,M,P*

Hewlett-Packard Co.
420 Wynn Drive
HUNTSVILLE, AL 35805
Tel: (205) 830-2000
C,CM,E,M*

Alaska
Hewlett-Packard Co.
4000 Old Seward Highway
Suite 101
ANCHORAGE, AK 99503
Tel: (907) 563-8855
C,E

Arizona
Hewlett-Packard Co.
8080 Pointe Parkway West
PHOENIX, AZ 85044
Tel: (602) 273-8000
A,C,CM,E,M,P

Hewlett-Packard Co.
3400 East Britannia Dr.
Bldg. C, Suite 124
TUCSON, AZ 85706
Tel: (602) 573-7400
C,E,M**

California
Hewlett-Packard Co.
99 South Hill Dr.
BRISBANE, CA 94005
Tel: (415) 330-2500
C

Hewlett-Packard Co.
1907 North Gateway Blvd.
FRESNO, CA 93727
Tel: (209) 252-9652
C,M

Hewlett-Packard Co.
1421 S. Manhattan Av.
FULLERTON, CA 92631
Tel: (714) 999-6700
C,CM,E,M

Hewlett-Packard Co.
7408 Hollister Ave. #A
GOLETA, CA 93117
Tel: (805) 685-6100
C,E

Hewlett-Packard Co.
2525 Grand Avenue
LONG BEACH, CA 90815
Tel: (213) 498-1111
C

Hewlett-Packard Co.
5651 West Manchester Ave.
LOS ANGELES, CA 90045
Tel: (213) 337-8000

Hewlett-Packard Co.
3155 Porter Drive
PALO ALTO, CA 94304
Tel: (415) 857-8000
C,E

Hewlett-Packard Co.
5725 W. Las Positas Blvd.
PLEASANTON, CA 94566
Tel: (415) 460-0282
C

Hewlett-Packard Co.
4244 So. Market Court, Suite A
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,C,E,M

Hewlett-Packard Co.
9606 Aero Drive
SAN DIEGO, CA 92123
Tel: (619) 279-3200
C,CM,E,M

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95054
Tel: (408) 988-7000
Telex: 910-338-0586
A,C,CM,E

Hewlett-Packard Co.
2150 W. Hillcrest Dr.
THOUSAND OAKS, CA 91320
(805) 373-7000
C,CM,E

Colorado

Hewlett-Packard Co.
2945 Center Green Court South
Suite A
BOULDER, CO 80301
Tel: (303) 499-6655
A,C,E

Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 649-5000
A,C,CM,E,M

Connecticut

Hewlett-Packard Co.
500 Sylvan Av.
BRIDGEPORT, CT 06606
Tel: (203) 371-6454
C,E

Hewlett-Packard Co.
47 Barnes Industrial Road South
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,C,CM,E,M

Florida

Hewlett-Packard Co.
2901 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
C,E,M,P*

Hewlett-Packard Co.
6800 South Point Parkway
Suite 301
JACKSONVILLE, FL 32216
Tel: (904) 636-9955
C*,M**

Hewlett-Packard Co.
255 East Drive, Suite B
MELBOURNE, FL 32901
Tel: (305) 729-0704
CM,E

Hewlett-Packard Co.
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,C,CM,E,P*

Hewlett-Packard Co.
4700 Bayou Blvd.
Building 5
PENSACOLA, FL 32503
Tel: (904) 476-8422
A,C,M

Hewlett-Packard Co.
5550 W. Idlewild, #150
TAMPA, FL 33614
Tel: (813) 884-3282
C,E,M,P

Georgia

Hewlett-Packard Co.
2015 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,C,CM,E,M,P*

Hewlett-Packard Co.
3607 Parkway Lane
Suite 300
NORCROSS, GA 30092
Tel: (404) 448-1894
C,E,P

Hawaii

Hewlett-Packard Co.
Pacific Tower
1001 Bishop St.
Suite 2400
HONOLULU, HI 96813
Tel: (808) 526-1555
A,C,E,M

Idaho

Hewlett-Packard Co.
11309 Chinden Blvd.
BOISE, ID 83714
Tel: (208) 323-2700
C

Illinois

Hewlett-Packard Co.
2205 E. Empire St.
P.O. Box 1607
BLOOMINGTON, IL 61702-1607
Tel: (309) 662-9411
A,C,E,M**

Hewlett-Packard Co.
525 W. Monroe, #1308
CHICAGO, IL 60606
Tel: (312) 930-0010
C

Hewlett-Packard Co.
1200 East Diehl Road
NAPERVILLE, IL 60566
Tel: (312) 357-8800
C

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
Telex: 910-687-1066
A,C,CM,E,M

Indiana

Hewlett-Packard Co.
11911 N. Meridian St.
CARMEL, IN 46032
Tel: (317) 844-4100
A,C,CM,E,M

Hewlett-Packard Co.
111 E. Ludwig Road
Suite 108
FT. WAYNE, IN 46825
Tel: (219) 482-4283
C,E

Iowa

Hewlett-Packard Co.
4070 22nd Av. SW
CEDAR RAPIDS, IA 52404
Tel: (319) 390-4250
C,E,M

Hewlett-Packard Co.
4201 Corporate Dr.
WEST DES MOINES, IA 50265
Tel: (515) 224-1435
A**,C,M**

Kansas

Hewlett-Packard Co.
North Rock Business Park
3450 N. Rock Rd.
Suite 300
WICHITA, KS 67226
Tel: (316) 684-8491
C,E

Kentucky

Hewlett-Packard Co.
305 N. Hurstbourne Lane,
Suite 100
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,C,M

Louisiana

Hewlett-Packard Co.
160 James Drive East
ST. ROSE, LA 70087
P.O. Box 1449
KENNER, LA 70063
Tel: (504) 467-4100
A,C,E,M,P

Maryland

Hewlett-Packard Co.
3701 Koppers Street
BALTIMORE, MD 21227
Tel: (301) 644-5800
Telex: 710-862-1943
A,C,CM,E,M

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
A,C,CM,E,M

Massachusetts

Hewlett-Packard Co.
1775 Minuteman Road
ANDOVER, MA 01810
Tel: (617) 682-1500
A,C,CM,E,M,P*

Hewlett-Packard Co.
29 Burlington Mall Rd
BURLINGTON, MA 01803-4514
Tel: (617) 270-7000
C,E

Michigan

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
C,M

Hewlett-Packard Co.
39550 Orchard Hill Place Drive
NOVI, MI 48050
Tel: (313) 349-9200
A,C,E,M

Hewlett-Packard Co.
560 Kirts Rd.
Suite 101
TROY, MI 48084
Tel: (313) 362-5180
C

Minnesota

Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,C,CM,E,M

Missouri

Hewlett-Packard Co.
1001 E. 101st Terrace Suite 120
KANSAS CITY, MO 64131-3368
Tel: (816) 941-0411
A,C,CM,E,M

Hewlett-Packard Co.
13001 Hollenberg Drive
BRIDGETON, MO 63044
Tel: (314) 344-5100
A,C,E,M

Nebraska

Hewlett-Packard
11626 Nicholas St.
OMAHA, NE 68154
Tel: (402) 493-0300
C,E,M

New Jersey

Hewlett-Packard Co.
120 W. Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,C,CM,E,M

Hewlett-Packard Co.
20 New England Av. West
PISCATAWAY, NJ 08854
Tel: (201) 562-6100
A,C,CM,E

New Mexico

Hewlett-Packard Co.
7801 Jefferson N.E.
ALBUQUERQUE, NM 87109
Tel: (505) 823-6100
C,E,M

Hewlett-Packard Co.
1362-C Trinity Dr.
LOS ALAMOS, NM 87544
Tel: (505) 662-6700
C,E

New York

Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
A,C,E,M

Hewlett-Packard Co.
9600 Main Street
CLARENCE, NY 14031
Tel: (716) 759-8621
C,E,M

Hewlett-Packard Co.
200 Cross Keys Office Park
FAIRPORT, NY 14450
Tel: (716) 223-9950
A,C,CM,E,M

Hewlett-Packard Co.
7641 Henry Clay Blvd.
LIVERPOOL, NY 13088
Tel: (315) 451-1820
A,C,CM,E,M

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 7th Avenue
MANHATTAN NY 10119
Tel: (212) 971-0800
C,M*

Hewlett-Packard Co.
15 Myers Corner Rd.
Hollowbrook Park, Suite 2D
WAPPINGERS FALLS, NY 12590
Tel: (914) 298-9125
CM,E

Hewlett-Packard Co.
2975 Westchester Ave
PURCHASE, NY 10577
Tel: (914) 935-6300
C,CM,E

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 682-7800
A,C,CM,E,M

North Carolina

Hewlett-Packard Co.
305 Gregson Dr.
CARY, NC 27511
Tel: (919) 467-6600
C,CM,E,M,P*

Hewlett-Packard Co.
9401 Arrow Point Blvd
Suite 100
CHARLOTTE, NC 28217
Tel: (704) 527-8780
C*

Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27420
Tel: (919) 852-1800
A,C,CM,E,M,P*

Ohio

Hewlett-Packard Co.
2717 S. Arlington Road
AKRON, OH 44312
Tel: (216) 644-2270
C,E

Hewlett-Packard Co.
4501 Erskine Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
C,M

Hewlett-Packard Co.
15885 Sprague Road
CLEVELAND, OH 44136
Tel: (216) 243-7300
A,C,CM,E,M

Hewlett-Packard Co.
9080 Springboro Pike
MIAMISBURG, OH 45342
Tel: (513) 433-2223
A,C,CM,E*,M

Hewlett-Packard Co.
One Maritime Plaza, 5th Floor
720 Water Street
TOLEDO, OH 43604
Tel: (419) 242-2200
C

Hewlett-Packard Co.
675 Brooksedge Blvd.
WESTERVILLE, OH 43081
Tel: (614) 891-3344
C,CM,E*

Oklahoma

Hewlett-Packard Co.
3525 N.W. 56th St.
Suite C-100
OKLAHOMA CITY, OK 73112
Tel: (405) 946-9499
C,E*,M

UNITED STATES (Cont'd)

Hewlett-Packard Co.
6655 South Lewis,
Suite 105
TULSA, OK 74136
Tel: (918) 481-6700
A**,C,E,M*,P*

Oregon

Hewlett-Packard Co.
9255 S. W. Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,C,E*,M

Pennsylvania

Hewlett-Packard Co.
Heatherwood Industrial Park
50 Dorchester Rd.
Route 22
HARRISBURG, PA 17112-2799
Tel: (717) 657-5900
C

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,C,E,M

Hewlett-Packard Co.
2750 Monroe Boulevard
VALLEY FORGE, PA 19482
Tel: (215) 666-9000
A,C,CM,E,M

South Carolina

Hewlett-Packard Co.
Brookside Park, Suite 122
1 Harbison Way
COLUMBIA, SC 29212
Tel: (803) 732-0400
C,M

Hewlett-Packard Co.
545 N. Pleasantburg Dr.
Suite 100
GREENVILLE, SC 29607
Tel: (803) 232-8002
C

Tennessee

Hewlett-Packard Co.
One Energy Centr. Suite 200
Pellissippi Pkwy.
KNOXVILLE, TN 37932
Tel: (615) 966-4747
A,C,E,M,P

Hewlett-Packard Co.
3070 Directors Row
Directors Square
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,C,E,M

Hewlett-Packard Co.
44 Vantage Way,
Suite 160
NASHVILLE, TN 37228
Tel: (615) 255-1271
A,C,E,M,P

Texas

Hewlett-Packard Co.
1826-P Kramer Lane
AUSTIN, TX 78758
Tel: (512) 835-6771
C,E,P*

Hewlett-Packard Co.
5700 Cromo Dr
EL PASO, TX 79912
Tel: (915) 833-4400
C,E*,M**

Hewlett-Packard Co.
3952 Sandshell Drive
FORT WORTH, TX 76137
Tel: (817) 232-9500
C

Hewlett-Packard Co.
10535 Harwin Drive
HOUSTON, TX 77036
Tel: (713) 776-6400
A,C,E,M,P*

Hewlett-Packard Co.
3301 West Royal Lane
IRVING, TX 75063
Tel: (214) 869-3377
C,E

Hewlett-Packard Co.
109 E. Toronto, Suite 100
McALLEN, TX 78501
Tel: (512) 630-3030
C

Hewlett-Packard Co.
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,C,CM,E,M,P*

Hewlett-Packard Co.
1020 Central Parkway South
SAN ANTONIO, TX 78232
Tel: (512) 494-9336
A,C,E,M,P*

Utah

Hewlett-Packard Co.
3530 W. 2100 South St.
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,C,E,M

Virginia

Hewlett-Packard Co.
840 Greenbrier Circle
Suite 101
CHESAPEAKE, VA 23320
Tel: (804) 424-7105
C,E,M

Hewlett-Packard Co.
4305 Cox Road
GLEN ALLEN, VA 23060
Tel: (804) 747-7750
A,C,E,M,P*

Hewlett-Packard Co.
Tanglewood West Bldg.
Suite 240
3959 Electric Road
ROANOKE, VA 24018
Tel: (703) 774-3444
C,E,P

Washington

Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4004
A,C,CM,E,M

Hewlett-Packard Co.
1225 Argonne Rd
SPOKANE, WA 99212
Tel: (509) 922-7000
C

West Virginia

Hewlett-Packard Co.
501 56th Street
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,C,M

Wisconsin

Hewlett-Packard Co.
275 N. Corporate Dr.
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,C,E*,M

URUGUAY

Pablo Ferrando S.A.C. e I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 59-82-802-586
Telex: 398802586
A,CM,E,M

Olympia de Uruguay S.A.
Maquines de Oficina
Avda. del Libertador 1997
Casilla de Correos 6644
MONTEVIDEO
Tel: 91-1809, 98-3807
Telex: 6342 OROU UY
P

VENEZUELA

Hewlett-Packard de Venezuela C.A.
3A Transversal Los Ruices Norte
Edificio Segre 2 & 3
Apartado 50933
CARACAS 1050
Tel: (582) 239-4133
Telex: 251046 HEWPACK
A,C,CM,E,M,P

Hewlett-Packard de Venezuela, C.A.
Centro Ciudad Comercial Tamanaco
Nivel C-2 (Nueva Etapa)
Local 53H05
Chuao, **CARACAS**
Tel: 928291
P

Albis Venezolana S.R.L.
Av. Las Marias, Ota. Alix,
El Pedregal
Apartado 81025
CARACAS 1080A
Tel: 747984, 742146
Telex: 24009 ALBIS VC
A

Tecnologica Medica del Caribe, C.A.
Multicentro Empresarial del Este
Ave. Libertador
Edif. Libertador
Nucleo "C" - Oficina 51-52
CARACAS
Tel: 339867/333780
M

Hewlett-Packard de Venezuela C.A.
Residencias Tia Betty Local 1
Avenida 3 y con Calle 75
MARACAIBO, Estado Zulia
Apartado 2646
Tel: 58-2-617-5669
Telex: 62464 HPMAR
C,E*

Hewlett-Packard de Venezuela C.A.
Urb. Lomas de Este
Torre Trebol — Piso 11
VALENCIA, Estado Carabobo
Apartado 3347
Tel: (5841) 222992
C,P

YUGOSLAVIA

Do Hermes
General Zdanova 4
YU-11000 **BEOGRAD**
Tel: (011) 342 641
Telex: 11433
A,C,E,M,P

Do Hermes
Celovska 73
YU-61000 **LJUBLJANA**
Tel: (061) 553 170
Telex: 31583
A,C,E,M,P

Elektrotehna
Titova 51
YU-61000 **LJUBLJANA**
CM

Do Hermes
Kralja Tomislava 1
YU-71000 **SARAJEVO**
Tel: (071) 35 859
Telex: 41634
C**,P

ZAIRE

Computer & Industrial Engineering
25, Avenue de la Justice
B.P. 12797
KINSHASA, Gombe
Tel: 32063
Telex: 21552
C,P

ZAMBIA

R.J. Tilbury (Zambia) Ltd.
P.O. Box 32792
LUSAKA
Tel: 215590
Telex: 40128
E

ZIMBABWE

Field Technical Sales (Private) Limited
45, Kelvin Road North
P.O. Box 3458
SALISBURY
Tel: 705 231
Telex: 4-122 RH
E,P

September 1987

Part No. 5958-8581
Printed in U.S.A. 07/88
E0788

HEWLETT PACKARD